



**Università
degli Studi
di Palermo**



Corso di Laurea Magistrale in **Ingegneria Informatica**
Progetto di **Robotica Medica**

A cura di: Ambrogi Federico Ennio
Burgio Gabriele
Masi Luca
Sollazzo Ivan

Cybersterilizer

La robotica che fa **luce** sul domani della sanità

01

Introduzione

L'impatto della robotica sulla medicina



Precisione superiore

I robot riducono il margine di errore in operazioni sensibili.

Ripetibilità e affidabilità

Ogni procedura può essere eseguita con la stessa qualità, senza cali rilevanti di attenzione.

Tempi di lavoro e costi ridotti

I tempi di lavoro, i corsi e i ritardi nei processi opedalieri possono essere ridotti drasticamente.

L'impatto dell'informatica sulla medicina

Visione artificiale

La possibilità di consentire ai calcolatori di “vedere il mondo” ha dato luogo a molteplici soluzioni di automazione moderne.

Potenza di calcolo

L'avvento di calcolatori sempre più potenti ha reso possibile portare a compimento task pesanti senza particolari rinunce.

Ingegneria del software

Il software progettato con principi e modelli sempre più rigorosi ha permesso di eseguire processi di automazione con maggior fiducia e sicurezza.



Vogliamo fare luce sul
domani della sanità

Cos'è Cybersterilizer

È un sistema robotico sviluppato per automatizzare e ottimizzare il processo di **sterilizzazione** di un'**area operatoria** attraverso l'uso combinato della visione artificiale e la tecnologia UV.

Goal del progetto

Sterilizzazione intelligente

Il robot è in grado di scansionare un'area delimitata da dei marcatori e applicare un protocollo di rilevamento e sterilizzazione dei batteri.

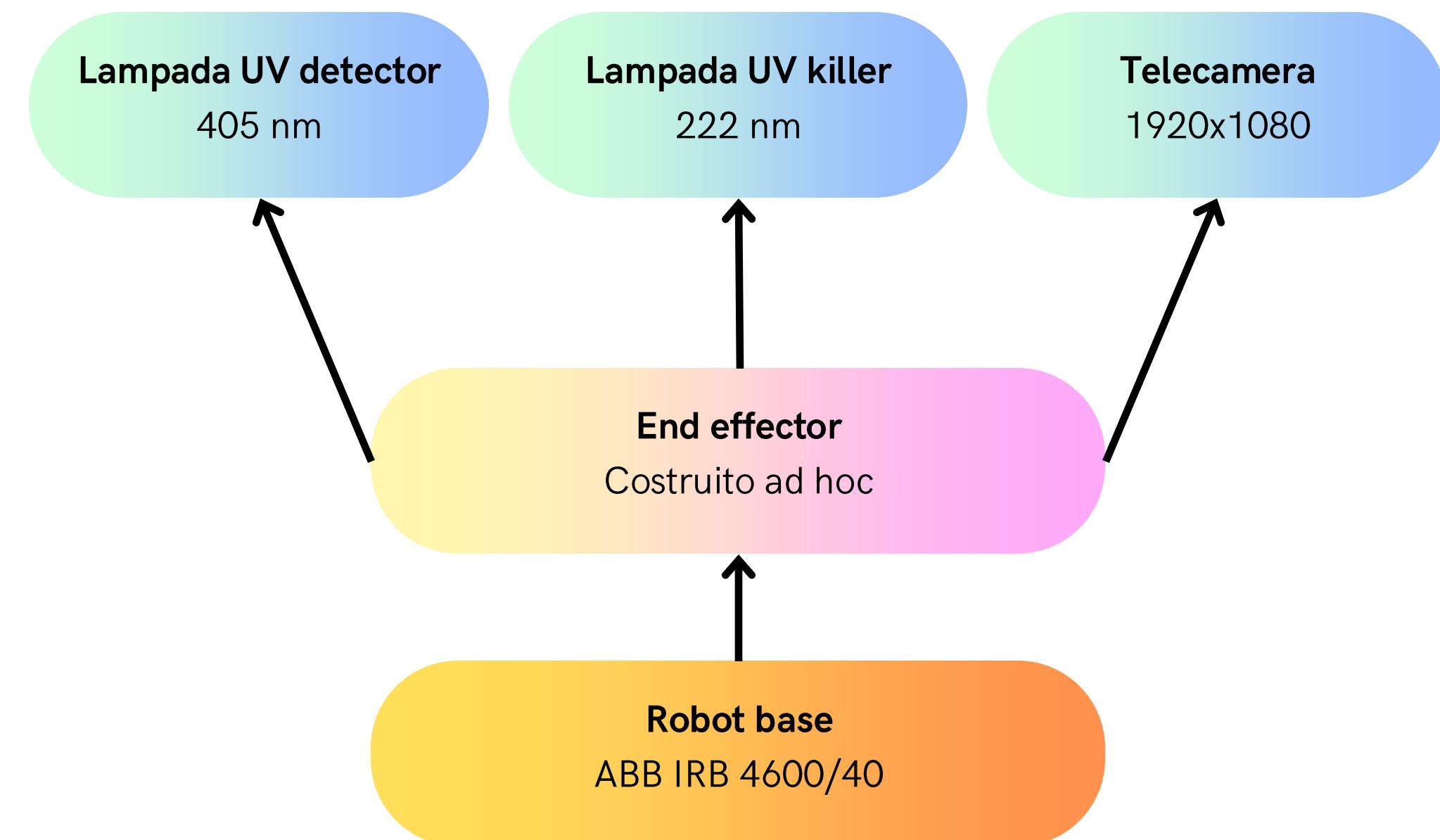
Ripetibilità del processo

Il robot esegue ogni ciclo di sterilizzazione seguendo sequenze programmate e ottimizzate, prive di fluttuazioni fisiologiche tipiche dell'intervento manuale.

Riduzione dell'errore umano

Mentre l'essere umano è soggetto a distrazioni, affaticamento o approssimazioni involontarie, il braccio robotico opera con costanza e precisione millimetrica.

Architettura del sistema



02

Specifiche tecniche del robot



ABB IRB 4600/40

Gradi di libertà	6
Tipologia dei giunti	rotoidale
Area del basamento	512x676 mm ²
Altezza	1922 mm
Peso	465 kg
Ripetibilità di posizione (a 250 mm/s)	0.06 mm

Per un elenco completo delle specifiche è possibile consultare il datasheet ufficiale e la documentazione del progetto.

03

Risoluzione della cinematica diretta

Convenzione di Denavit-Hartenberg

