

Università
degli Studi
di Palermo

d[i]
dipartimento
di ingegneria
unipa

PROGETTO FINALE

Prof. Rosario Sorbello

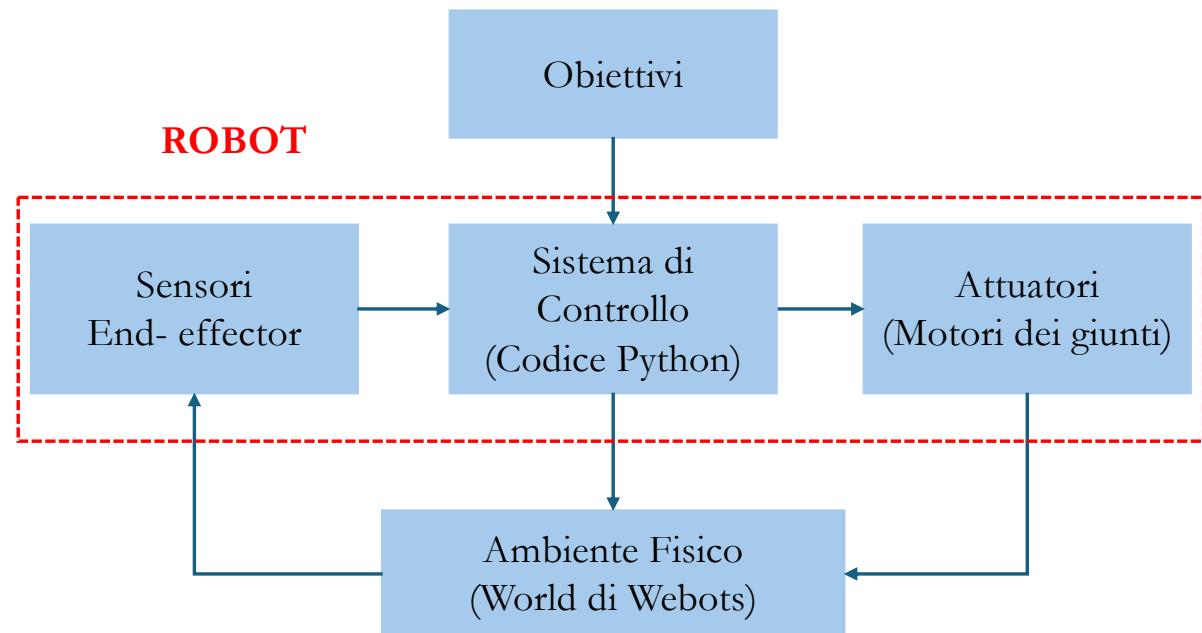


ROBOT: UN SISTEMA AUTONOMO

COS'È UN ROBOT?

Un robot è un sistema **autonomo** che esiste nel **mondo fisico**, interagisce con l'ambiente in cui si trova e prende decisioni per raggiungere un obiettivo.

Le azioni del robot derivano da un **sistema di controllo** che percepisce l'ambiente esterno tramite dei **sensori** e restituisce in uscita delle azioni in grado di modificare l'ambiente fisico attraverso degli **attuatori**.



SIMULATORE VIRTUALE: WEBOTS

La robotica è la scienza che si occupa dello studio, della progettazione e della realizzazione di un robot. Esistono dei simulatori virtuali che permettono di simulare, programmare e modellare robot reali.

WEBOTS

Simulatore di robotica professionale
per robot in ambienti 3D realistici



- Supporta vari tipi di robot, sensori ed attuatori
- Permette la realizzazione di ambienti complessi e dinamici



Webots
robot simulation

PERCHÉ USARE WEBOTS?

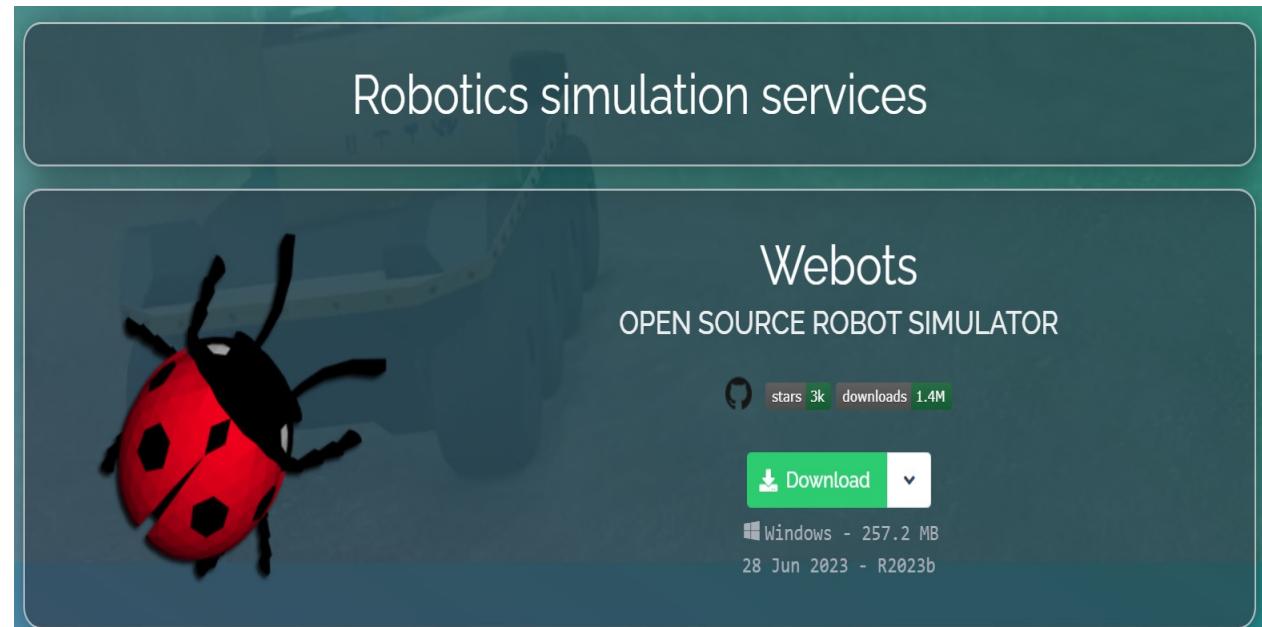
Facilita la prototipazione rapida e sicura,
riducendo costi e tempi di sviluppo.



WEBOTS

COME SCARICARE IL SOFTWARE:

<https://cyberbotics.com/#download>



Get Started

New to Webots? Get started now:

1. [Download Webots.](#)
2. [Install it.](#)
3. [Start it.](#)
4. Visit the Webots Guided Tour from the Help menu of Webots.
5. Follow the [Webots tutorials](#).
6. Explore [examples](#) and create your own simulation from them.

PROGRAMMI DI SUPPORTO

Anaconda

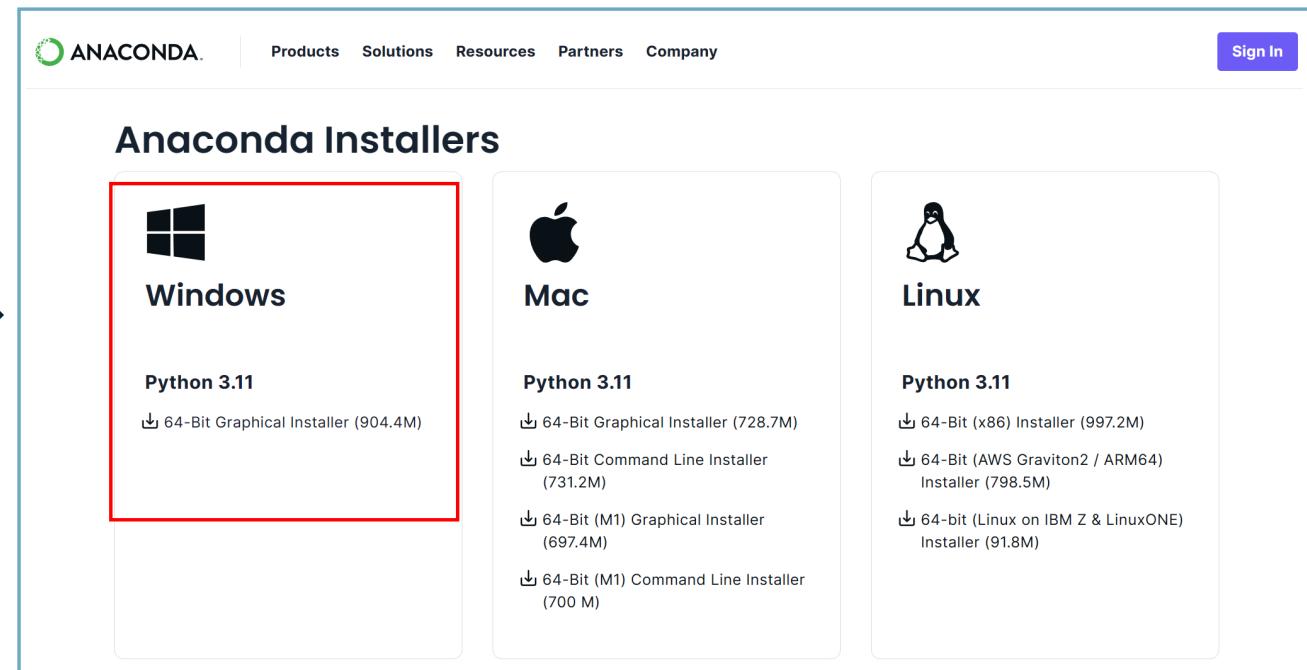
Python

```
import sys
sys.path.append('/opt/anaconda3/lib/python3.
11/site-packages/cv2')

import cv2
import numpy as np
from controller import Supervisor
```

ANACONDA

<https://www.anaconda.com/download>



The screenshot shows the Anaconda website's 'Anaconda Installers' page. At the top, there is a navigation bar with links for Products, Solutions, Resources, Partners, and Company, along with a 'Sign In' button. Below the navigation bar, the title 'Anaconda Installers' is displayed. There are three main sections: 'Windows' (highlighted with a red box), 'Mac', and 'Linux'. Each section contains a logo, the Python version, and a list of installers.

Platform	Python Version	Installers
Windows	3.11	64-Bit Graphical Installer (904.4M)
Mac	3.11	64-Bit Graphical Installer (728.7M) 64-Bit Command Line Installer (731.2M) 64-Bit (M1) Graphical Installer (697.4M) 64-Bit (M1) Command Line Installer (700 M)
Linux	3.11	64-Bit (x86) Installer (997.2M) 64-Bit (AWS Graviton2 / ARM64) Installer (798.5M) 64-bit (Linux on IBM Z & LinuxONE) Installer (91.8M)

PYTHON

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



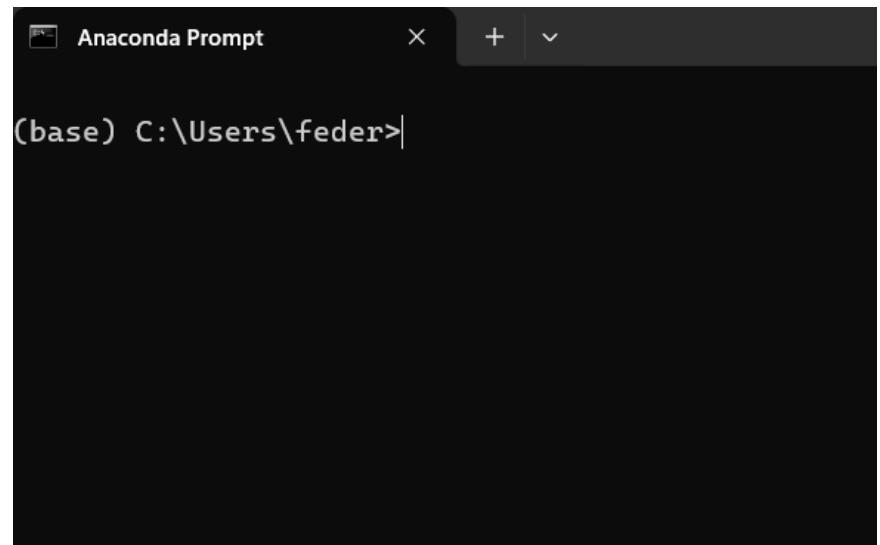
The screenshot shows the Python website's Downloads section. At the top, there is a yellow button labeled "Download Python 3.12.3". Below it, text says "Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [macOS](#), [Other](#)". Further down, it says "Want to help test development versions of Python 3.13? [Prereleases](#), [Docker images](#)". To the right of the text, there is a cartoon illustration of two boxes hanging from parachutes against a blue background with white clouds.

INSTALLAZIONE LIBRERIE

LIBRERIE DA UTILIZZARE
PER IL CODICE PYTHON:

1. OPENCV-PYTHON
2. NUMPY
3. IKPY

Attraverso prompt di anaconda



Anaconda Prompt

(base) C:\Users\feder>

INSTALLAZIONE OPENCV-PYTHON

Comando

pip3 install opencv-python

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

Installa la versione più recente di PowerShell per nuove funzionalità e miglioramenti. https://aka.ms/PSWindows

PS C:\Users\isabe> pip3 install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.9.0.80-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting numpy>=1.21.2 (from opencv-python)
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
                                             61.0/61.0 kB 232.0 kB/s eta 0:00:00
  Downloading opencv_python-4.9.0.80-cp37-abi3-win_amd64.whl (38.6 MB)
                                             38.6/38.6 MB 470.3 kB/s eta 0:00:00
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
                                             15.5/15.5 MB 1.3 MB/s eta 0:00:00
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.26.4 opencv-python-4.9.0.80
PS C:\Users\isabe> pip3 install opencv-python
Requirement already satisfied: opencv-python in c:\users\isabe\appdata\local\programs\python\python312\lib\site-packages
(4.9.0.80)
Requirement already satisfied: numpy>=1.21.2 in c:\users\isabe\appdata\local\programs\python\python312\lib\site-packages
(from opencv-python) (1.26.4)
```

Verifica corretta installazione

Percorso file in cui è avvenuta l'installazione

INSTALLAZIONE OPENCV-PYTHON

Comando

pip3 install opencv-python

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

Installa la versione più recente di PowerShell per nuove funzionalità e miglioramenti. https://aka.ms/PSWindows

PS C:\Users\isabe> pip3 install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.9.0.80-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting numpy>=1.21.2 (from opencv-python)
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
                                             61.0/61.0 kB 232.0 kB/s eta 0:00:00
  Downloading opencv_python-4.9.0.80-cp37-abi3-win_amd64.whl (38.6 MB)
                                             38.6/38.6 MB 470.3 kB/s eta 0:00:00
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
                                             15.5/15.5 MB 1.3 MB/s eta 0:00:00
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.26.4 opencv-python-4.9.0.80
PS C:\Users\isabe> pip3 install opencv-python
Requirement already satisfied: opencv-python in c:\users\isabe\appdata\local\programs\python\python312\lib\site-packages
(4.9.0.80)
Requirement already satisfied: numpy>=1.21.2 in c:\users\isabe\appdata\local\programs\python\python312\lib\site-packages
(from opencv-python) (1.26.4)
```

Verifica corretta installazione

Percorso file in cui è avvenuta l'installazione

INSTALLAZIONE IKPY

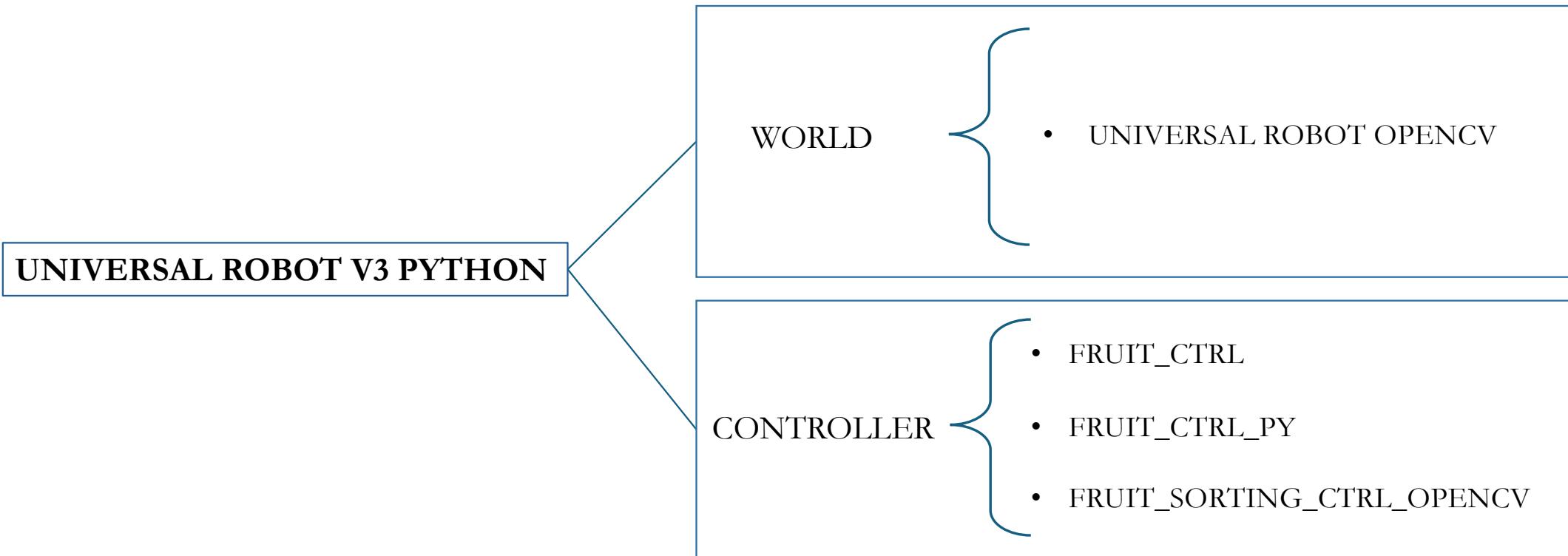
Comando → pip install ikpy

```
C:\Users\aless>pip install ikpy
Collecting ikpy
  Downloading ikpy-3.3.4-py3-none-any.whl.metadata (4.6 kB)
Requirement already satisfied: numpy in c:\users\aless\appdata\local\programs\python\python312\lib\site-packages (from ikpy) (1.26.4)
Requirement already satisfied: scipy in c:\users\aless\appdata\local\programs\python\python312\lib\site-packages (from ikpy) (1.13.0)
Collecting sympy (from ikpy)
  Downloading sympy-1.12-py3-none-any.whl.metadata (12 kB)
Collecting mpmath>=0.19 (from sympy->ikpy)
  Downloading mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Downloading ikpy-3.3.4-py3-none-any.whl (24 kB)
Downloading sympy-1.12-py3-none-any.whl (5.7 MB)
  5.7/5.7 MB 321.2 kB/s eta 0:00:00
Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
  536.2/536.2 kB 218.6 kB/s eta 0:00:00
Installing collected packages: mpmath, sympy, ikpy
(base) C:\Users\aless>pip install ikpy
Requirement already satisfied: ikpy in c:\users\aless\anaconda3\lib\site-packages (3.3.4)
Requirement already satisfied: numpy in c:\users\aless\anaconda3\lib\site-packages (from ikpy) (1.24.3)
Requirement already satisfied: scipy in c:\users\aless\anaconda3\lib\site-packages (from ikpy) (1.11.1)
Requirement already satisfied: sympy in c:\users\aless\anaconda3\lib\site-packages (from ikpy) (1.11.1)
Requirement already satisfied: mpmath>=0.19 in c:\users\aless\anaconda3\lib\site-packages (from sympy->ikpy) (1.3.0)
```

Verifica corretta installazione

Percorso file in cui è avvenuta l'installazione

UNIVERSAL ROBOT V3 PYTHON

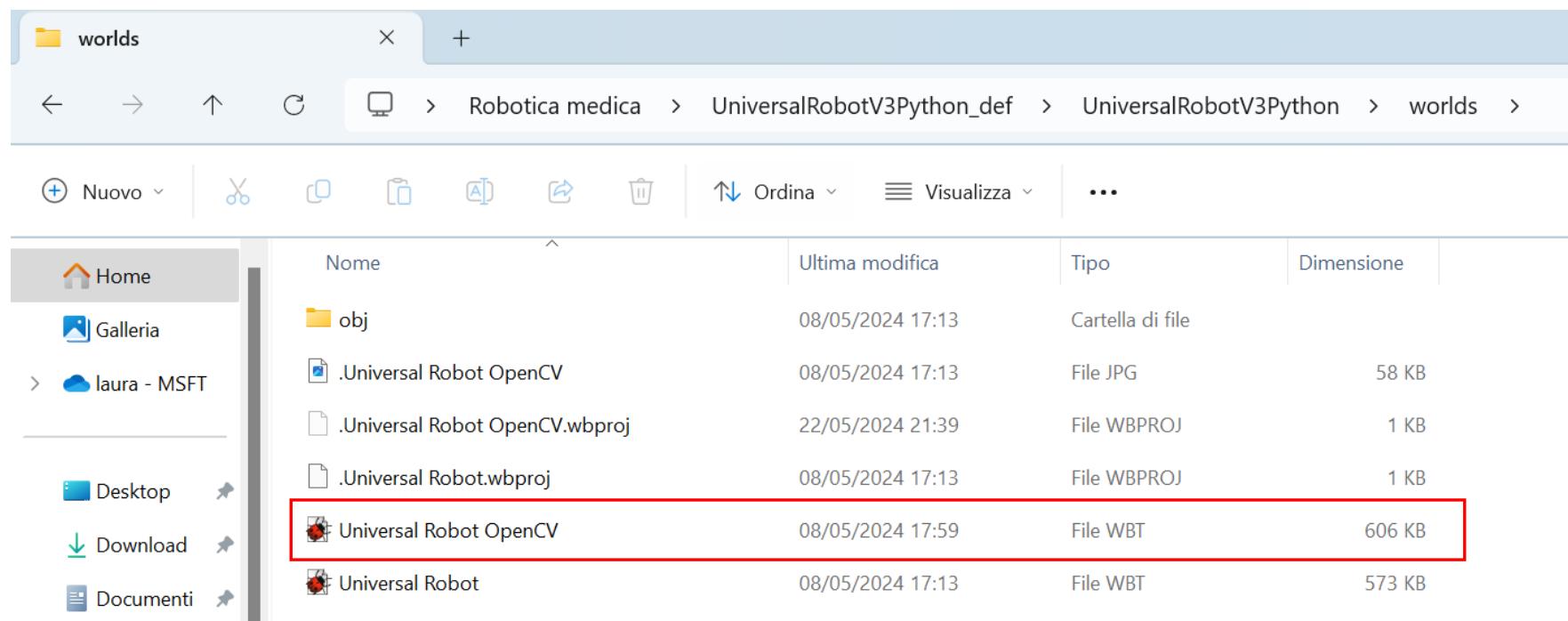


WORLD

Un **world** in Webots è un file che descrive l'ambiente in cui i robot operano (il terreno, gli oggetti, i robot stessi, le luci, le telecamere)

File WEBOTS

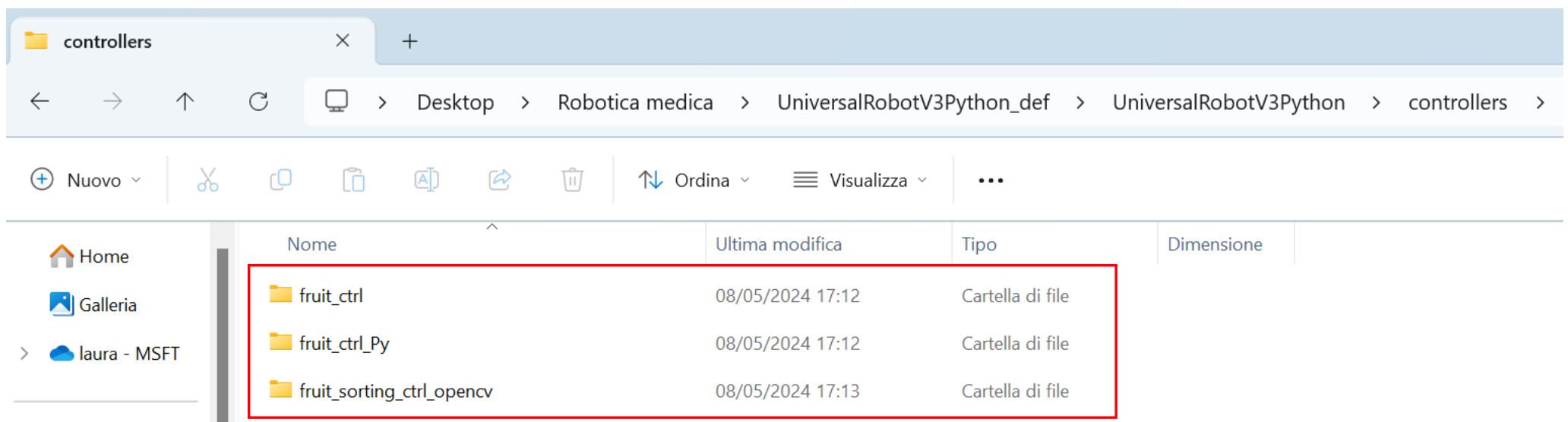
UNIVERSAL ROBOT OPENCV.wbt



CONTROLLER

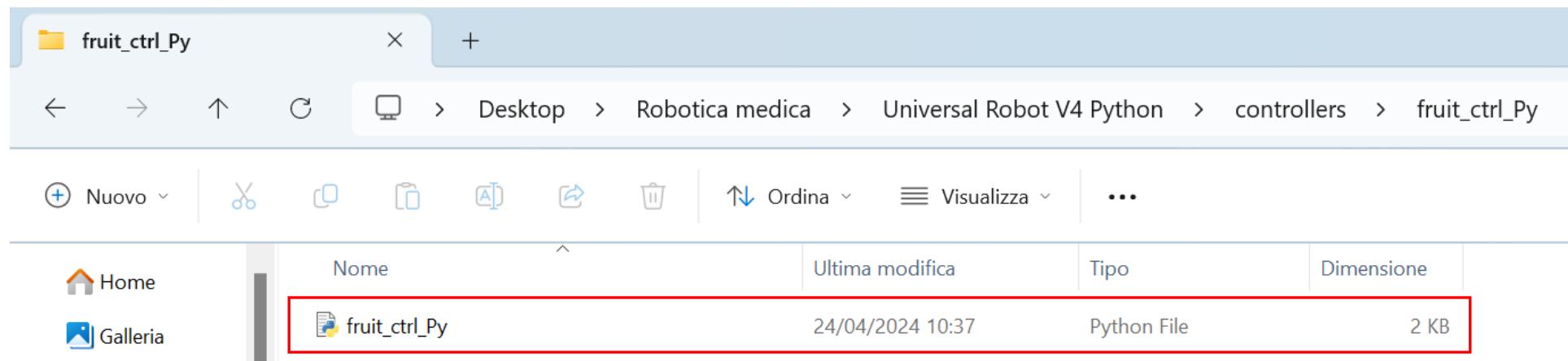
Un **controller** in Webots è un programma software che controlla il comportamento di un robot all'interno della simulazione. È essenzialmente il "cervello" del robot.

- I controller possono essere scritti in diversi linguaggi di programmazione supportati da Webots, tra cui C, C++, Python, Java, MATLAB.
- Il controller riceve dati dai sensori del robot, elabora queste informazioni e invia comandi agli attuatori per controllare i movimenti e le azioni del robot.



FRUIT_CTRL_PY

Generazione di frutti virtuali e posizionamento (orange, apple, rotten-apple) in una posizione ben specifica



FRUIT_CTRL_PY

```
"""fruit_ctrl_Py controller."""
```

```
from controller import Supervisor  
from controller import Node  
from controller import Field  
import random
```

```
random.seed()
```

```
robot = Supervisor()
```

```
timestep = 64
```

```
i = 0  
j = 8  
k = 8  
l = 8  
fr = 1  
max = 50
```

```
# Initialize camera  
camera = robot.getDevice('camera')  
camera.enable(timestep)
```

```
fruit_initial_translation = [0.570002, 2.85005, 0.349962]
```



Importazione delle Librerie

Impostazione del Timestep della simulazione



Inizializzazione della Camera

FRUIT_CTRL_PY

```
# Main Loop:  
while robot.step(timestep) != -1:
```

```
    if robot.getTime() > 7.5:
```

```
        if i == 0:
```

```
            if fr > 0:
```

```
                fr = int(random.choice([1,2,3]))
```

```
                if l == max: fr = 3
```

```
                if j == max: fr = 2
```

```
                if k == max: fr = 1
```

```
                if fr == 1 and j < max:
```

```
                    name = f'apple{j:d}'
```

```
                    j += 1
```

```
                if fr == 2 and k < max:
```

```
                    name = f'orange{k:d}'
```

```
                    k += 1
```

```
                if fr == 3 and l < max:
```

```
                    name = f'rottenapple{l:d}'
```

```
                    l += 1
```

```
                fruit = robot.getFromDef(name)
```

```
                fruit_trans_field = Node.getField(fruit, 'translation')
```

```
                Field.setSFVec3f(fruit_trans_field, fruit_initial_translation)
```

```
                if j == max and k == max and l == max: fr = 0
```

```
                i += 1
```

```
            if i == 120: i = 0
```

```
pass
```

```
    if i == 0:
```

Dopo 7.5 secondi inizia a generare frutta

```
        if fr > 0:
```

```
            if l == max: fr = 3
```

```
            if j == max: fr = 2
```

```
            if k == max: fr = 1
```

```
            if fr == 1 and j < max:
```

```
                name = f'apple{j:d}'
```

```
                j += 1
```

```
            if fr == 2 and k < max:
```

```
                name = f'orange{k:d}'
```

```
                k += 1
```

```
            if fr == 3 and l < max:
```

```
                name = f'rottenapple{l:d}'
```

```
                l += 1
```

```
            fruit = robot.getFromDef(name)
```

```
            fruit_trans_field = Node.getField(fruit, 'translation')
```

```
            Field.setSFVec3f(fruit_trans_field, fruit_initial_translation)
```

```
            if j == max and k == max and l == max: fr = 0
```

```
            i += 1
```

```
        if i == 120: i = 0
```

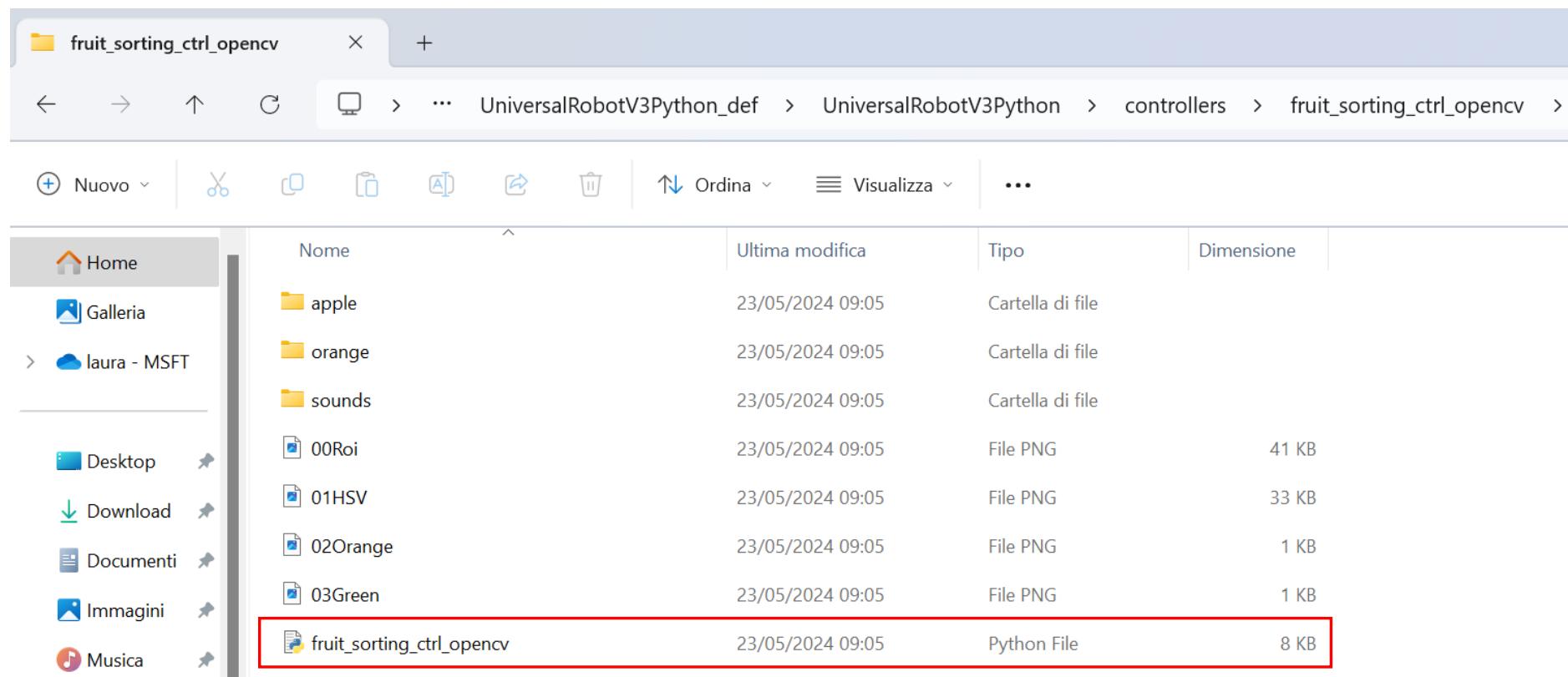
```
    pass
```

Sceglie in maniera randomica il tipo di frutta da generare; impostando il tipo di frutta da generare in base alla tipologia, raggiunto il massimo

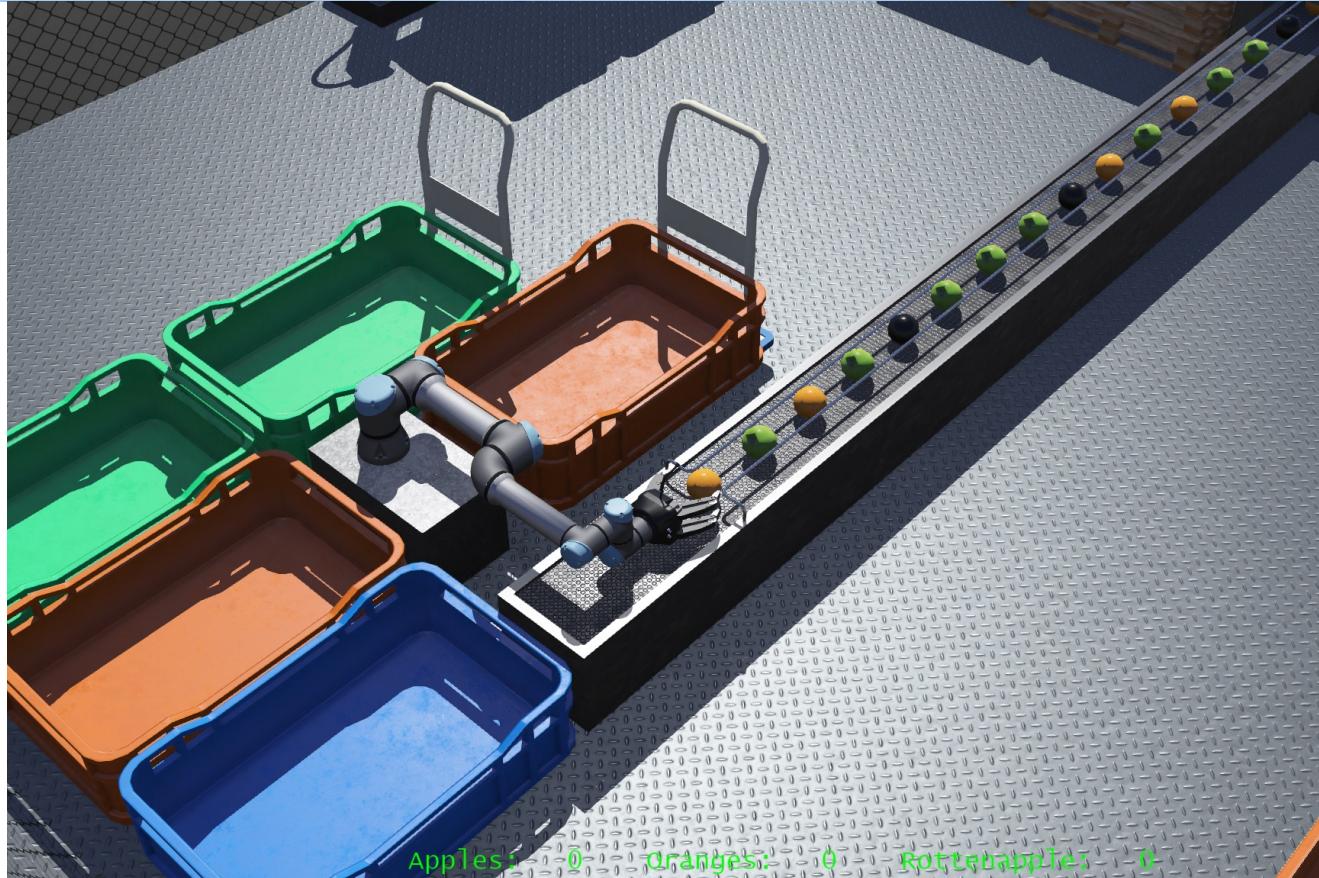
Se ha raggiunto il numero massimo di frutta da generare non ne genera più

FRUIT_SORTING_CTRL_OPENCV

Sistema di controllo → fruit_sorting_ctrl_opencv.py



WEBOTS SIMULATION VIEW



Apples: 0 Oranges: 0 Rottenapple: 0

```
INFO: fruit_sorting_ctrl_opencv: Starting controller: python.exe -u fruit_sorting_ctrl_opencv.py
INFO: conveyor_belt: Starting controller: "C:\Program Files\Webots\projects\objects\factory\conveyors\controllers\conveyor_belt\conveyor_belt.exe" 0.1 900
INFO: fruit_ctrl_Py: Starting controller: python.exe -u fruit_ctrl_Py.py
```

UNIVERSAL ROBOTS UR5e

Il **UR5e** è un robot collaborativo, noto per la sua versatilità, facilità d'uso e capacità di lavorare in stretta collaborazione con gli esseri umani senza la necessità di barriere di sicurezza. Presenta 6 gradi di libertà che gli permettono una grande flessibilità di movimento. Data la presenza dei **6 DOF** viene anche denominato «manipolatore diretto».

Le parti principali che si possono distinguere in un robot sono essenzialmente 2:

- 1. MANIPOLATORE**
- 2. SISTEMA DI PROGRAMMAZIONE E CONTROLLO**

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

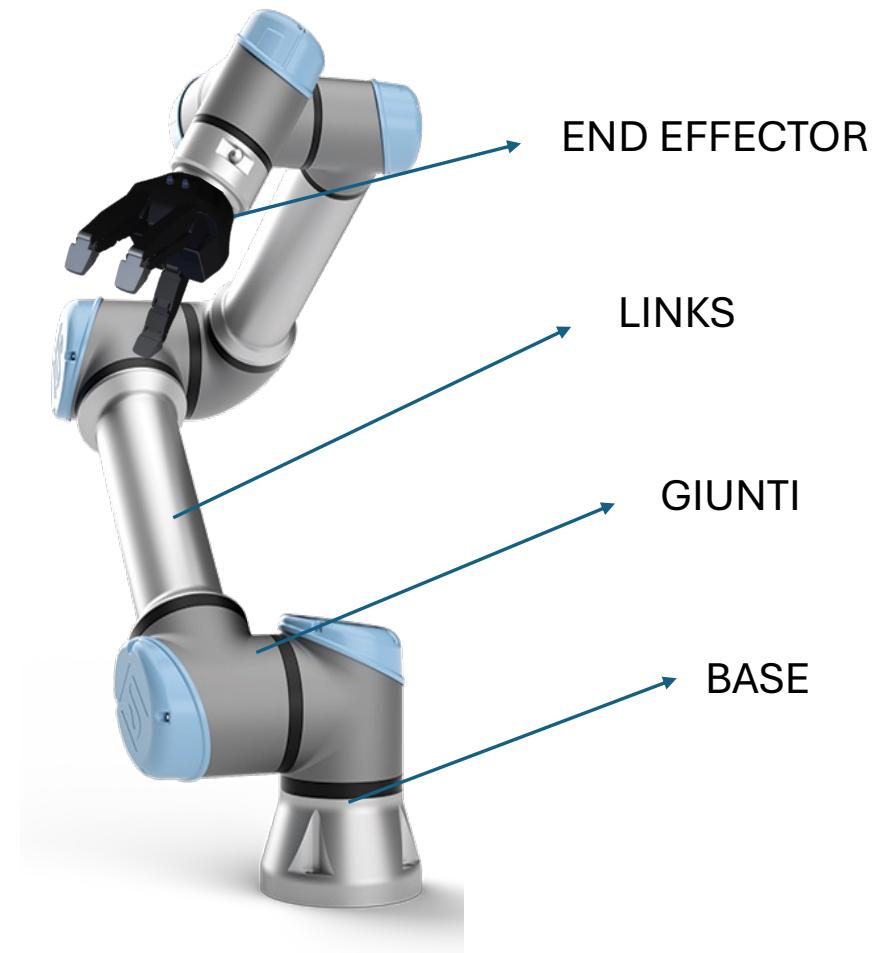


MANIPOLATORE

È la parte meccanica vera e propria ed è costituito dai seguenti componenti:

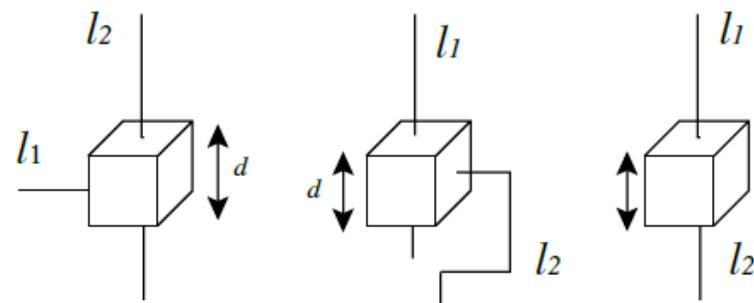
- **Base:** fissata nell'ambiente di lavoro
- **Links:** corpi rigidi di collegamento
- **Giunti:** snodi che connettono i links
- **End-effector:** l'organo terminale, connesso al manipolatore tramite un polso, che gli permette di muoversi liberamente con un orientamento arbitrario

CONFIGURAZIONE ANTROPOMORFA

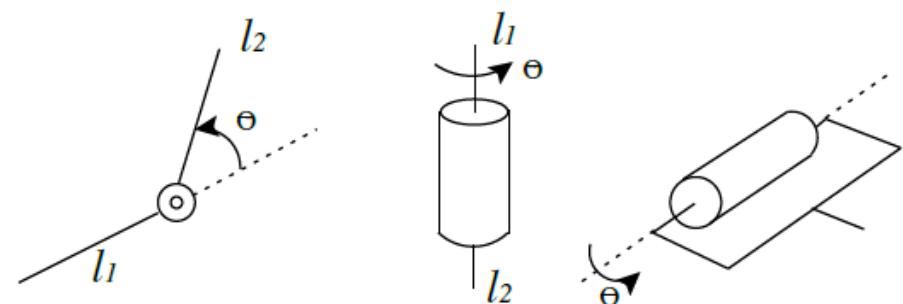


GIUNTI

Si possono distinguere fondamentalmente due tipi di giunti, **rotoidali e prismatici**, ogni giunto realizza l'interconnessione fra due collegamenti (links).



Giunti prismatici



Giunti rotoidali

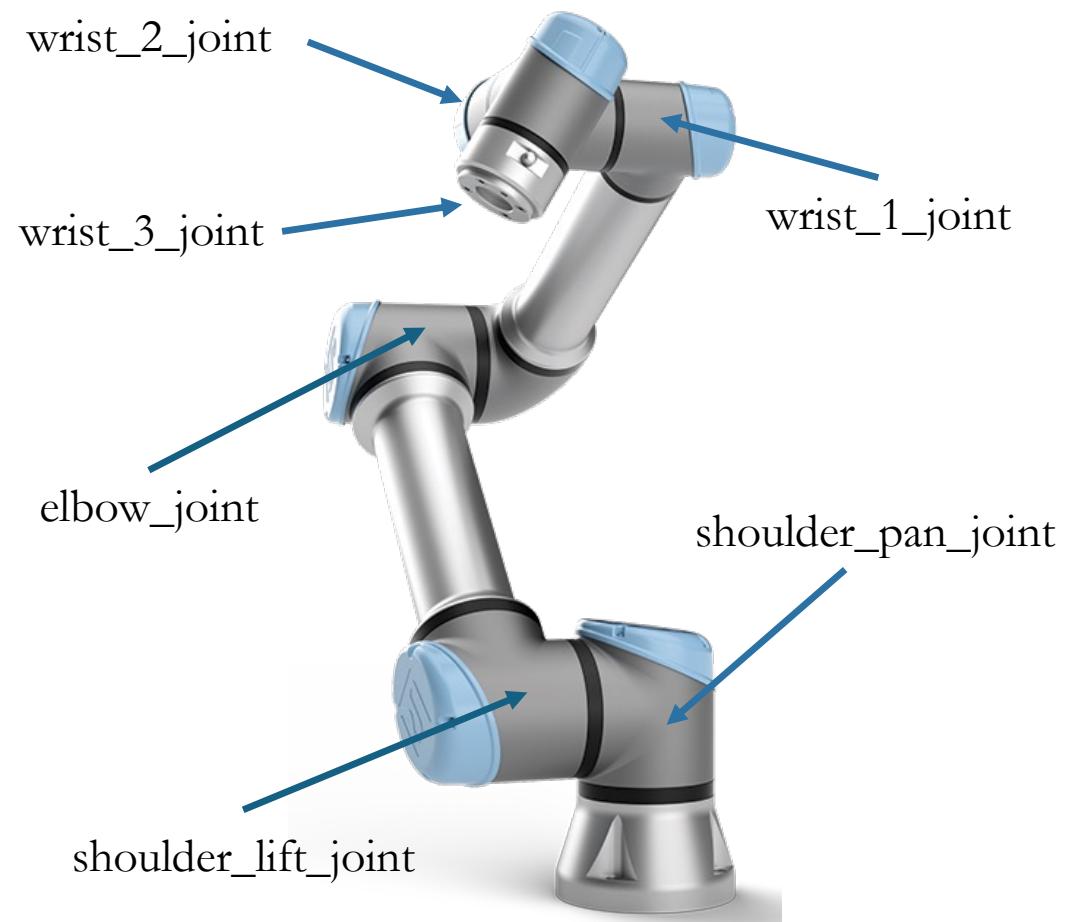
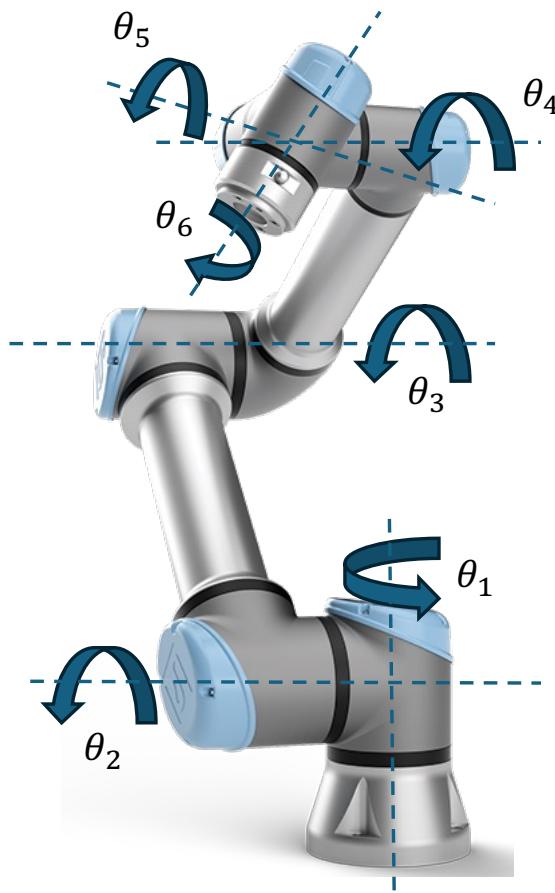
Le variabili dei giunti che rappresentano il relativo spostamento tra i links sono:

$\theta \rightarrow$ Se il giunto è rotoidale

$d \rightarrow$ Se il giunto è prismatico

GIUNTI ROTOIDALI

Identifichiamo graficamente gli assi di rotazione dei 6 giunti



END-EFFECTOR

L' **End-effector** è il dispositivo con cui il manipolatore può interagire con l'ambiente.

Compiti svolti dall'end-effector del robot

1 Afferrare un oggetto

2 Rilasciare l'oggetto in una posizione ben definita



Dispositivo molto specializzato



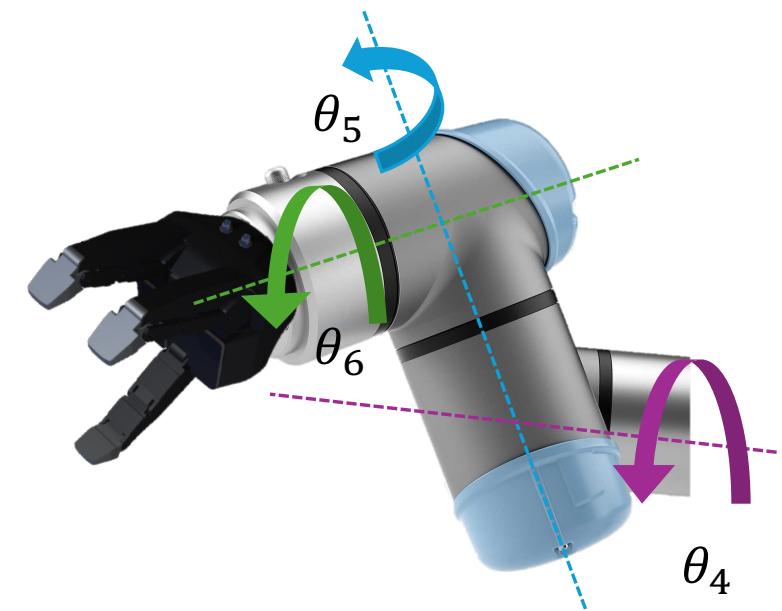
“poco flessibile”

POLSO

Dispositivo collegato all'estremità mobile del manipolatore per orientare l'end-effector nello spazio di lavoro.

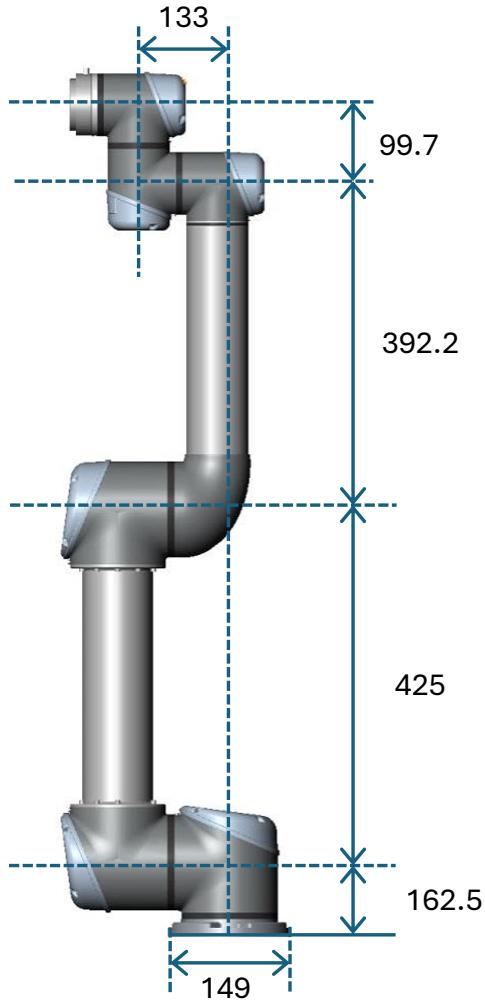
Deve possedere almeno **3 DOF** realizzati da giunti di tipo rotoidale.

Essendo la parte terminale del manipolatore, sono quindi richieste caratteristiche di compattezza che ne rendono difficoltosa la progettazione.



ATTENZIONE ALLE CONFIGURAZIONI SINGOLARI

QUOTATURA UR5e



FONDAMENTI TEORICI PER UR5e

Per poter dire come un robot si è mosso da un punto **p1** espresso nel sistema di riferimento **S1** verso un punto **p0**, espresso nel sistema di riferimento **S0**, è necessario poter relazionare matematicamente i punti del sistema di riferimento S1 con quelli di S0 attraverso:

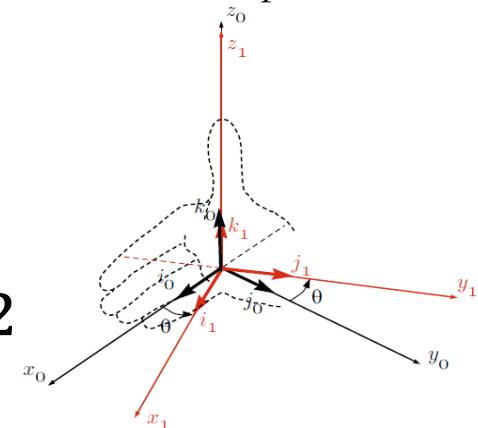
- ROTAZIONI
 - Matrice di rotazione
 - Regola della mano destra
 - Composizione di rotazioni
 - TRASFORMAZIONI OMOGENEE
 - Trasformazione di coordinate
 - Matrice di trasformazione omogenea
 - CINEMATICA DIRETTA
 - Utilizza le trasformazioni omogenee per determinare la posizione finale partendo dalle coordinate articolari.
 - CINEMATICA INVERSA
 - Utilizza le trasformazioni omogenee per risolvere il problema inverso: determinare le coordinate articolari necessarie per ottenere una specifica posizione finale.

$p_0 = R_0^1 p_1$

$p_0 = R_0^1 R_2^1 p_2$

$p_0 = R_0^1 p_1 + d_0^1$

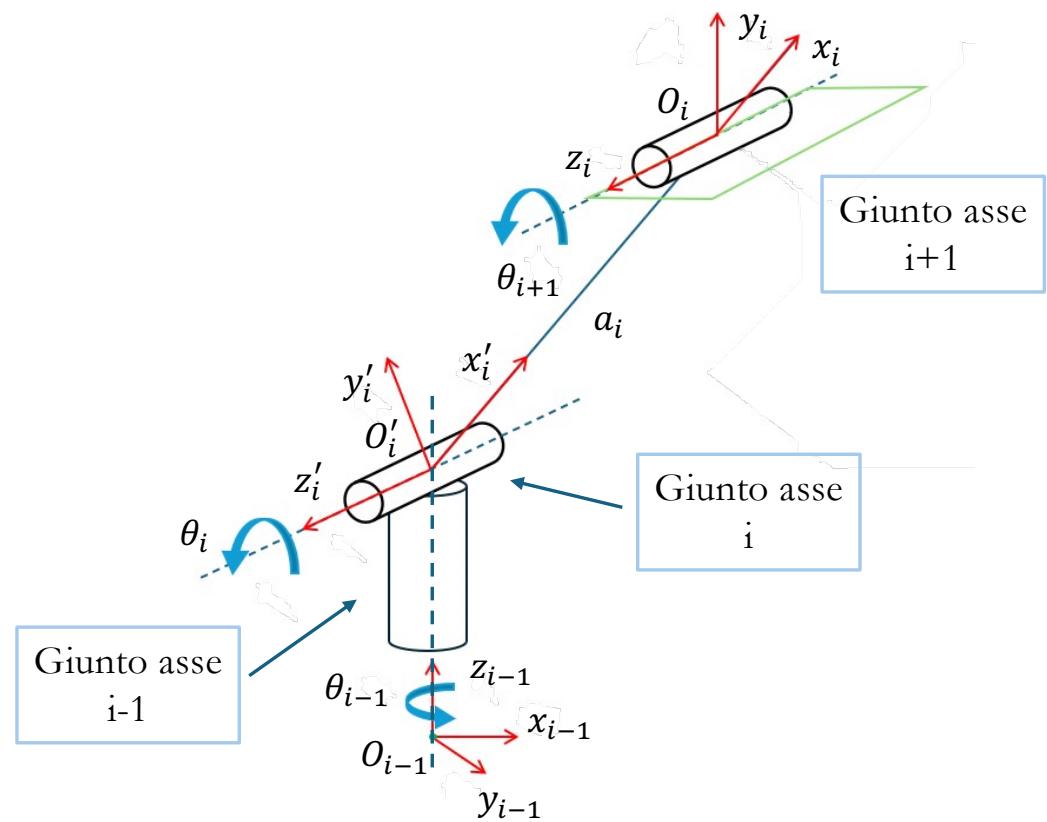
$T \triangleq \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}$



CINEMATICA DIRETTA – CONVENZIONE DI DENAVIT-HARTEMBERG

Detto i l'asse del giunto che connette il braccio $i-1$ al braccio i , per definire la **terna** i (solidale al braccio i) si opera secondo la cosiddetta Convenzione di Denavit-Hatemberg:

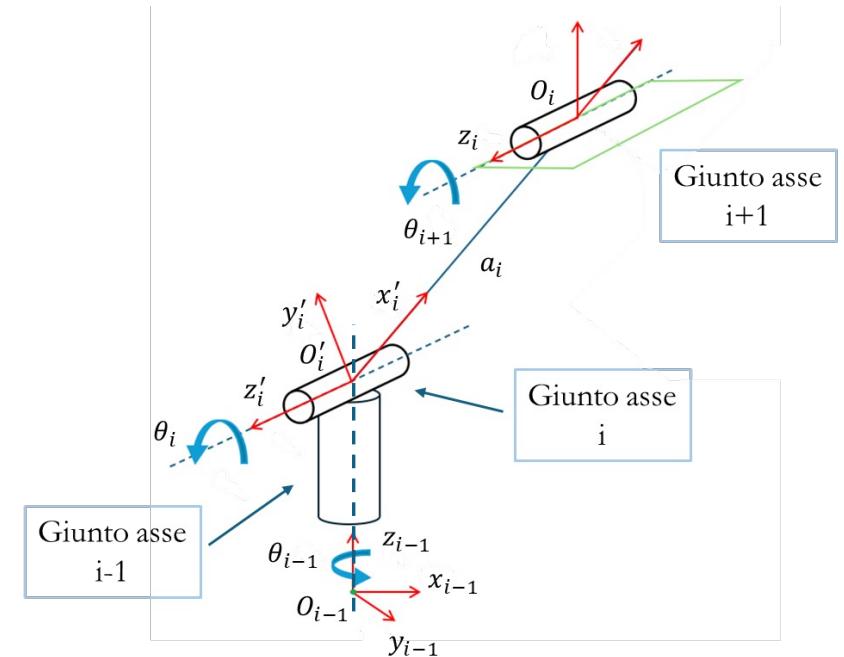
1. Scegliere l'asse z_i lungo l'asse del giunto $i+1$;
2. Si individua O_i come l'intersezione dell'asse z_i con la normale comune agli assi z_{i-1} e z_i e con O'_i si indica l'intersezione della normale comune con z_{i-1} ;
3. Scegliere l'asse x_i diretto lungo la normale comune agli assi z_{i-1} e z_i con verso considerato positivo se va dal giunto i al giunto $i + 1$;
4. Scegliere l'asse y_i ortogonalmente a quelli precedentemente ricavati.



CINEMATICA DIRETTA – CONVENZIONE DI DENAVIT-HARTEMBERG

Una volta definite le terne solidali ai bracci, la posizione e l'orientamento della terna i rispetto alla terna $i-1$, sono completamente definiti anche i seguenti parametri:

- a_i distanza di O_i da O'_i
- d_i coordinata su z_{i-1} di O'_i
- α_i angolo intorno all'asse x_i tra l'asse z_{i-1} e l'asse z_i
valutato positivo in senso antiorario
- θ_i angolo intorno all'asse z_{i-1} tra l'asse x_{i-1} e l'asse x_i
valutato positivo in senso antiorario

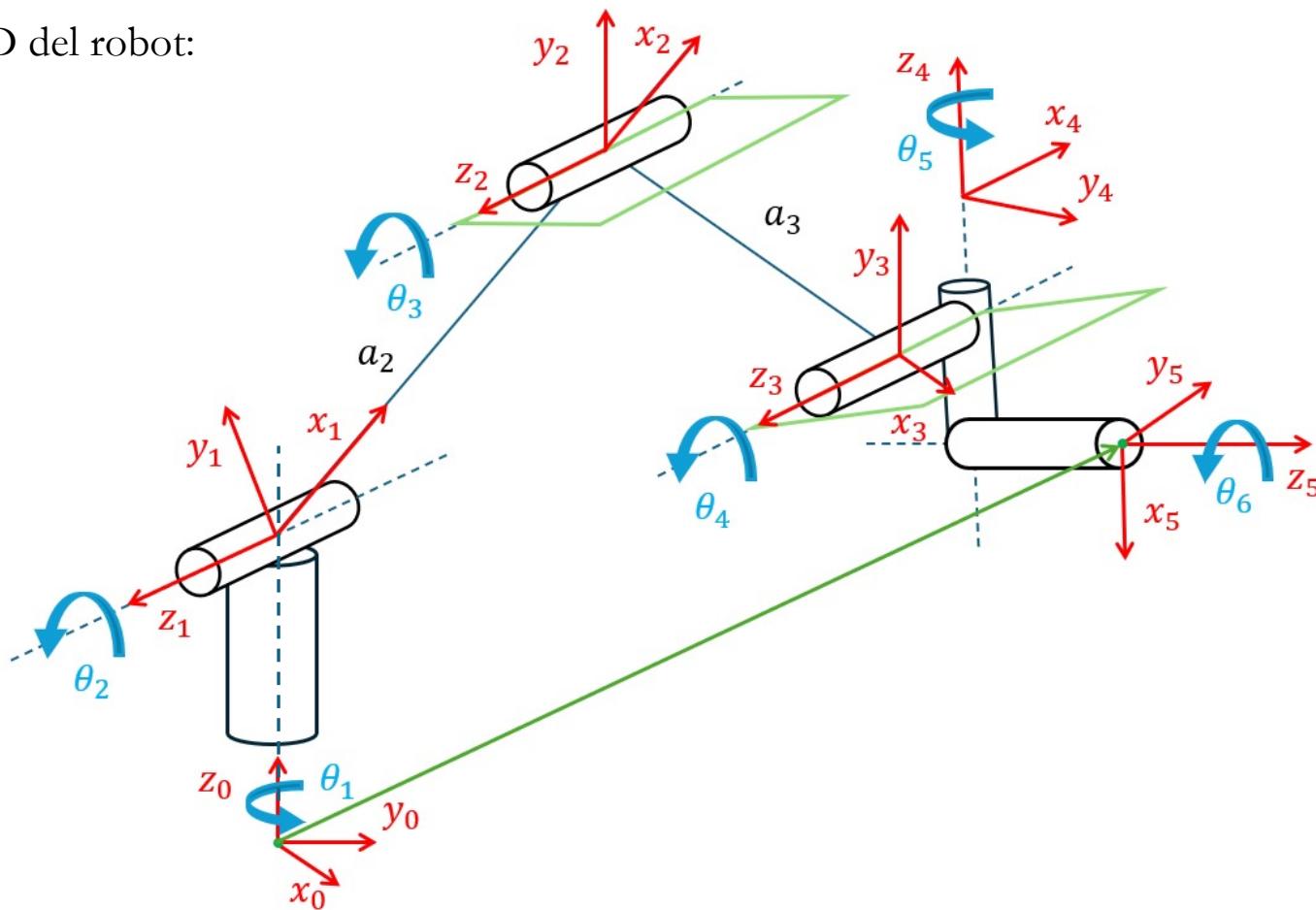


**MATRICE DI TRASFORMAZIONE
OMOGENEA**
(Convenzione di Denavit-Hatemberg)

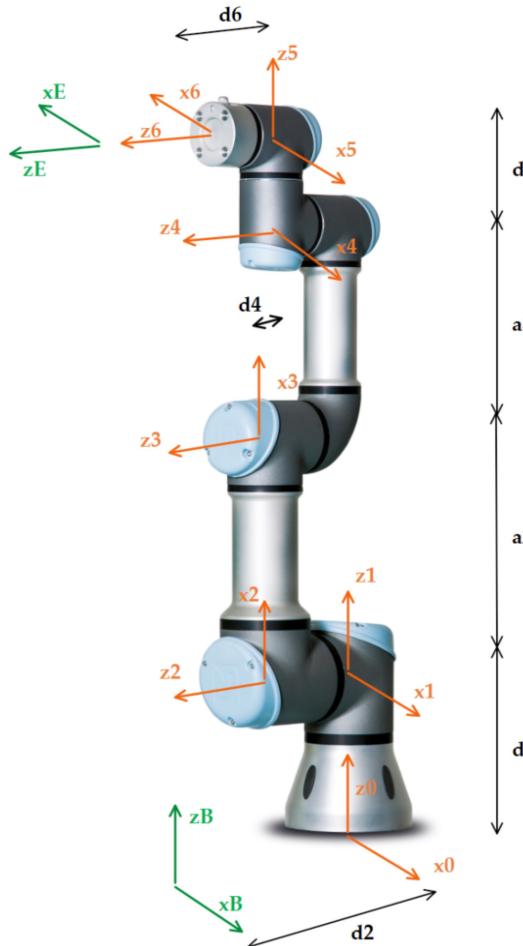
$$A_{i-1}^i(q_i) = A_{i-1}^{i'} A_{i'}^i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

CINEMATICA DIRETTA – UR5e TRIDIMENSIONALE

Rappresentazione 3D del robot:



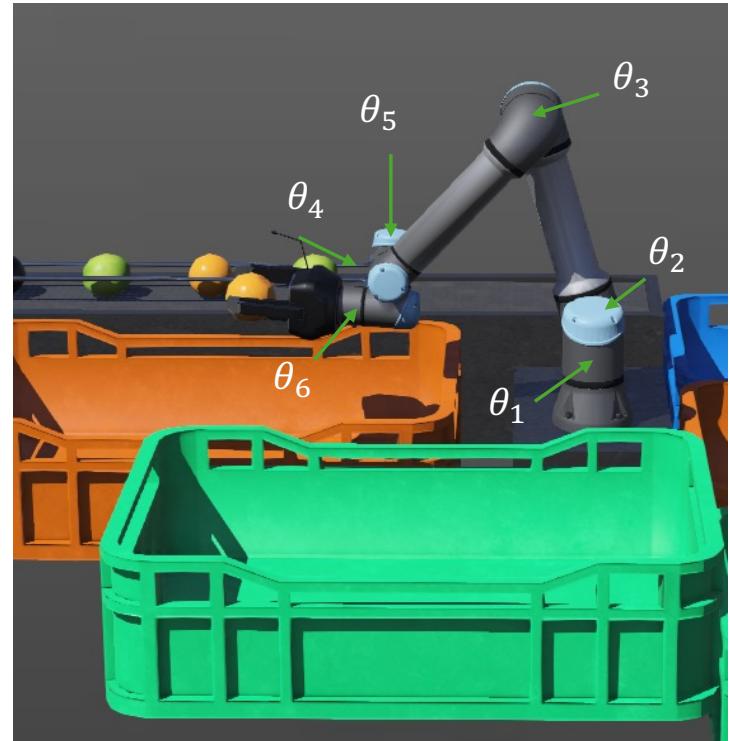
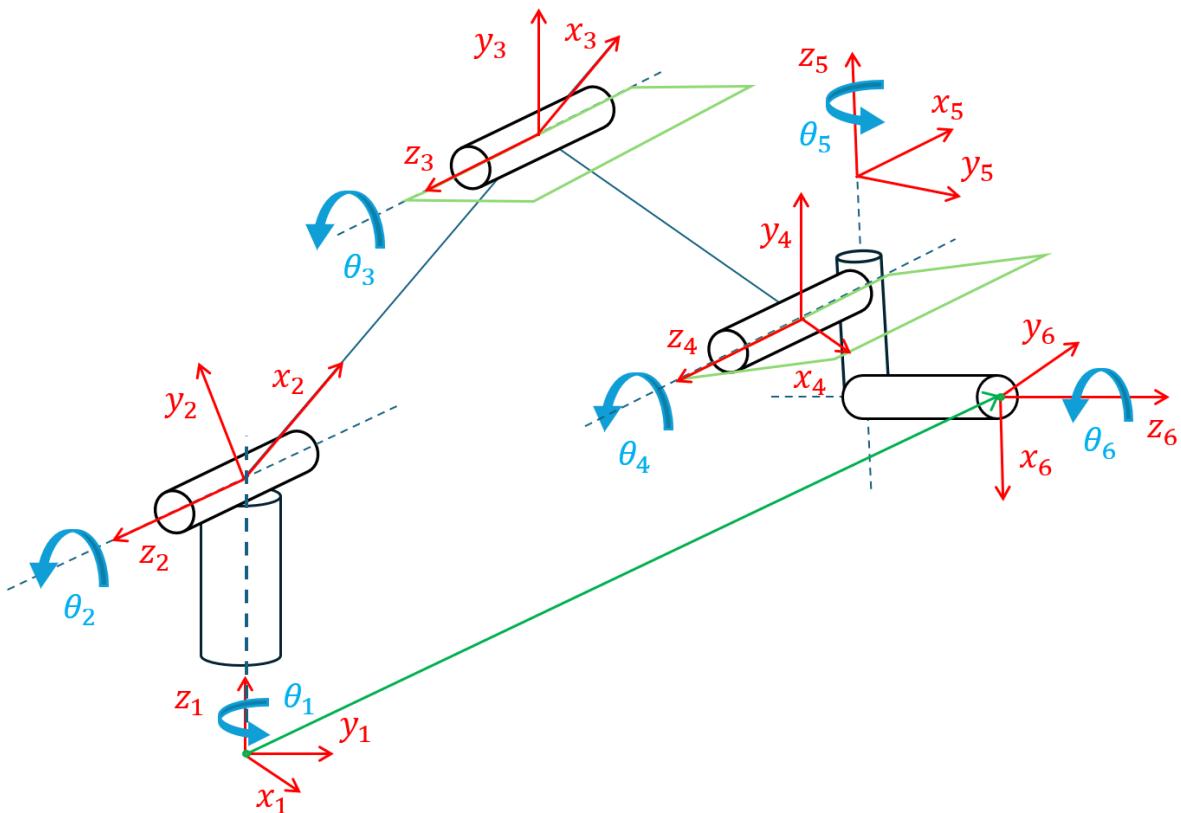
ESEMPIO TARGET POSITION ORANGE FASE 1



Parametri di Denavit-Hartemberg				
LINK	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0.089	θ_1
2	0.425	0	0	θ_2
3	0.392	0	0	θ_3
4	0	$\pi/2$	0.109	θ_4
5	0	$-\pi/2$	0.094	θ_5
6	0	0	0.082	θ_6

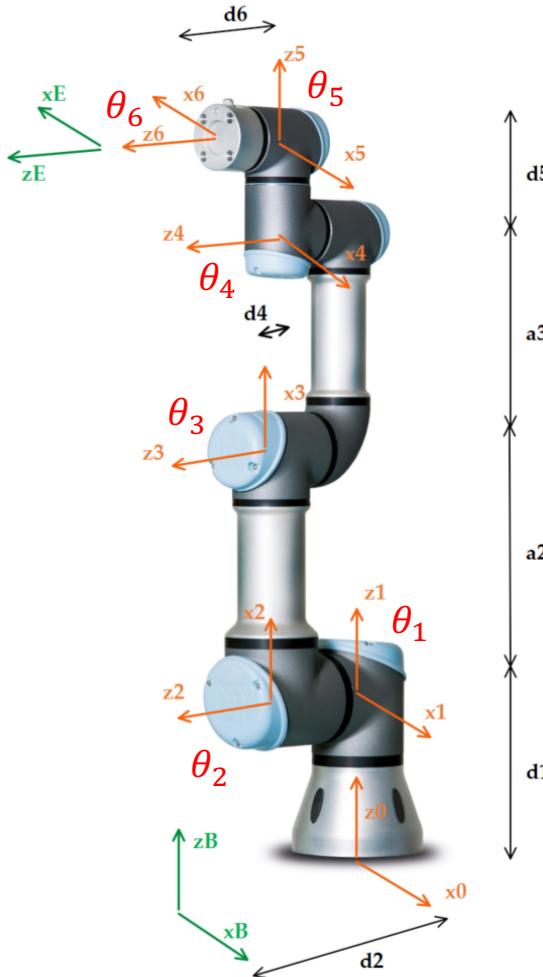
target_pos_orange = [θ_1 θ_2 θ_3 θ_4 θ_5 θ_6]
 target_pos_orange = [-1.570796 -1.87972 -2.139774 -2.363176 -1.50971 0]

CONVENZIONE DI DENAVIT-HARTEMBERG : CASO 3D



target_pos_orange=[θ_1 θ_2 θ_3 θ_4 θ_5 θ_6]
target_pos_orange =[-1.570796 -1.87972 -2.139774 -2.363176 -1.50971 0]

CONVENZIONE DI DENAVIT-HARTEMBERG : CASO 3D



CODICE MATLAB

```

%% Definizione delle a_i : distanza di O_i da O_i^'
a1=0;
a2=0.425;
a3=0.392;
a4=0;
a5=0;
a6=0;

%% Definizione delle d_i : coordinata su z_{(i-1)} di O_i^'
d1=0.089;
d2=0;
d3=0;
d4=0.109;
d5=0.094;
d6=0.082;

%% Definizione delle α_i : coordinata su z_{(i-1)} di O_i^'
alfa1=pi/2;
alfa2=0;
alfa3=0;
alfa4=pi/2;
alfa5=-pi/2;
alfa6=0;

%% Definizione θ_i : angolo intorno all'asse z_{(i-1)} tra l asse x_{(i-1)}
% e l'asse x_i valutato positivo in senso antiorario
teta1=-1.5708;
teta2=-1.8797;
teta3=-2.1398;
teta4=-2.3632;
teta5=-1.5097;
teta6=0;

```

CONVENZIONE DI DENAVIT-HARTEMBERG : CASO 3D

CODICE MATLAB

```
%% Definizione della MATRICE DI TRASFORMAZIONE OMOGENEA
```

```
A_2_3=[cos(teta3) -sin(teta3)*cos(alfa3) sin(teta3)*sin(alfa3) a3*cos(teta3) ;  
       sin(teta3)   cos(teta3)*cos(alfa3) -cos(teta3)*sin(alfa3) a3*sin(teta3) ;  
           0          sin(alfa3)           cos(alfa3)           d3 ;  
           0            0                0           1];
```

```
A_1_2=[cos(teta2) -sin(teta2) 0 a2*cos(teta2) ;  
       sin(teta2)   cos(teta2) 0 a2*sin(teta2) ;  
           0            0      1    0 ;  
           0            0      0    1];
```

```
A_0_1=[cos(teta1) 0 sin(teta1) 0 ;  
       sin(teta1) 0 -cos(teta1) 0 ;  
           0            0      1    0 ;  
           0            0      0    1];
```

```
T_0_3=A_0_1*A_1_2*A_2_3;
```

```
A_3_4=[cos(teta4) 0 -sin(teta4) 0 ;  
       sin(teta4) 0 cos(teta4) 0 ;  
           0        -1    0    0 ;  
           0        0    0    1];
```

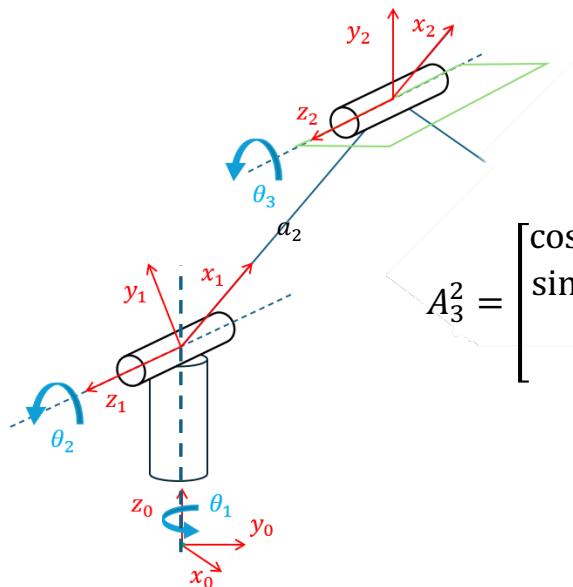
```
A_4_5=[cos(teta5) 0 sin(teta5) 0 ;  
       sin(teta5) 0 cos(teta5) 0 ;  
           0        1    0    0 ;  
           0        0    0    1];
```

```
A_5_6=[cos(teta6) -sin(teta6) 0 0 ;  
       sin(teta6)  cos(teta6) 0 0 ;  
           0            0      1 0 ;  
           0            0      0 1];
```

```
T_3_6=A_3_4*A_4_5*A_5_6;
```

CINEMATICA DIRETTA MANIPOLATORE

LINK	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0.089	1.570796
2	0.425	0	0	-1.87972
3	0.392	0	0	-2.139774



$$A_1^0 = \begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2 \cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & a_2 \sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.3040 & 0.9527 & 0 & -0.1292 \\ -0.9527 & -0.3040 & 0 & -0.4049 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

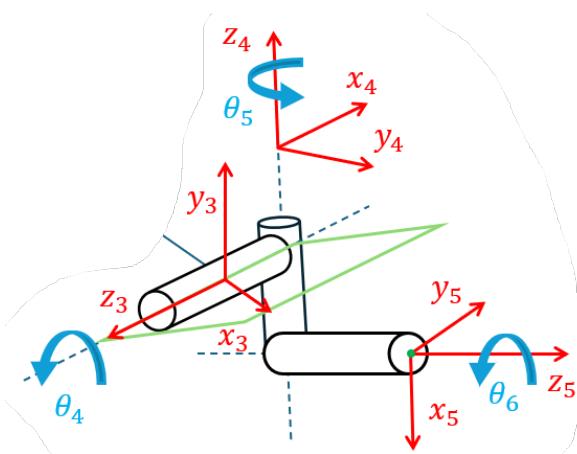
$$A_3^2 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3)\cos(\alpha_3) & \sin(\theta_3)\sin(\alpha_3) & a_3 \cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3)\cos(\alpha_3) & -\cos(\theta_3)\sin(\alpha_3) & a_3 \sin(\theta_3) \\ 0 & \sin(\alpha_3) & \cos(\alpha_3) & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.5388 & 0.8425 & 0 & -0.2112 \\ -0.8425 & -0.5388 & 0 & -0.3302 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^0 = A_1^0 A_2^1 A_3^2$$

$$T_3^0 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0.6388 & 0.7694 & 0 & 0.3796 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

CINEMATICA DIRETTA POLSO SFERICO

LINK	a_i	α_i	d_i	θ_i
4	0	$\pi/2$	0.109	2.363176
5	0	$-\pi/2$	0.094	-1.50971
6	0	0	0.082	0



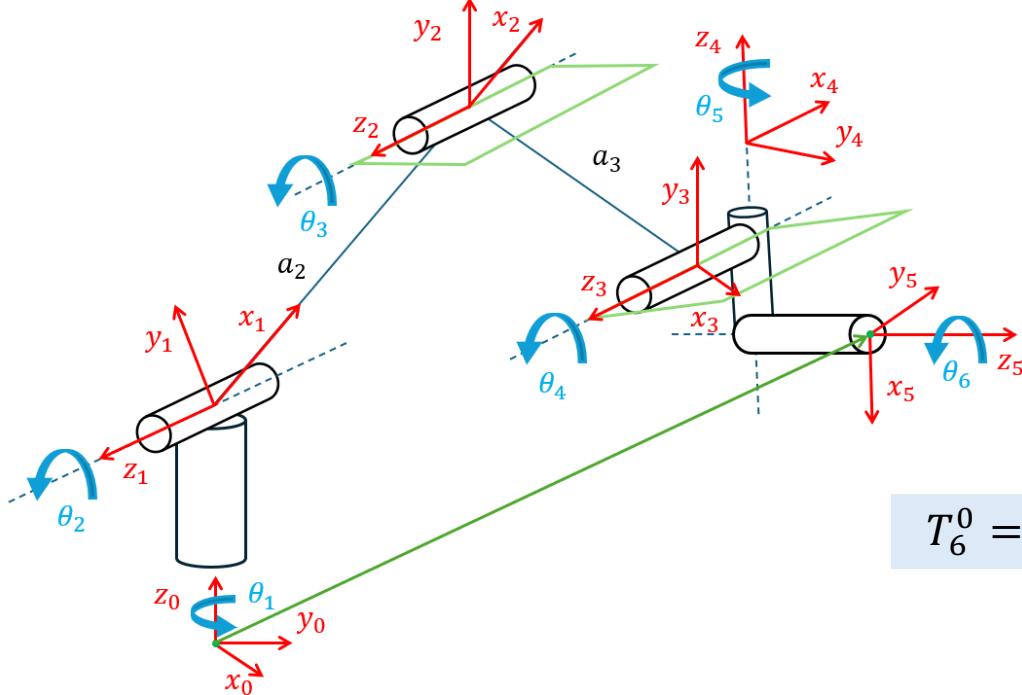
$$A_4^3 = \begin{bmatrix} \cos(\theta_4) & 0 & -\sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & \cos(\theta_4) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.7120 & 0 & 0.7021 & 0 \\ -0.7021 & 0 & -0.7120 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5^4 = \begin{bmatrix} \cos(\theta_5) & 0 & \sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & -\cos(\theta_5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.0611 & 0 & -0.9981 & 0 \\ -0.9981 & 0 & -0.0611 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6^5 = \begin{bmatrix} \cos(\theta_6) & -\sin(\theta_6) & 0 & 0 \\ \sin(\theta_6) & \cos(\theta_6) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^3 = A_4^3 A_5^4 A_6^5 = \begin{bmatrix} -0.0435 & 0.7021 & 0.7107 & 0 \\ -0.0429 & -0.7120 & 0.7008 & 0 \\ 0.9981 & 0 & 0.0611 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

CINEMATICA DIRETTA MANIPOLATORE ANTROPOMORFO



$$T_6^0 = T_3^0 T_6^3$$

$$T_3^0 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0.6388 & 0.7694 & 0 & 0.3796 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

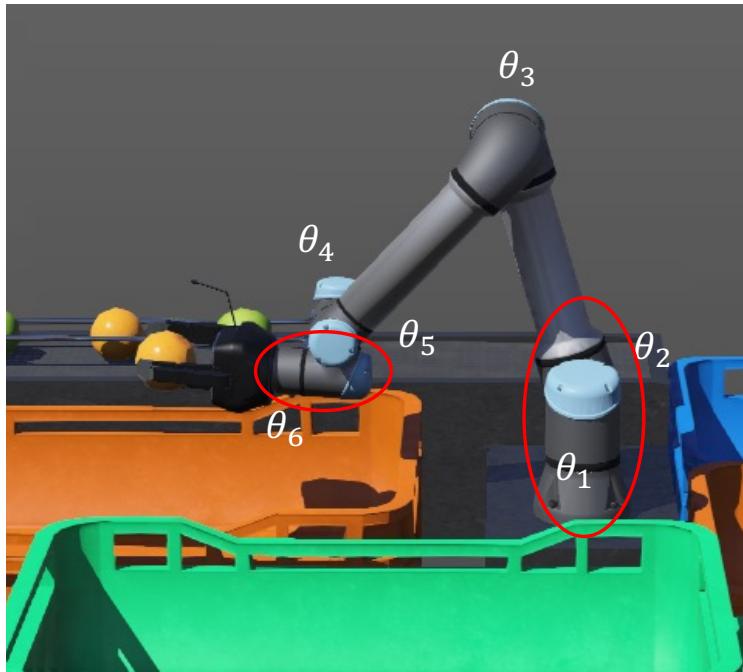
$$T_6^3 = \begin{bmatrix} -0.0435 & 0.7021 & 0.7107 & 0 \\ -0.0429 & -0.7120 & 0.7008 & 0 \\ 0.9981 & 0 & 0.0611 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^0 = \begin{bmatrix} -0.9981 & 0 & -0.0611 & 0 \\ -0.0608 & -0.0994 & 0.9932 & 0.3796 \\ 0.9981 & 0 & 0.0611 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$p_6 = T_6^0 p_0$$

CINEMATICA DIRETTA – UR5e PLANARE

Scegliamo una configurazione specifica del manipolatore: target_position_orange



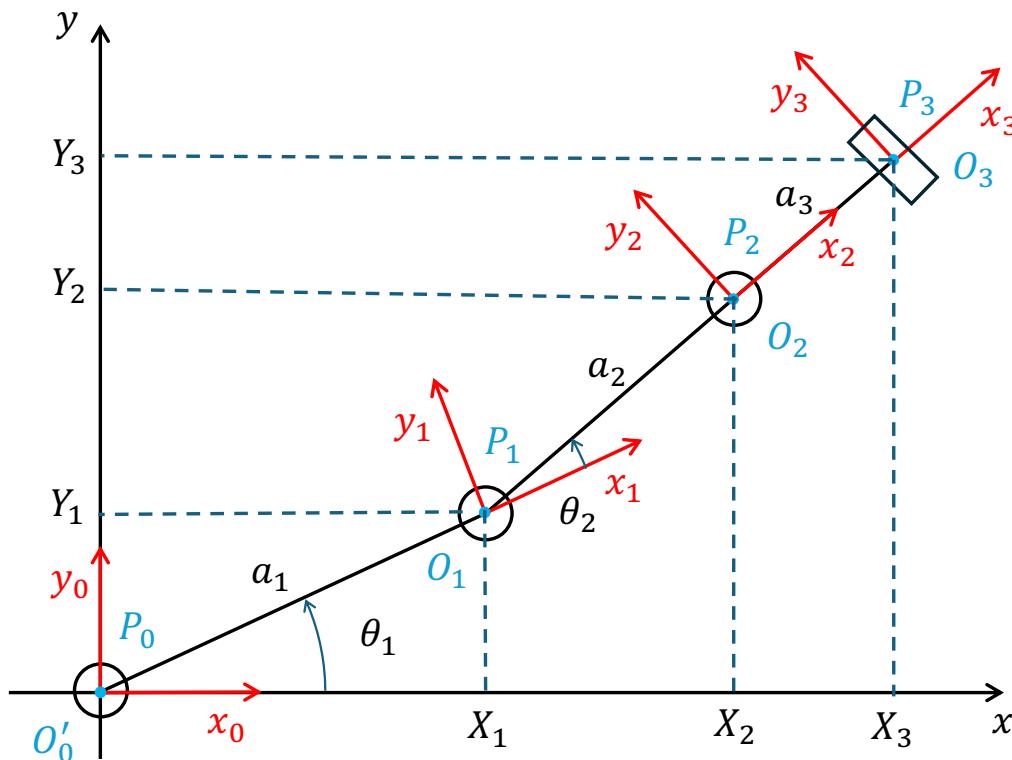
IPOTESI:

- Giunti shoulder_lift_joint e shoulder_pan_joint = unico giunto
- Giunti wrist_2_joint e wrist_3_joint = unico giunto
- $\theta_4 = 0$

$$\begin{aligned} \text{target_pos_orange} &= [\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6] \\ \text{target_pos_orange} &= [-1.570796 \quad -1.87972 \quad -2.139774 \quad -2.363176 \quad -1.50971 \quad 0] \end{aligned}$$

CINEMATICA DIRETTA – UR5e PLANARE

Obiettivo: Identificare la posizione del punto P_3 con sistema di riferimento O_3 in funzione del punto P_0 con sistema di riferimento O'_0



Parametri di Denavit-Hartenberg

LINK	a_i	α_i	d_i	θ_i
1	0,425+0,1625	0	0	-1.87972
2	0.3922	0	0	-2.139774
3	0.0997	0	0	0

$$i = 1, 2, 3$$

Identificazione della T_3^0

CINEMATICA DIRETTA – UR5e PLANARE

Dati θ_1, θ_2 e θ_3 ricaviamo la posizione finale del punto P_3 , attraverso l'utilizzo delle matrici:

$$\begin{cases} \theta_1 = -1.87972 \\ \theta_2 = -2.139774 \\ \theta_3 = 0 \end{cases}$$

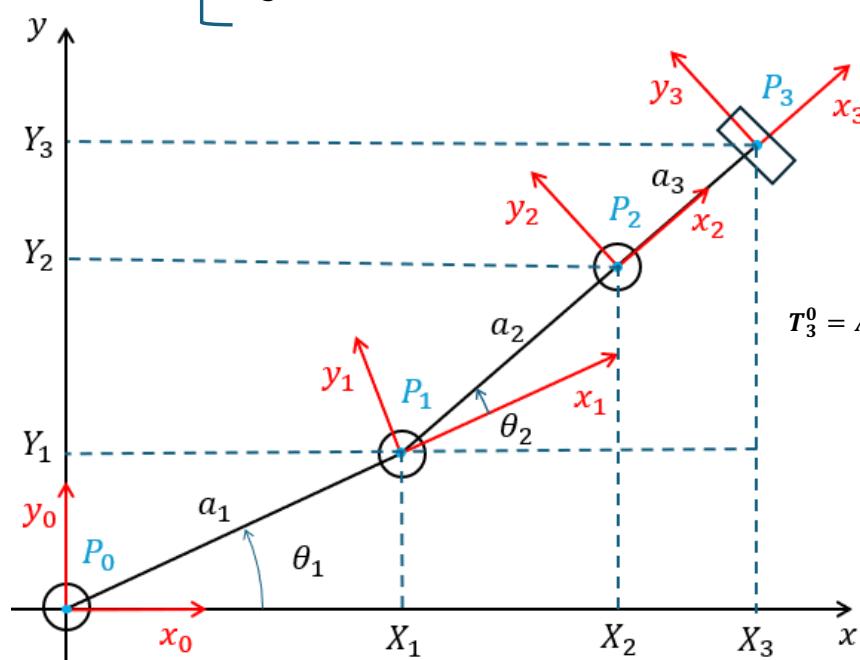
$$A_1^0 = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & a_1 \cos(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) & 0 & a_1 \sin(\theta_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1 = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 & a_2 \cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & 0 & a_2 \sin(\theta_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 & a_3 \cos(\theta_3) \\ \sin(\theta_3) & \cos(\theta_3) & 0 & a_3 \sin(\theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^0 = A_1^0 A_2^1 A_3^2 = \begin{bmatrix} \cos(\theta_1 + \theta_2 + \theta_3) & -\sin(\theta_1 + \theta_2 + \theta_3) & 0 & a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) + a_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ \sin(\theta_1 + \theta_2 + \theta_3) & \cos(\theta_1 + \theta_2 + \theta_3) & 0 & a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) + a_3 \sin(\theta_1 + \theta_2 + \theta_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

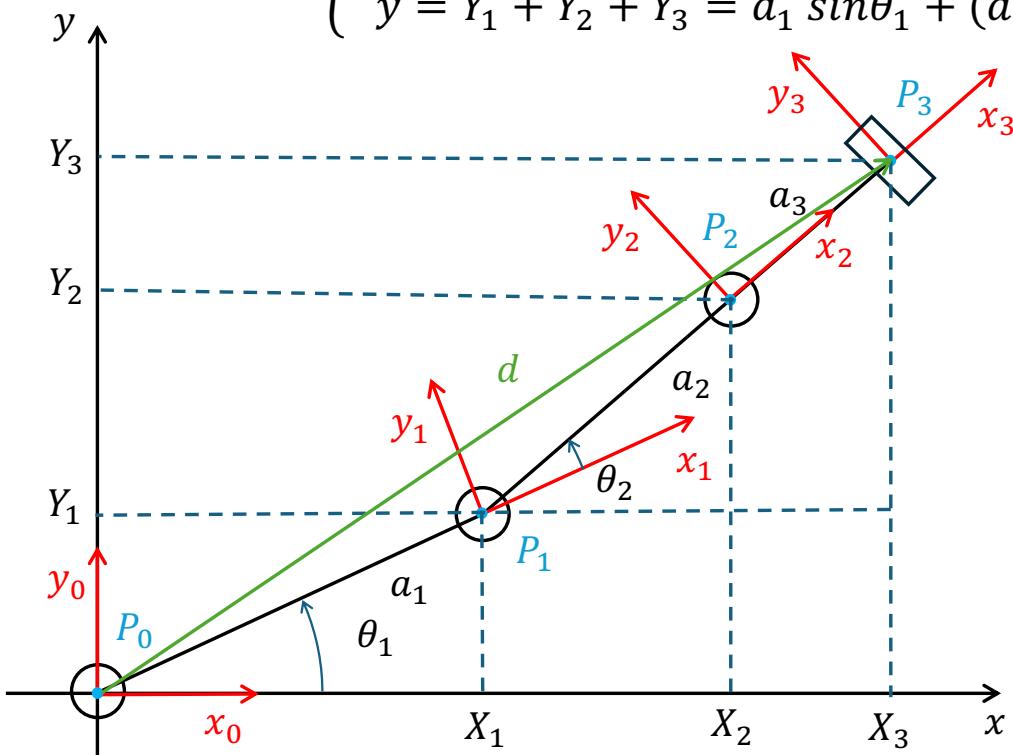
$$T_3^0 = \begin{bmatrix} -0.6388 & -0.7694 & 0 & -0.4928 \\ 0.7694 & -0.6388 & 0 & -0.1812 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



CINEMATICA INVERSA – UR5e PLANARE

Obiettivo: Identificare gli angoli θ_1 , θ_2 e θ_3 nota la posizione dell'end-effector nello spazio

$$\begin{cases} x = X_1 + X_2 + X_3 = a_1 \cos\theta_1 + (a_2 + a_3) \cos(\theta_1 + \theta_2) = -0,4928 \text{ mm} \\ y = Y_1 + Y_2 + Y_3 = a_1 \sin\theta_1 + (a_2 + a_3) \sin(\theta_1 + \theta_2) = -0,1812 \text{ mm} \end{cases}$$



Data **d** la distanza tra la base e l'end-effector:

$$d = \sqrt{x^2 + y^2}$$

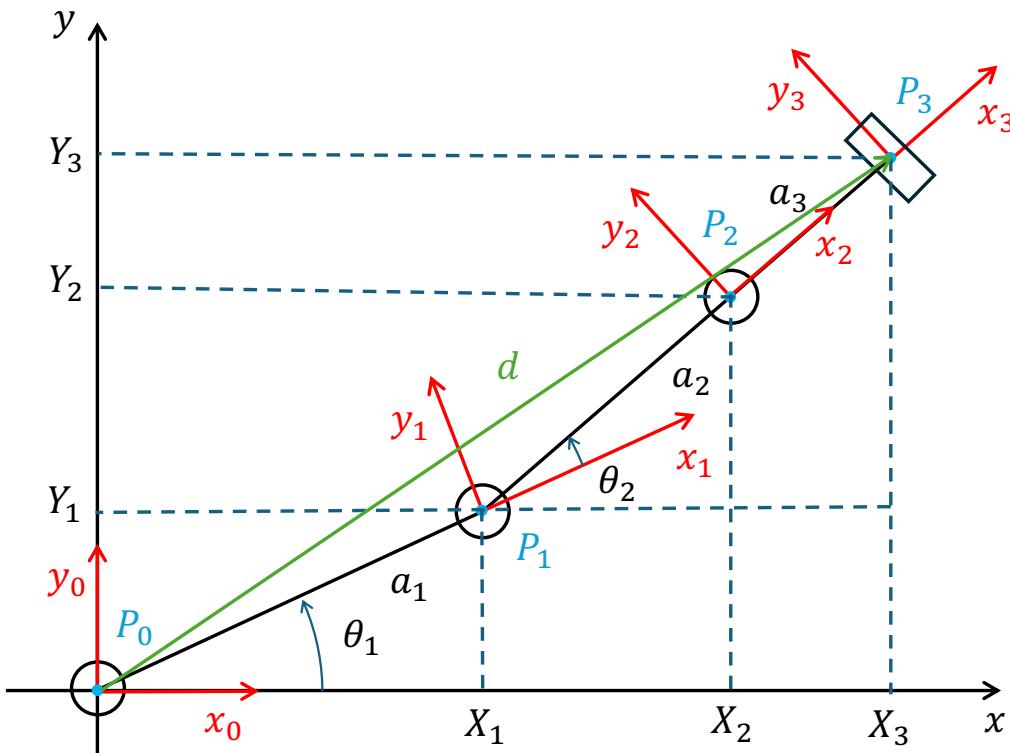
Determiniamo θ_1 utilizzando la trigonometria inversa:

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) \pm \cos^{-1}\left(\frac{a_2^2 + d^2 - a_3^2}{2a_1d}\right)$$

Stesso calcolo viene eseguito per trovare θ_2 e θ_3

CINEMATICA INVERSA – UR5e PLANARE

Obiettivo: Identificare gli angoli θ_1 , θ_2 e θ_3 nota la posizione dell'end-effector nello spazio



In generale la soluzione si trova senza una procedura sistematica, piuttosto basandosi sull'intuizione nel manipolare le equazioni



Utilizzo di tecniche numeriche per individuare la soluzione (MatLab)

1. La soluzione analitica (in forma chiusa) potrebbe non esistere.
2. Potrebbero esistere soluzioni multiple o un numero infinito di soluzioni

CODICE MATLAB CINEMATICA INVERSA

```
% Definire i parametri noti
a1 = 0.5875;
a2 = 0.3922;
a3 = 0.0997 ; % esempio, sostituire con il valore reale
x = -0.4928;
y = -0.1812;

% Valori iniziali per theta1, theta2 e theta3
initial_guess = [0.0, 0.0, 0.0];

% Creare una funzione anonima che passa i parametri aggiuntivi
equations_with_params = @(vars) equations(vars, a1, a2, a3, x, y);

% Risolviamo il sistema di equazioni usando fsolve, mostriamo le
% iterazioni durante il processo di risoluzione
options = optimoptions('fsolve', 'Display', 'iter'); % Mostra iterazioni
solution = fsolve(equations_with_params, initial_guess, options);
```

Soluzione attesa

$$\begin{aligned}\theta_1 &= -1.87972 \text{ rad} \\ \theta_2 &= -2.139774 \text{ rad} \\ \theta_3 &= 0 \text{ rad}\end{aligned}$$

```
% Estrarre i risultati
theta1_solution = solution(1);
theta2_solution = solution(2);
theta3_solution = solution(3);

% Stampare i risultati
fprintf('theta1: %.6f radians\n', theta1_solution);
fprintf('theta2: %.6f radians\n', theta2_solution);
fprintf('theta3: %.6f radians\n', theta3_solution);

% Definire il sistema di equazioni, equation viene definita per inserire i
% parametri a1, a2, a3, x e y.
function F = equations(vars, a1, a2, a3, x, y)
    theta1 = vars(1);
    theta2 = vars(2);
    theta3 = vars(3);

    F(1) = a1 * cos(theta1) + a2 * cos(theta1 + theta2) + a3 * cos(theta1 + theta2 + theta3) - x;
    F(2) = a1 * sin(theta1) + a2 * sin(theta1 + theta2) + a3 * sin(theta1 + theta2 + theta3) - y;
    F(3) = theta1 + theta2 + theta3 - 2*pi; % Vincolo per la somma degli angoli
end
```

Soluzione ottenuta tramite metodi numerici

$$\begin{aligned}\theta_1 &= 2.950365 \text{ rad} \\ \theta_2 &= 1.723623 \text{ rad} \\ \theta_3 &= 1.609197 \text{ rad}\end{aligned}$$

DEFINIZIONE DELLE FASI

NUMERO FASI : 4



FASE
1

4 MELE
CONTENITORE
A

4 ARANCE
CONTENITORE
B

FASE
2

4 MELE
CONTENITORE
C

4 ARANCE
CONTENITORE
D

FASE
3

4 MELE
CONTENITORE
B

4 ARANCE
CONTENITORE
A

FASE
4

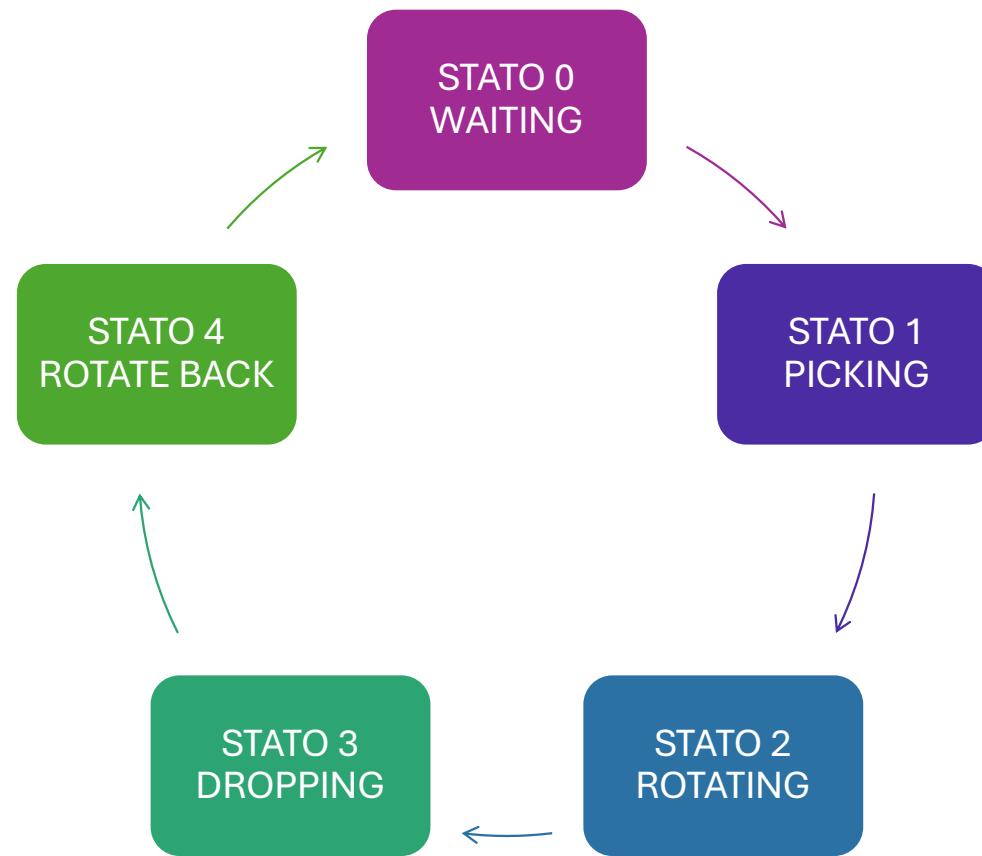
4 MELE
CONTENITORE
D

2 ARANCE
CONTENITORE
C

N.B. SE IL ROBOT INCONTRA ARANCE MARCE
POSIZIONAMENTO IN CONTENITORE E

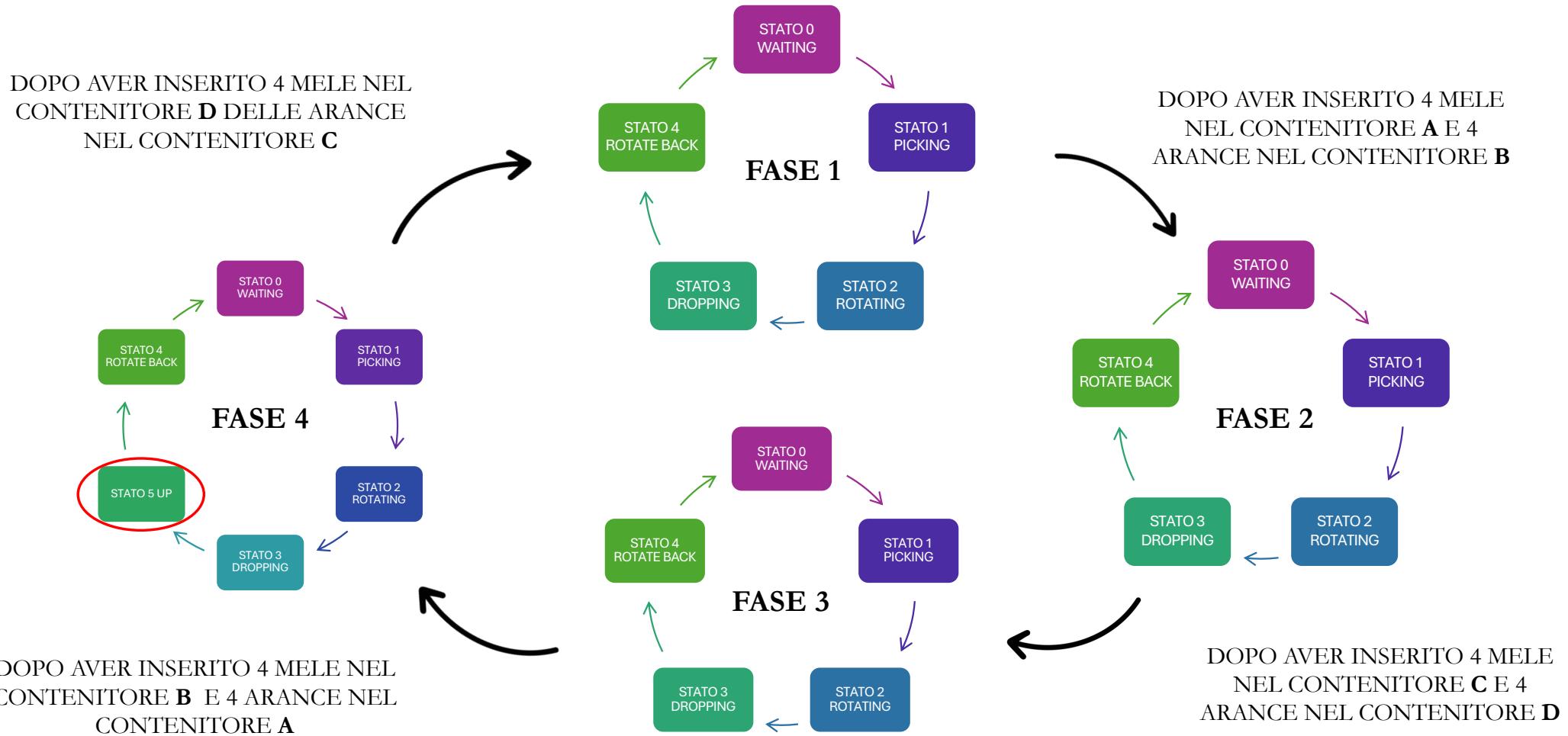
SI PASSA ALLA FASE SUCCESSIVA OGNI QUAL VOLTA
LA FASE PRECEDENTE VIENE COMPLETATA

AUTOMA A STATI FINITI



AUTOMA A STATI FINITI

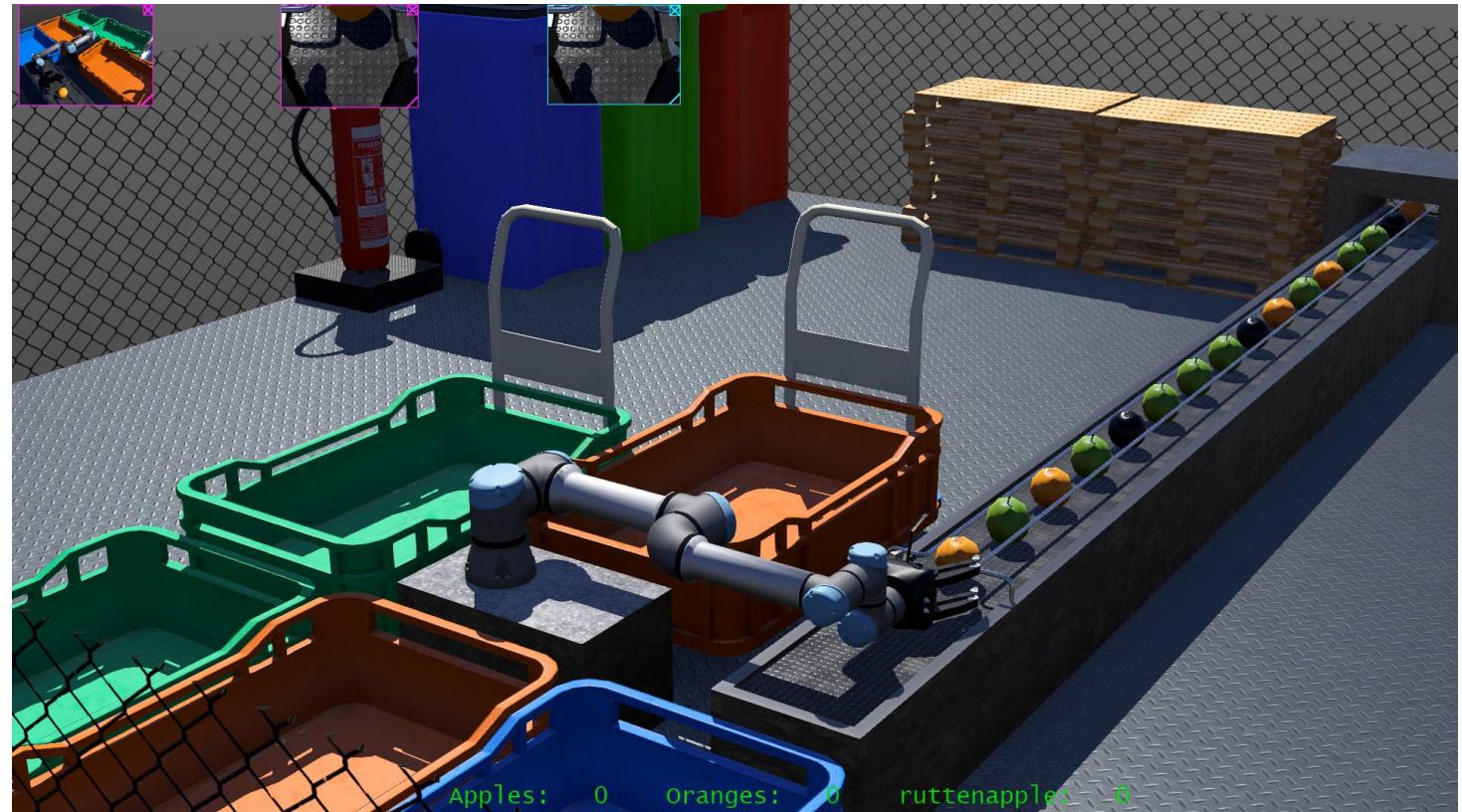
DOPO AVER INSERITO 4 MELE NEL
CONTENITORE **D** DELLE ARANCE
NEL CONTENITORE **C**



AUTOMA A STATI FINITI

STATI DEL FUNZIONAMENTO DEL ROBOT

STATO 0 WAITING



STATO 1 PICKING

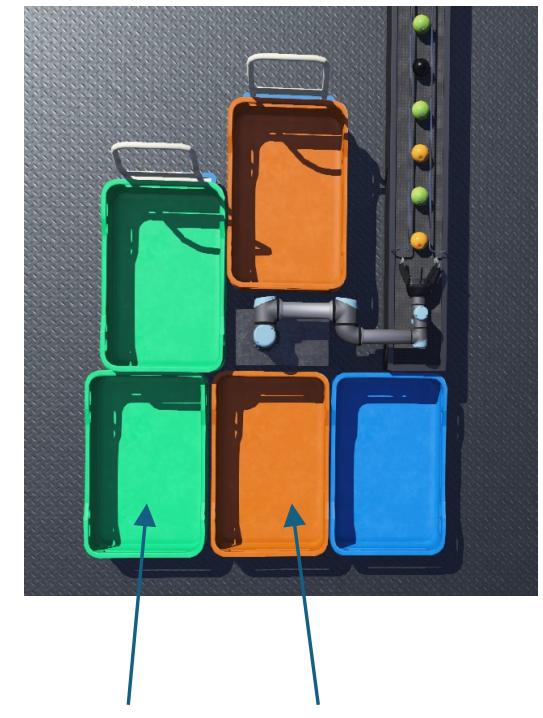
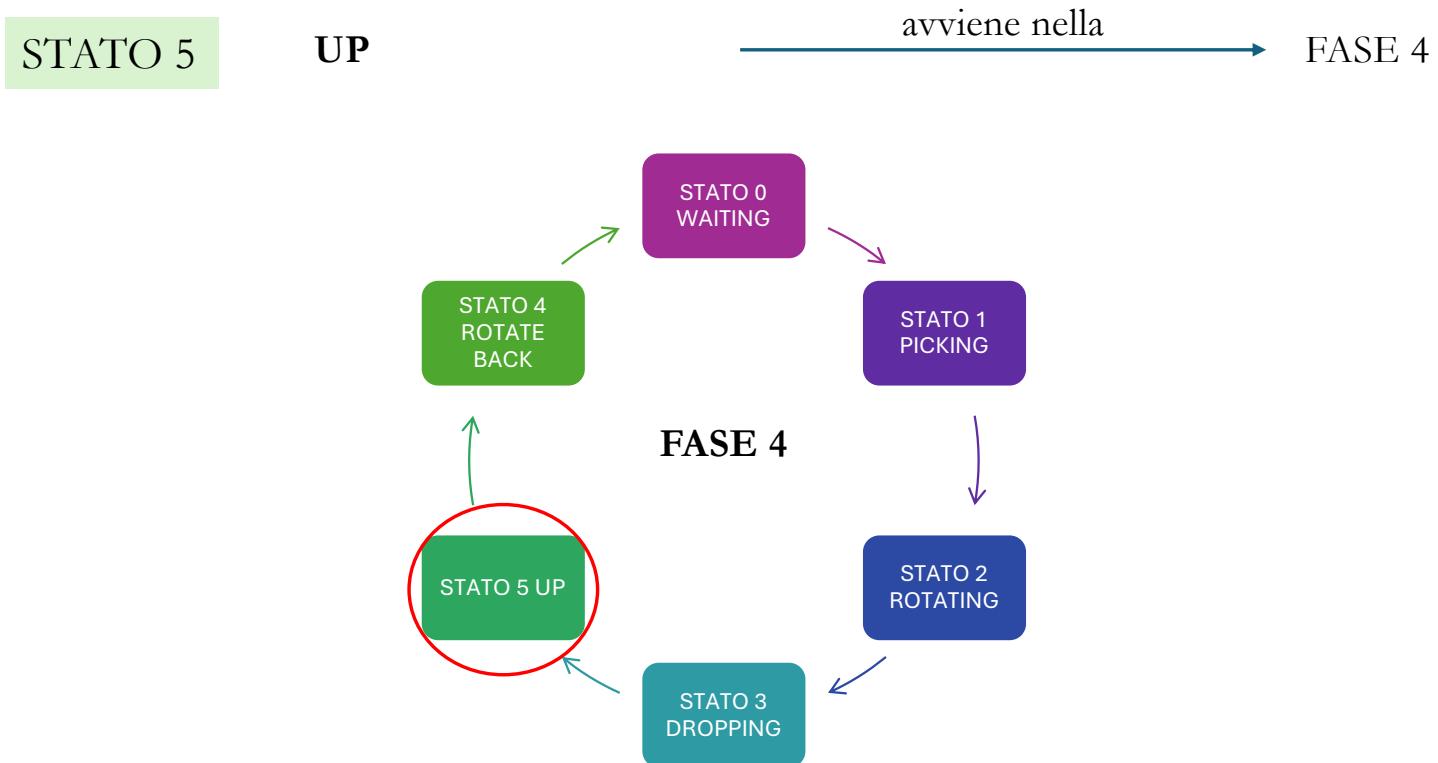
STATO 2 ROTATING

STATO 3 DROPPING

STATO 4 ROTATE BACK

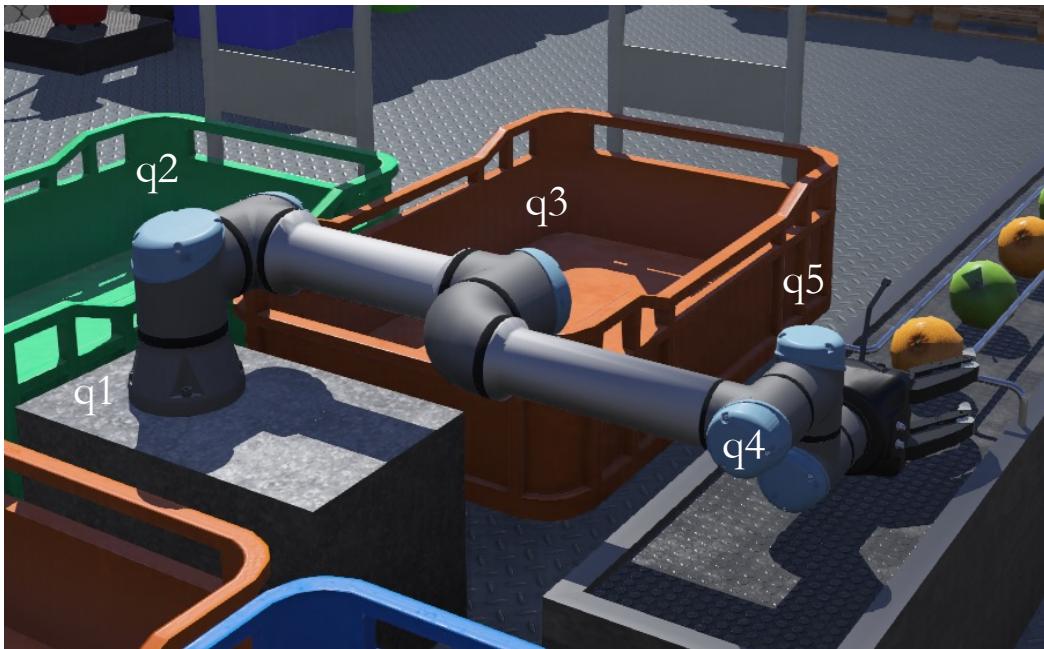
AUTOMA A STATI FINITI

STATI DEL FUNZIONAMENTO DEL ROBOT

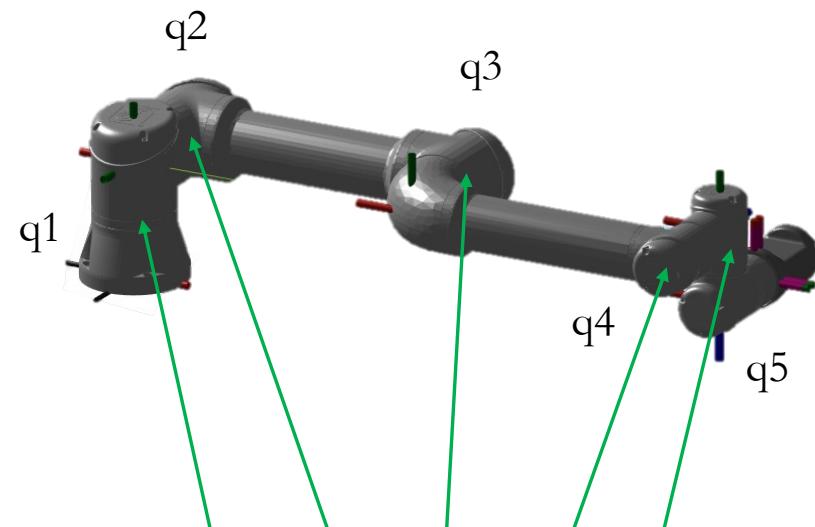


HOME POSITION

universalUR5e



UR5e



$$\text{homePosition} = ([180 \quad 0 \quad 0 \quad 0 \quad 90])$$

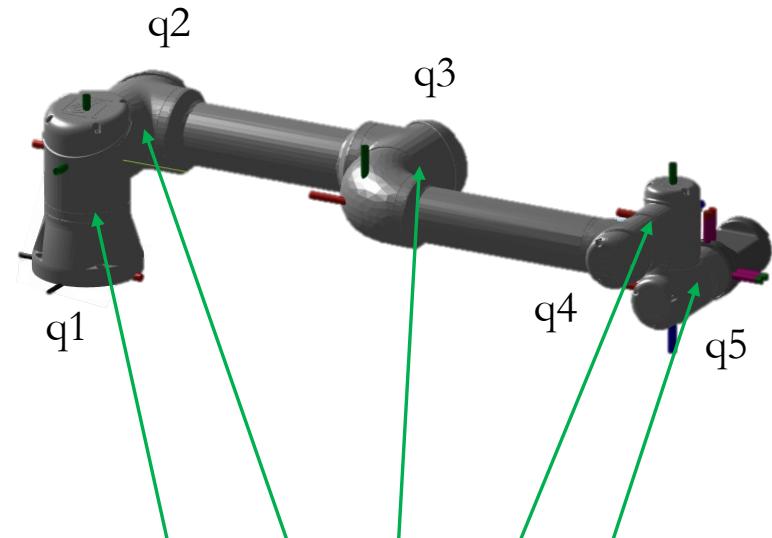
HOME POSITION

CODICE MATLAB

```
>> homePosition = deg2rad([180 0 0 0 0 90]);
% Set home position of each joint
ur5e.Bodies{1, 3}.Joint.HomePosition = homePosition(1);
ur5e.Bodies{1, 4}.Joint.HomePosition = homePosition(2);
ur5e.Bodies{1, 5}.Joint.HomePosition = homePosition(3);
ur5e.Bodies{1, 6}.Joint.HomePosition = homePosition(4);
ur5e.Bodies{1, 7}.Joint.HomePosition = homePosition(5);
ur5e.Bodies{1, 8}.Joint.HomePosition = homePosition(6);
```

```
% Show robot at home position
f1 = figure;
show(ur5e,homePosition,'Frames','on','PreservePlot',false,'Collisions','off','Visuals','on');
hold on
```

UR5e



homePosition = deg2rad ([180 0 0 0 0])

CODICE PYTHON

```
"""fruit_sorting_ctrl_opencv controller."""
import cv2
import numpy as np
from controller import Supervisor
```

Importa le librerie necessarie:

- cv2 per OpenCV
- numpy per la manipolazione delle matrici

```
def wait(milliseconds):
    start_time = robot.getTime() * 1000 # Get the current simulation
    while (robot.getTime() * 1000 - start_time) < milliseconds:
        robot.step(timestep) # Perform a simulation step

robot = Supervisor()
```

Definizione funzione **wait()**

Prende in ingresso i millisecondi desiderati di attesa e ferma il robot in tale lasso di tempo

CODICE PYTHON

```
# Set the time step in 64  
timestep = 64
```

Imposta il timestep (intervallo di tempo tra due istanti temporali consecutivi all'interno della simulazione) a 64 millisecondi.

```
# Type of fruit: 0 = orange, 1 = apple, 2 = Rutten apple  
fruit = -1
```

Inizializza la variabile fruit per tenere traccia del tipo di frutto.

```
# Fruits counters  
orange = 0  
apple = 0  
rottenapple = 0
```

Inizializza i contatori per i diversi tipi di frutta

```
# Delay counter  
counter = 0
```

Inizializza il contatore di ritardo

```
# Status of UR5e Robot  
state = 0
```

Inizializza lo stato iniziale robot (stato 0 → waiting)

TARGET POSITION

Le target position del robot corrispondono agli angoli di rotazione che i 6 giunti devono compiere per l'esecuzione del processo

Obiettivo del processo: posizionare mele e arance nei rispettivi contenitori

```
# Target positions to drop fruits
target_pos = [[] for _ in range(3)]
target_positions = list()
target_pos_orange=[-1.570796, -1.87972, -2.139774, -2.363176, -1.50971]
target_pos_apple=[0, -1.87972, -2.139774, -2.363176, -1.50971]
target_pos_rottenapple=[-1, -1.67972, +1.539774, -2.163176, -1.50971]
```

→ Target positions iniziali (FASE 1)


```
target_pos_orange_ts=[1.570796, -1.87972, -2.139774, -2.363176, -1.50971]
target_pos_apple_ts=[0.9, -1.87972, -2.139774, -2.363176, -1.50971]
```

→ Target positions al raggiungimento del **threshold** (FASE 2)


```
target_pos[0] = target_pos_orange
target_pos[1] = target_pos_apple
target_pos[2] = target_pos_rottenapple
```



```
dropValue=[[[] for _ in range(3)]]
dropValue[0]=-2.3
dropValue[1]=-2.3
dropValue[2]=-2
ts=4 #threshold for the apples and oranges
```

→ Valori di angolo a cui si passa alla fase di DROP, definiti per ogni tipo di frutta → Definisce i valori di rilascio e una soglia

TARGET POSITION FASE 1

```
# Target positions to drop fruits
target_pos = [[] for _ in range(3)]
target_positions = list()
target_pos_orange=[-1.570796, -1.87972, -2.139774, -2.363176, -1.50971]
target_pos_apple=[0, -1.87972, -2.139774, -2.363176, -1.50971]
target_pos_rottenapple=[-1, -1.67972, +1.539774, -2.163176, -1.50971]
```

Conversione da radianti a gradi:

$$\alpha^\circ : 180^\circ = \alpha_{rad} : \pi \quad \longrightarrow \quad \alpha^\circ = \frac{\alpha_{rad} \times 180^\circ}{\pi}$$

```
target_pos_orange=      [-90.0000  -107.7000  -122.6000  -135.4000  -86.5000  0 ]
target_pos_apple =      [    0     -107.7000  -122.6000  -135.4000  -86.5000  0 ]
target_pos_rottenapple=[-57.2958   -96.2409    88.2226  -123.9409  -86.5000  0 ]
```

CODICE MATLAB

Command Window

```
>> startup_rtb
- Robotics Toolbox for MATLAB (release 1.0)
- ARTE contributed code: 3D models for robot manipulators (C:\Users\aless\AppData\Roaming\MathWorks\MATLAB Add-Ons\Toolboxes
>> % Carica il modello del robot UR5
mdl_ur5;

% Definisci la configurazione desiderata (in radianti)
q = [0, pi/4, -pi/4, 0, 0, 0]; % Posizione arbitraria

% Visualizza il robot UR5 nella configurazione desiderata con le posizioni dei giunti
figure;
ur5.plot(q);
title('Robot UR5 con Posizioni dei Giunti');
xlabel('X');
ylabel('Y');
zlabel('Z');
ur5.teach(q); % Mostra le posizioni dei giunti
```

ROBOTICS_TOOLBOX

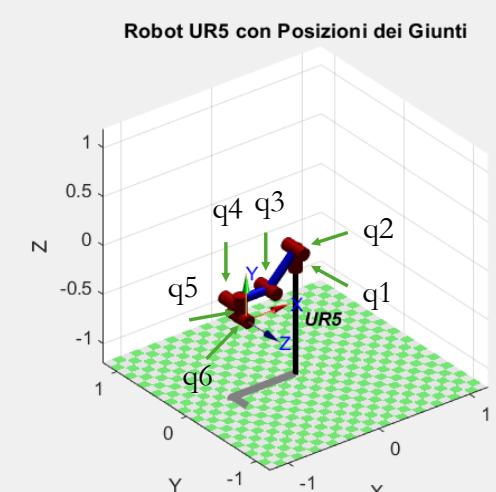
rtb

Coordinate end-effector →

Teach	
X:	-0.693
Y:	-0.191
Z:	-0.306
R:	-0.0
P:	0.0
Y:	90.0

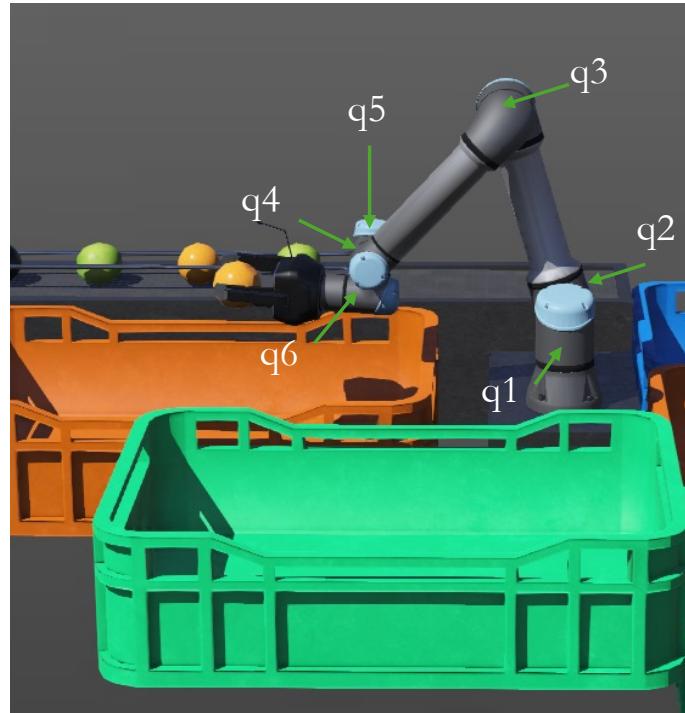
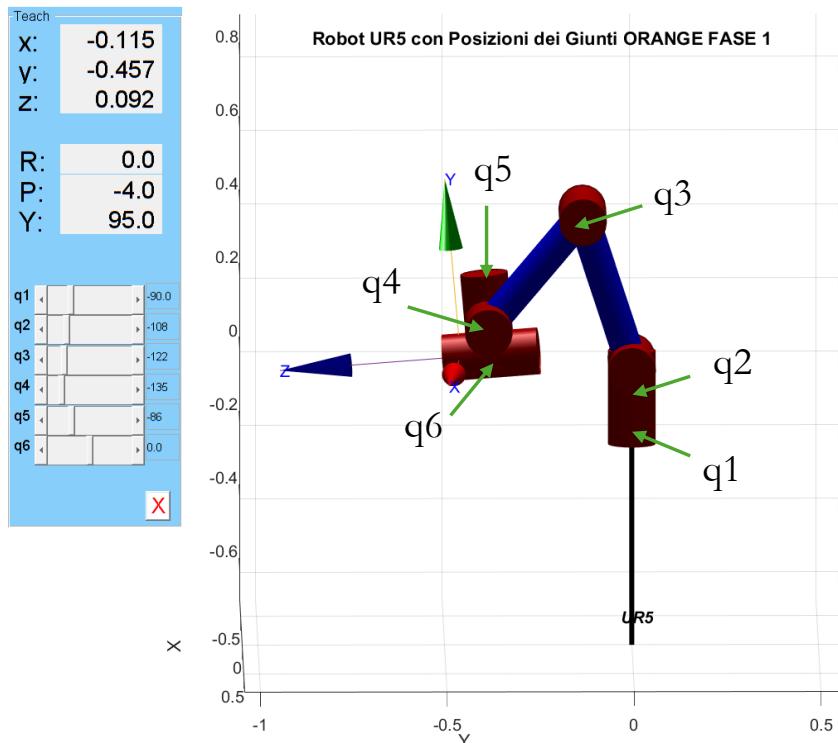
Angoli dei giunti →

q1	0
q2	45
q3	-45
q4	0
q5	0
q6	0



TARGET POSITION FASE 1 - ORANGE

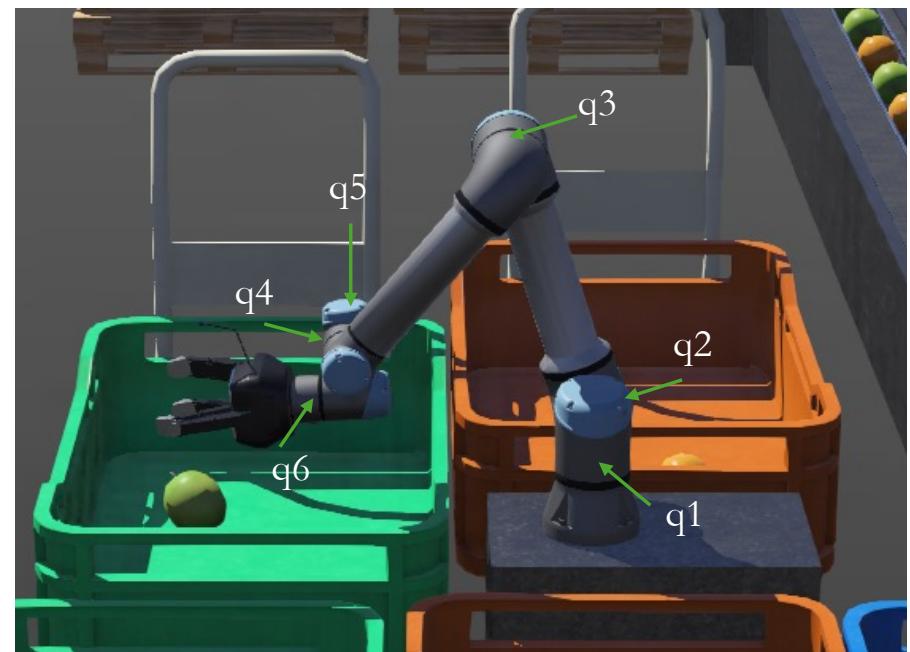
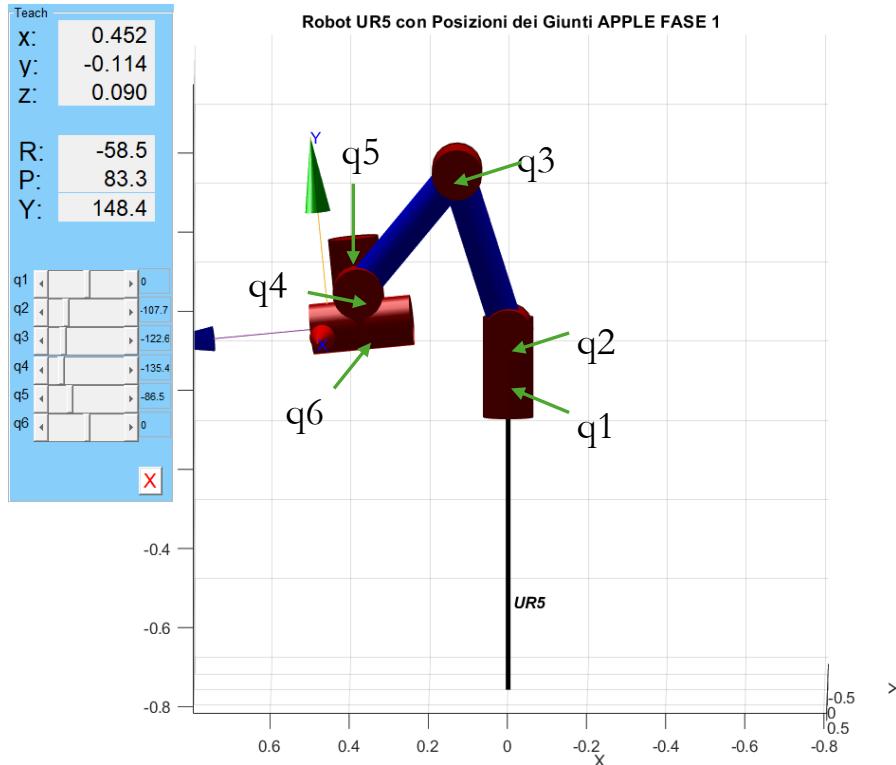
```
target_pos_orange=[-1.570796, -1.87972, -2.139774, -2.363176, -1.50971]
```



```
target_pos_orange=[ q1 q2 q3 q4 q5 q6 ]
target_pos_orange=[ -90.0000 -107.7000 -122.6000 -135.4000 -86.5000 0 ]
```

TARGET POSITION FASE 1 - APPLE

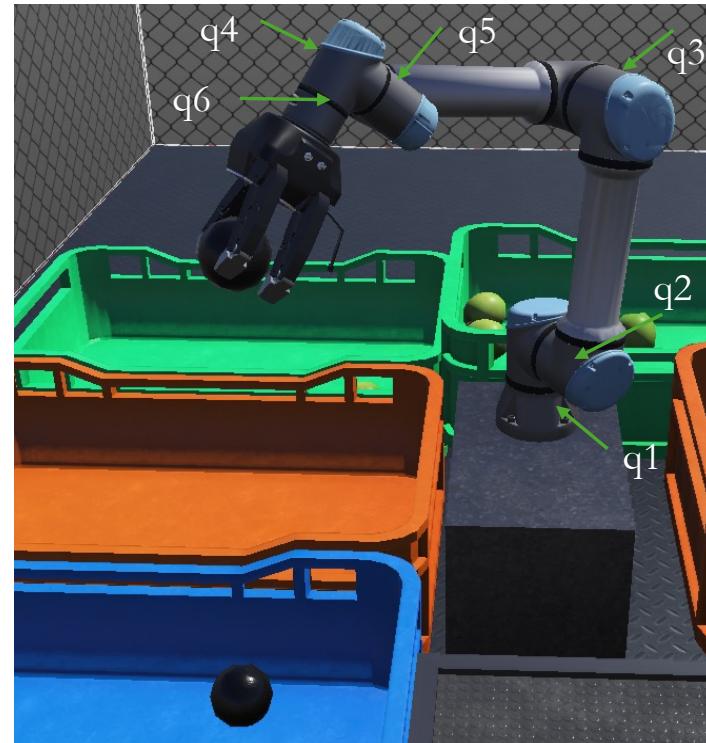
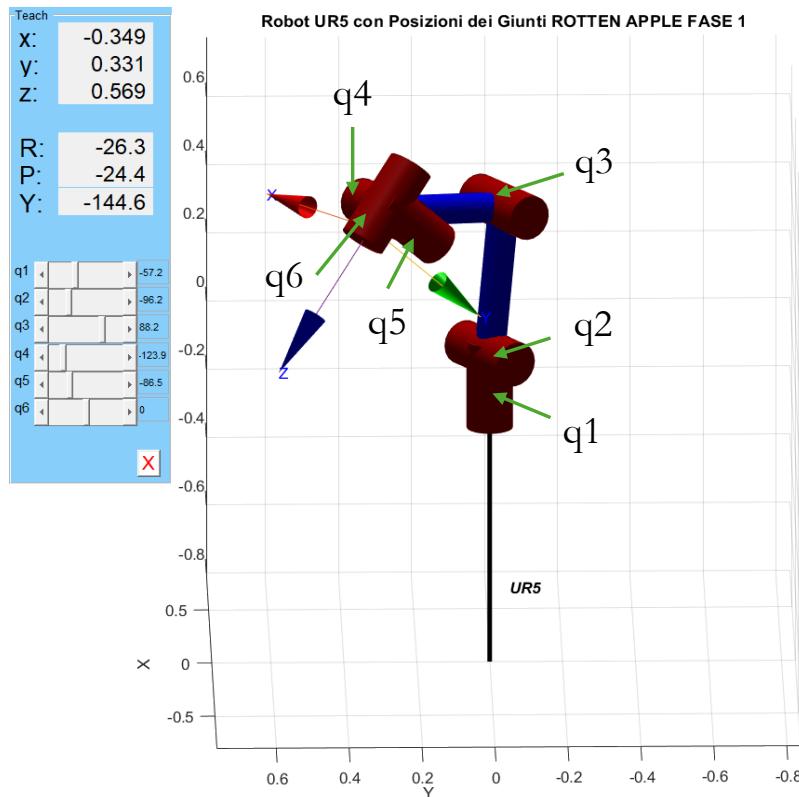
`target_pos_apple=[0, -1.87972, -2.139774, -2.363176, -1.50971]`



`target_pos_apple = [q1 q2
 0 -107.7000 -122.6000 -135.4000 -86.5000 0]`

TARGET POSITION FASE 1 – ROTTEN APPLE

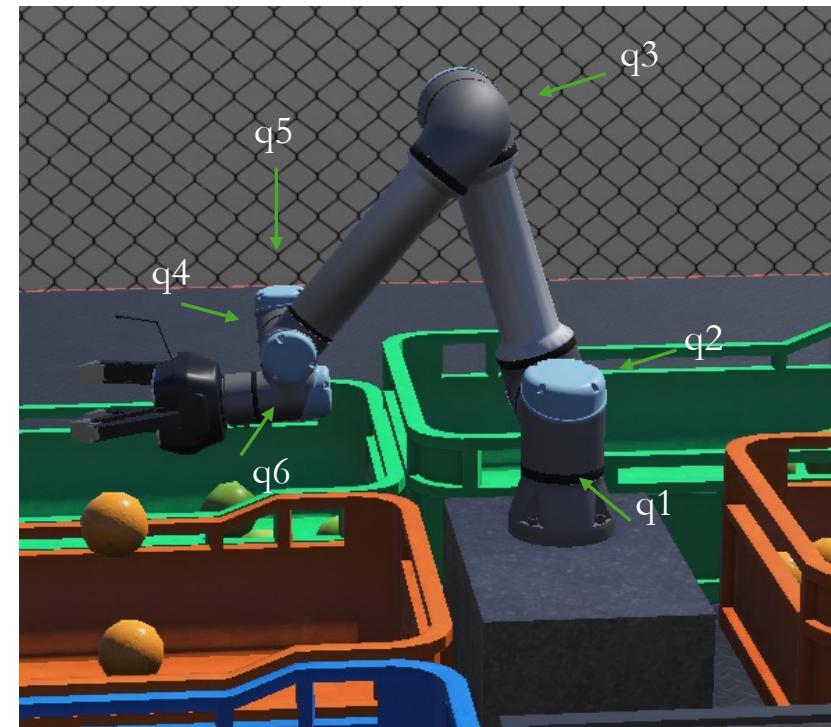
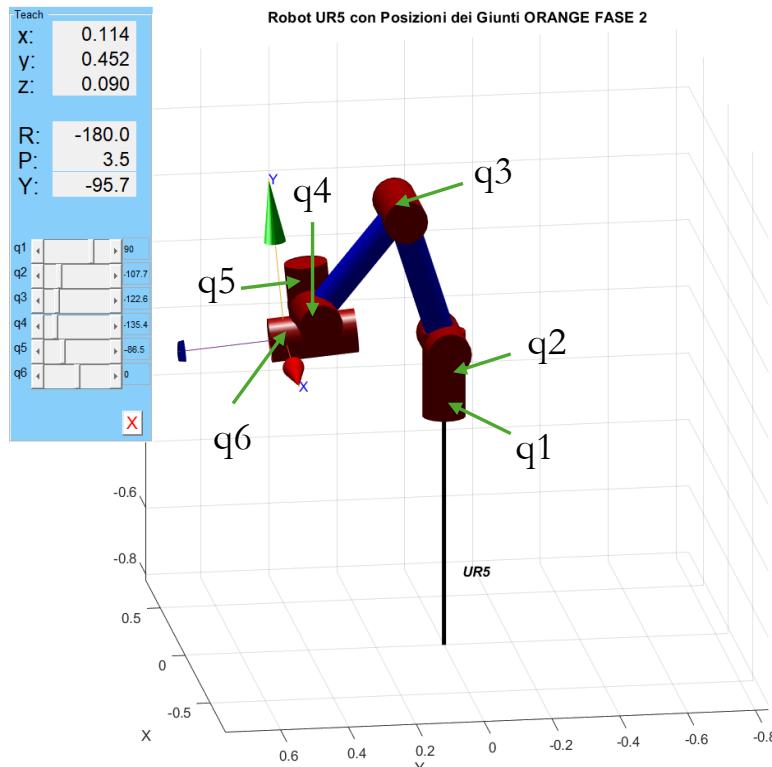
`target_pos_rottenapple=[-1, -1.67972, +1.539774, -2.163176, -1.50971]`



`target_pos_rottenapple = [q1 q2 q3 q4 q5 q6]`
`target_pos_rottenapple= [-57.2958 -96.2409 88.2226 -123.9409 -86.5000 0]`

TARGET POSITION FASE 2 - ORANGE

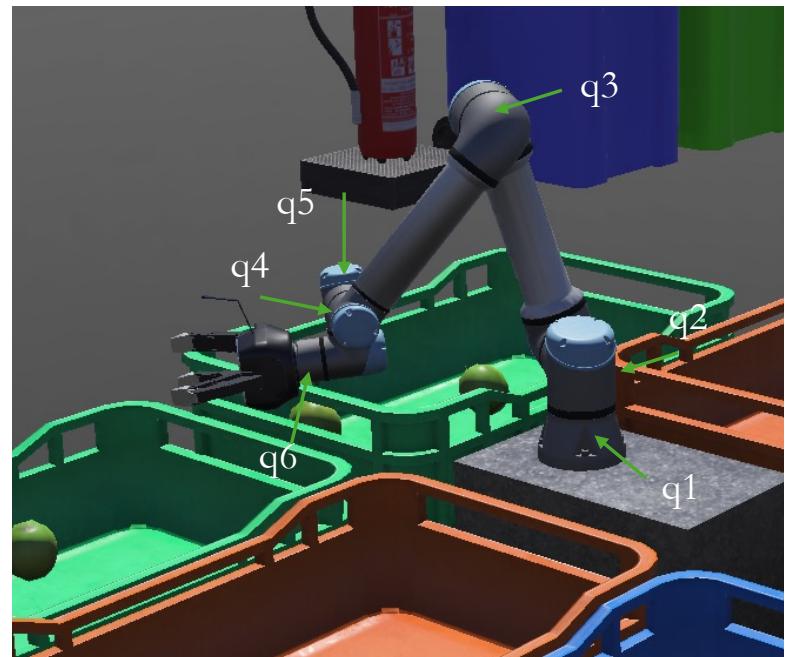
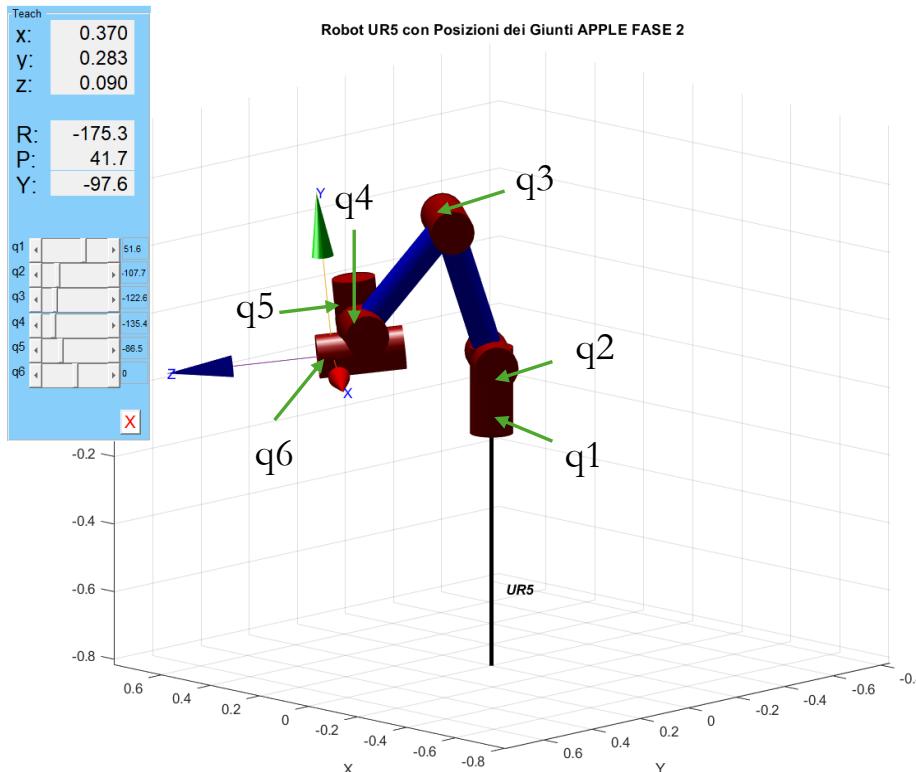
target_pos_orange_ts=[1.570796, -1.87972, -2.139774, -2.363176, -1.50971]



target_pos_orange_ts=[q1 q2 q3 q4 q5 q6]
target_pos_orange_ts=[90.0000 -107.7000 -122.6000 -135.4000 -86.5000 0]

TARGET POSITION FASE 2 - APPLE

target_pos_apple_ts=[0.9, -1.87972, -2.139774, -2.363176, -1.50971]



target_pos_apple_ts=[q1 q2 q3 q4 q5 q6]
target_pos_apple_ts=[51.5662 -107.7000 -122.6000 -135.4000 -86.5000 0]

INIZIALIZZAZIONE DEI MOTORI

```
# Speed of UR5e Robot  
speed = 2
```

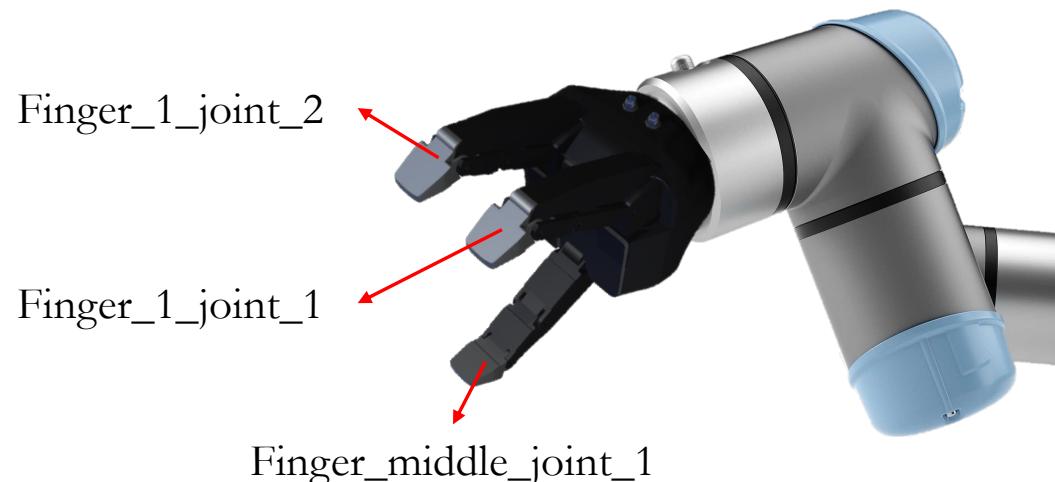


Imposta la velocità del robot UR5e

```
# Getting and declaring the 3 finger motors of the 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'))
```

Ottiene e dichiara i motori delle dita del finger e li aggiunge alla lista 'hand_motors'



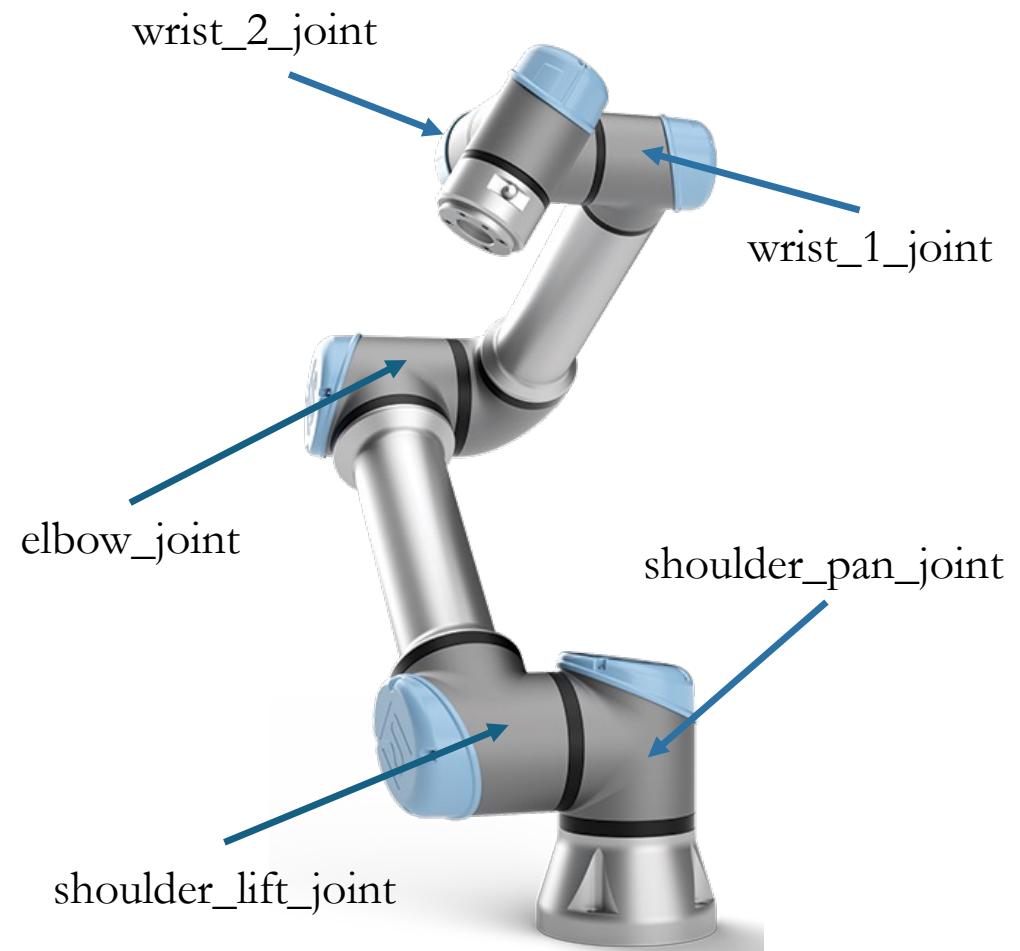
INIZIALIZZAZIONE DEI MOTORI

```
# Getting and declaring the robot motor
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'))
```

Ottiene e dichiara i motori del robot
e li aggiunge alla lista **ur_motors**

```
# Seting velocity of UR5e motors
for i in range(5):
    ur_motors[i].setVelocity(speed)
```

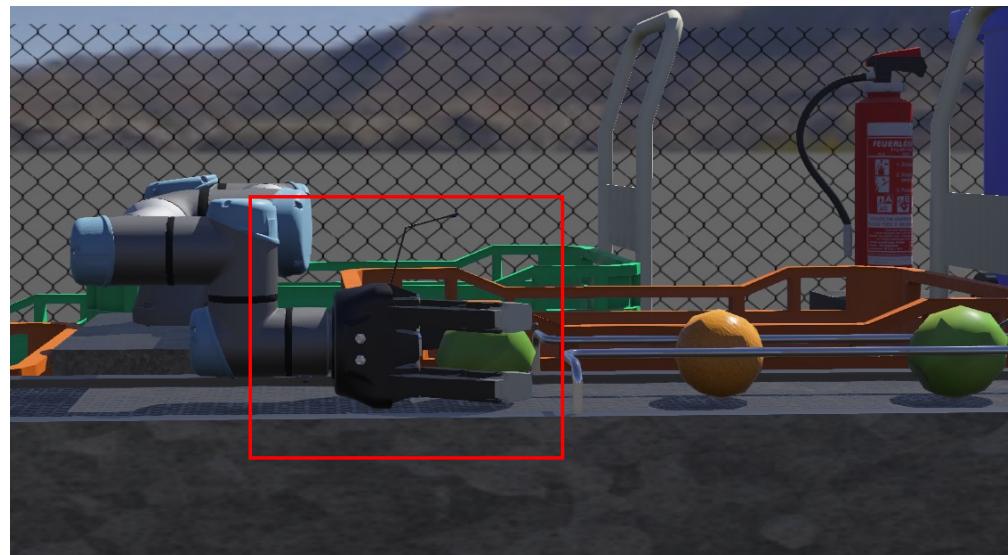
Imposta la velocità dei motori del robot



SENSORI DI PROSSIMITÀ E POSIZIONE

```
# Getting and declaring distance sensor of gripper  
distance_sensor = robot.getDevice('distance sensor')  
distance_sensor.enable(timestep)  
  
# Getting and declaring position sensor of wrist robot  
position_sensor = robot.getDevice('wrist_1_joint_sensor')  
position_sensor.enable(timestep)
```

Ottiene e abilita il **sensore di distanza** e il **sensore di posizione** del polso del robot



CAMERE E DISPLAY

```
# Initialize camera
camera = robot.getDevice('camera') → Il codice inizializza e abilita una telecamera del robot,
camera.enable(timestep)           impostando la frequenza di campionamento per acquisire
                                    immagini.

# Initialize display
display = robot.getDevice('display')
display.attachCamera(camera) → Il codice inizializza un display, collega la telecamera ad esso
display.setColor(0x00FF00)          per mostrare il feed video, imposta il colore di disegno e il
display.setFont('Verdana', 16, True) font del testo

def resetDisplay():
    display.setAlpha(0.0) → Cancella il display rendendolo completamente trasparente
    display.fillRect(0, 0, 200, 150) (0), riempiendo un rettangolo con le dimensioni specificate,
    display.setAlpha(1.0)           e poi reimpostando la trasparenza a opaca (1).

def printDisplay(x, y, w, h, name): → Ripulisce il display chiamando la funzione 'resetDisplay',
    resetDisplay()                disegna un rettangolo con le coordinate e dimensioni
    display.drawRect(x, y, w, h)   specificate, e poi disegna un testo.
    display.drawText(name, x - 2, y - 20)
```

CAMERE E DISPLAY

```
def findFruit():          → Definizione funzione findfruit():  
    min = []  
    max = []  
    cnts = []  
    mask = []  
    model = -1  
    # Fruit names  
    fname = ['Orange', 'Apple', 'Rottenapple']  
  
    img = np.frombuffer(camera.getImage(), dtype=np.uint8).reshape((camera.getHeight(), camera.getWidth(), 4))  
    roi = img[0:150, 35:165]  
    imHSV = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)  
  
    # Orange HSV color range  
    min.append(np.array([10, 135, 135], np.uint8))  
    max.append(np.array([32, 255, 255], np.uint8))  
  
    # Green HSV color range  
    min.append(np.array([30, 50, 50], np.uint8))  
    max.append(np.array([90, 255, 255], np.uint8))  
  
    # Black HSV color range  
    min.append(np.array([0, 0, 0], np.uint8))  
    max.append(np.array([179, 50, 30], np.uint8))
```

→ Definizione dei range di colori per ogni tipo di frutta

Definizione funzione **findfruit()**:

- Print a schermo del nome del frutto dopo averlo riconosciuto
- Restituisce **model**, un numero legato al tipo di frutto (0=Orange, 1=Apple, 2=Rottenapple)

IMPLEMENTAZIONE FILTRO DI KERNEL

```
# Kernel filter
Kernel = np.ones((5, 5), np.uint8)

for i in range(3):
    mask.append(cv2.inRange(imHSV, min[i], max[i]))

# Morphological transformations with orange mask and kernel
mask[i] = cv2.morphologyEx(mask[i], cv2.MORPH_CLOSE, Kernel)
mask[i] = cv2.morphologyEx(mask[i], cv2.MORPH_OPEN, Kernel)

# Finding contours
cnts.append(cv2.findContours(mask[i], cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0])

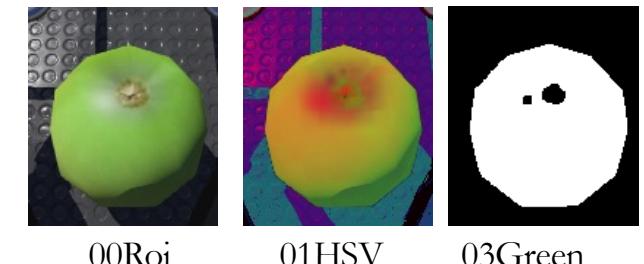
for c in cnts[i]:
    x, y, w, h = cv2.boundingRect(c)
    if w > 80:
        model = i
        printDisplay(x + 35, y, w, h, fname[i])

"""
Test images
cv2.imwrite('00Roi.png', roi)
cv2.imwrite('01HSV.png', imHSV)
cv2.imwrite('02Orange.png', mask[0])
cv2.imwrite('03Green.png', mask[1])
"""

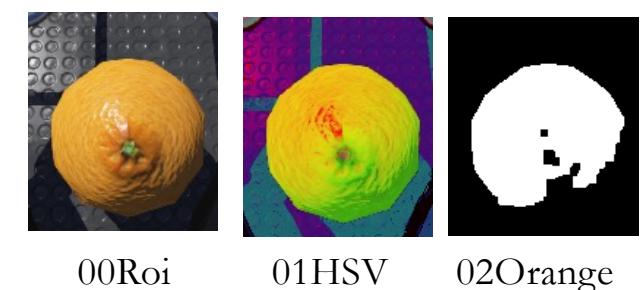
return model
```



Inizializzazione filtro di Kernel
Questo processo è utile per il rilevamento e la visualizzazione di oggetti di colore specifico in un'immagine.



Disegno di un rettangolo attorno al frutto riconosciuto e print a schermo del relativo nome



STATO 0 - WAITING

```
# Main Loop:  
while robot.step(timestep) != -1:  
    # ifs for the different state of the arm  
  
    if counter <= 0:  
        if state == 0: # WAITING  
            fruit = findFruit()  
            if distance_sensor.getValue() < 500:  
                state = 1 # PICKING  
                playSnd(fruit)  
                if fruit == 0:  
                    orange += 1  
                elif fruit == 1:  
                    apple += 1  
                elif fruit == 2:  
                    rottenapple += 1  
            counter = 8  
        for i in range(3):  
            hand_motors[i].setPosition(0.52)
```



STATO 0 - WAITING

```
# Main Loop:  
while robot.step(timestep) != -1:      →  
    # ifs for the different state of the arm  
  
  
if counter <= 0:  
    if state == 0: # WAITING  
        fruit = findFruit()  
  
  
    if distance_sensor.getValue() < 500:  →  
        state = 1 # PICKING  
        if fruit == 0:  
            orange += 1  
        elif fruit == 1:  
            apple += 1  
        elif fruit == 2:  
            rottenapple += 1  
        counter = 8  
        for i in range(3):  
            hand_motors[i].setPosition(0.52) →
```

Loop principale: Esegue passi di simulazione a ogni iterazione. Il ciclo continua finché la simulazione non viene terminata.

Identificazione del frutto: Chiama la funzione **findFruit** per identificare il frutto.

Controllo del sensore di distanza: Se la distanza tra il sensore dell'end-effector e l'oggetto che trasla sul rullo è < di 500 μm

Passa allo stato 1 di PICKING

Aggiornamento contatori frutti

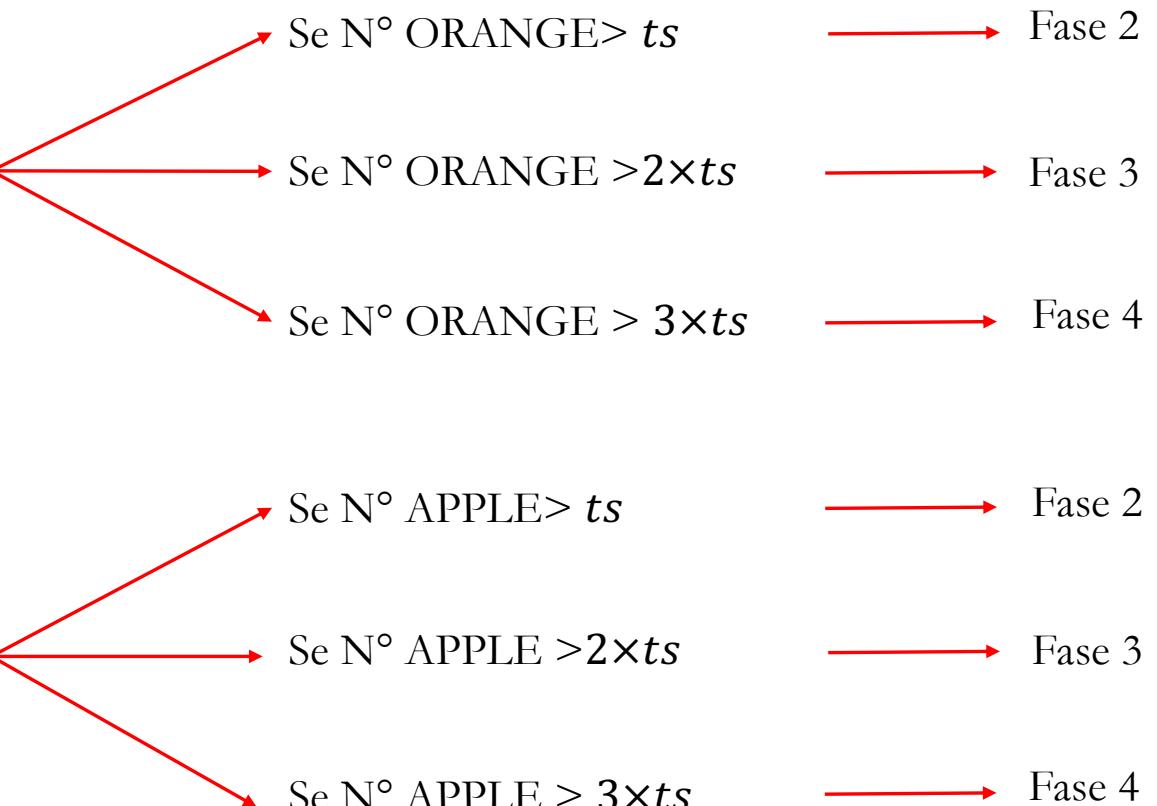
Imposta la posizione dei motori del finger a 0.52 per afferrare il frutto

STATO 0 - WAITING

Aggiornamento posizioni target: In base ai contatori dei frutti, aggiorna le posizioni target

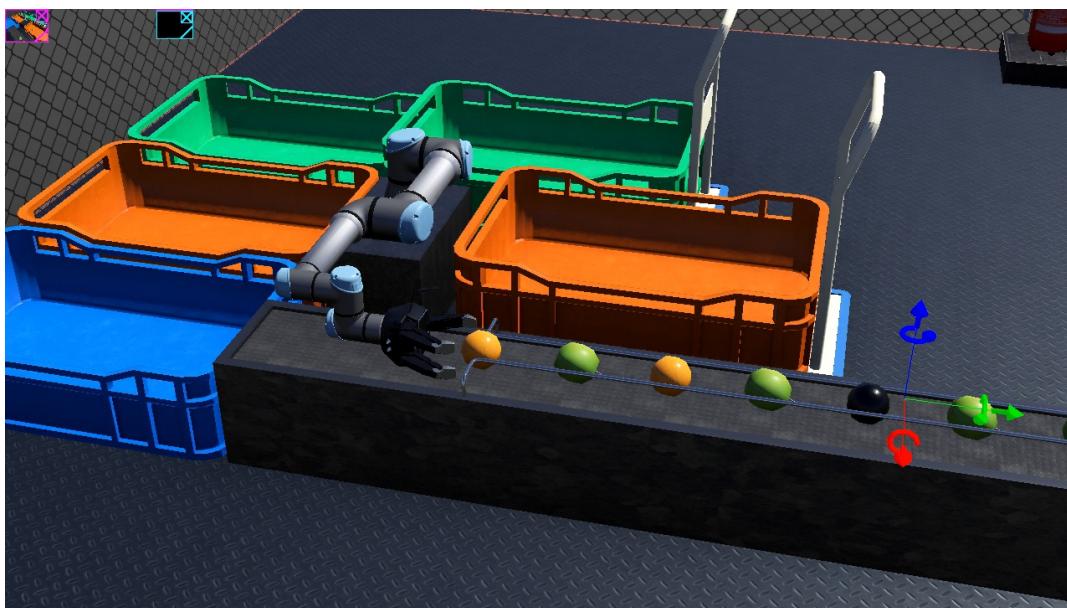
```
# Switch the positions
if orange>ts:
    target_pos[0] = target_pos_orange_ts
    if orange>2*ts:
        target_pos[0] = target_pos_apple
        if orange>3*ts:
            target_pos[0] = target_pos_apple_ts
```

```
if apple>ts:
    target_pos[1] = target_pos_apple_ts
    if apple>2*ts:
        target_pos[1] = target_pos_orange
        if apple>3*ts:
            target_pos[1] = target_pos_orange_ts
```



STATO 1 - PICKING

```
elif state == 1: # PICKING
    for i in range(5):
        if fruit == 0 or fruit ==1:
            ur_motors[i].setPosition(target_positions[i])
    elif fruit ==2:
        for i in range(0,5):
            ur_motors[i].setPosition(target_positions[i])
state = 2 # ROTATING
```



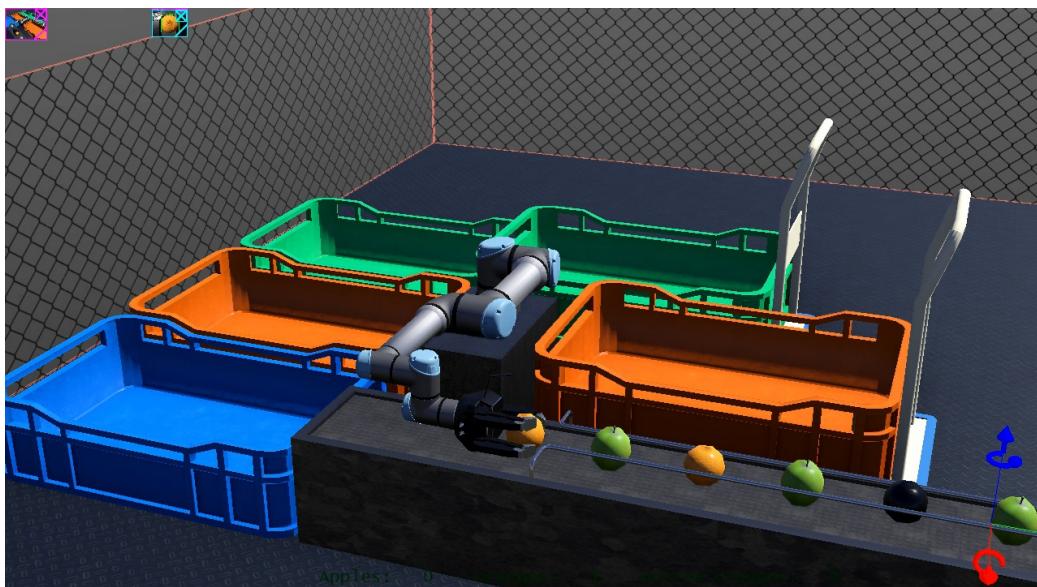
Ciclo di controllo per un robot che si trova nello stato di **PICKING** → prende il frutto

A seconda del tipo di frutto (**fruit**), il codice imposta le posizioni dei motori del robot (**ur_motors**) e quindi cambia lo stato del robot a **ROTATING**.

Passa allo stato 2 di **ROTATING**

STATO 2 - ROTATING

```
elif state == 2: # ROTATING
    if position_sensor.getValue() < dval:
        counter = 8
        state = 3 # DROPPING
        resetDisplay()
        for i in range(3):
            hand_motors[i].setPosition(hand_motors[i].getMinPosition())
    state=3
    if tot==10:
        state = 5 # UP
```



Il codice controlla il valore del sensore di posizione e, in base a questo valore (dval), aggiorna lo stato del robot e imposta le posizioni dei motori della mano.

Per ogni motore nella lista hand_motors viene impostata la posizione minima utilizzando il metodo setPosition con il valore ottenuto da getMinPosition (0,0,0).
Apre le dita in una posizione minima, in modo da passare allo stato di dropping.

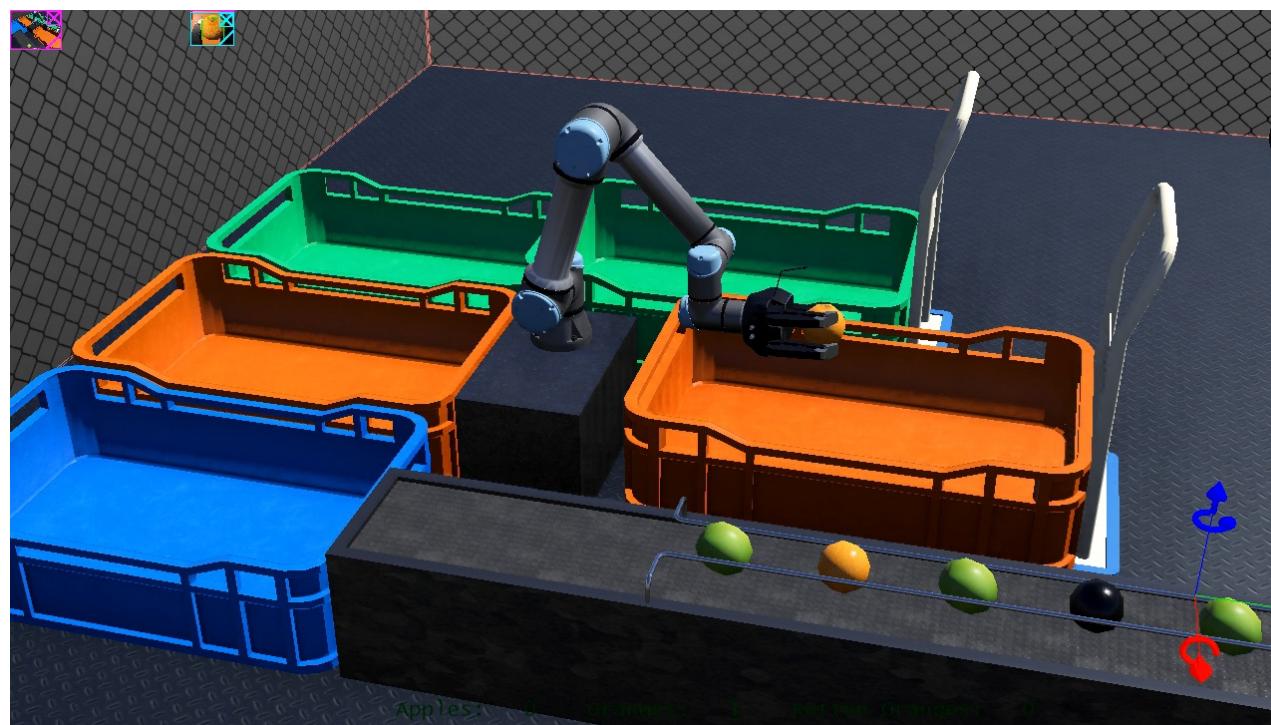
tot è la somma di arance e mele, che identifica la fine della fase 4.
Raggiunto il totale si passa allo stato 5 di UP

STATO 3 - DROPPING

```
elif state == 3: # DROPPING
    for i in range(5):
        ur_motors[i].setPosition(0.0)
state = 4 # UP
```



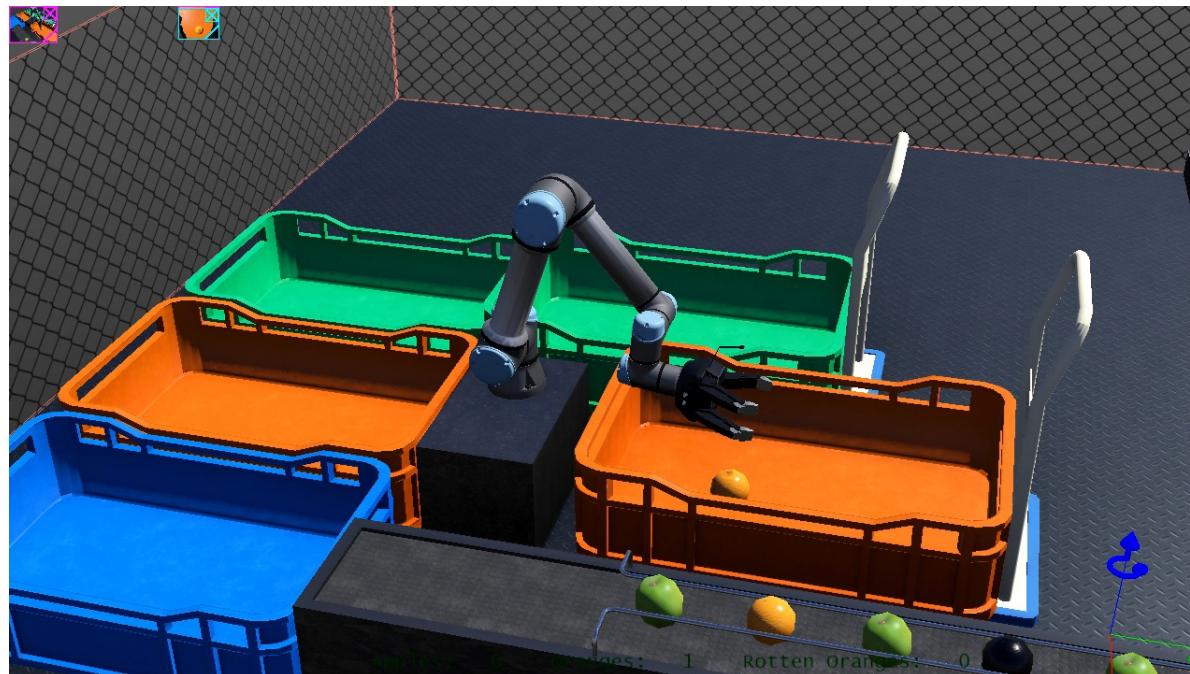
Il robot rilascia il frutto nell'apposito contenitore impostando la posizione a 0 e passa allo stato 4



STATO 4 - ROTATE BACK

```
elif state == 4: # ROTATE_BACK
    for i in range(5):
        ur_motors[i].setPosition(0.0)
    if position_sensor.getValue() > -0.1:
        state=0 # WAITING
```

Il robot ritorna indietro alle posizioni iniziali e si mette in stato di **waiting** se il valore del sensore di posizione è maggiore di -0,1

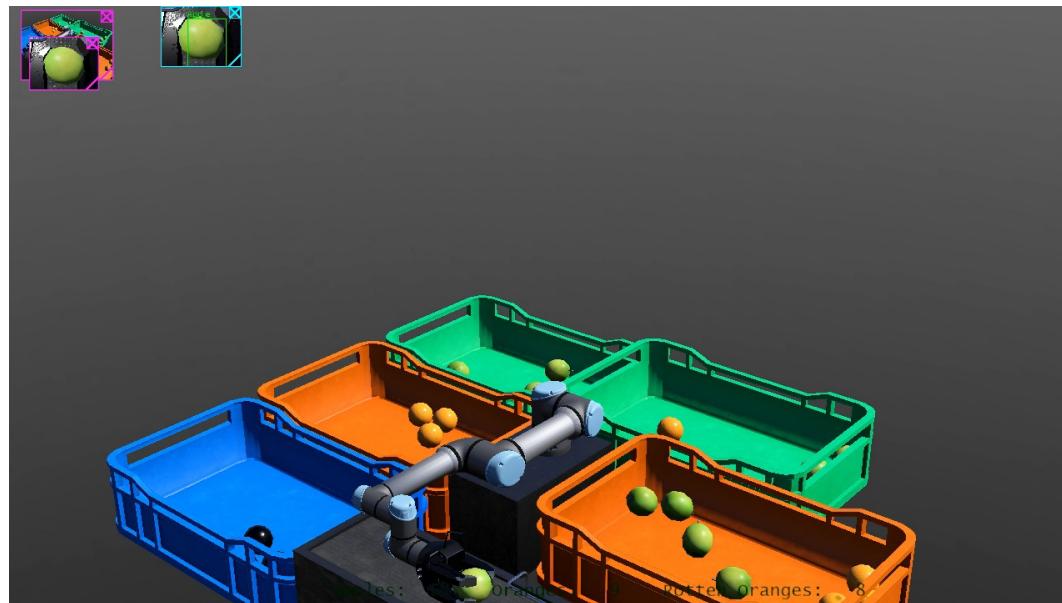


STATO 5 - UP

Stato in cui il robot entra **una sola volta** alla fine della fase 4:

```
elif state==5: # UP
    for i in range(0,5):
        target_positions = [-1.570796, -1.57, 0, 0, 1.50971]
        ur_motors[i].setPosition(target_positions[i])
    wait(4000)
    state=4 # WAITING
```

Setta il robot in posizione verticale e attende 4 secondi, poi passa a stato 4 (rotate back)



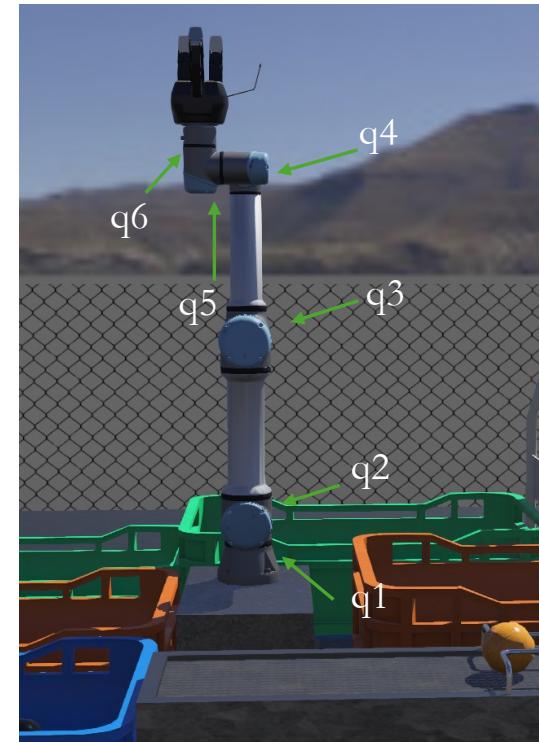
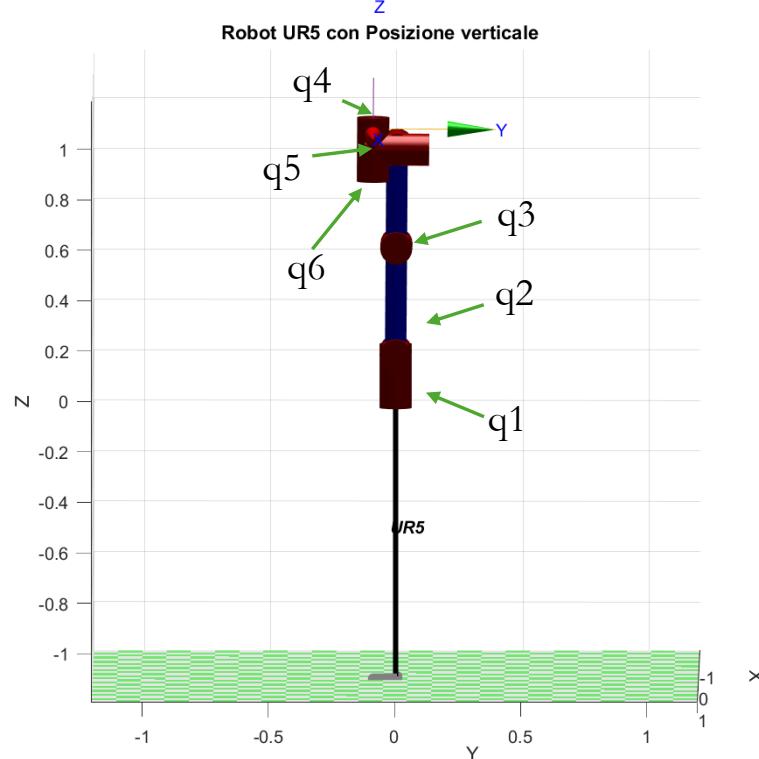
STATO 5 - UP

Teach

X:	0.109
Y:	-0.088
Z:	0.990
R:	-0.0
P:	0.0
Y:	-0.4

q1 [90.0] q2 [-90.4] q3 [0.0] q4 [0.0] q5 [90.0] q6 [0.0]

X



target_positions = [q1 q2 q3 q4 q5 q6]
 target_positions = [-1.570796 -1.57 0 0 1.50971 0] RADIANTI
 target_positions = [-90.0000 -89.9544 0 0 86.5000 0] GRADI

STRINGA DEL COUNTER

Si definisce e si visualizza il contatore della frutta:

```
else:  
    counter -= 1  
  
    strP = f'Apples: {apple:3d}      Oranges: {orange:3d}      Rotten Apple: {rottenapple:3d}'  
    robot.setLabel(1, strP, 0.3, 0.96, 0.06, 0x003500, 0, 'Lucida Console')  
  
pass
```

ESEMPIO FASE 1

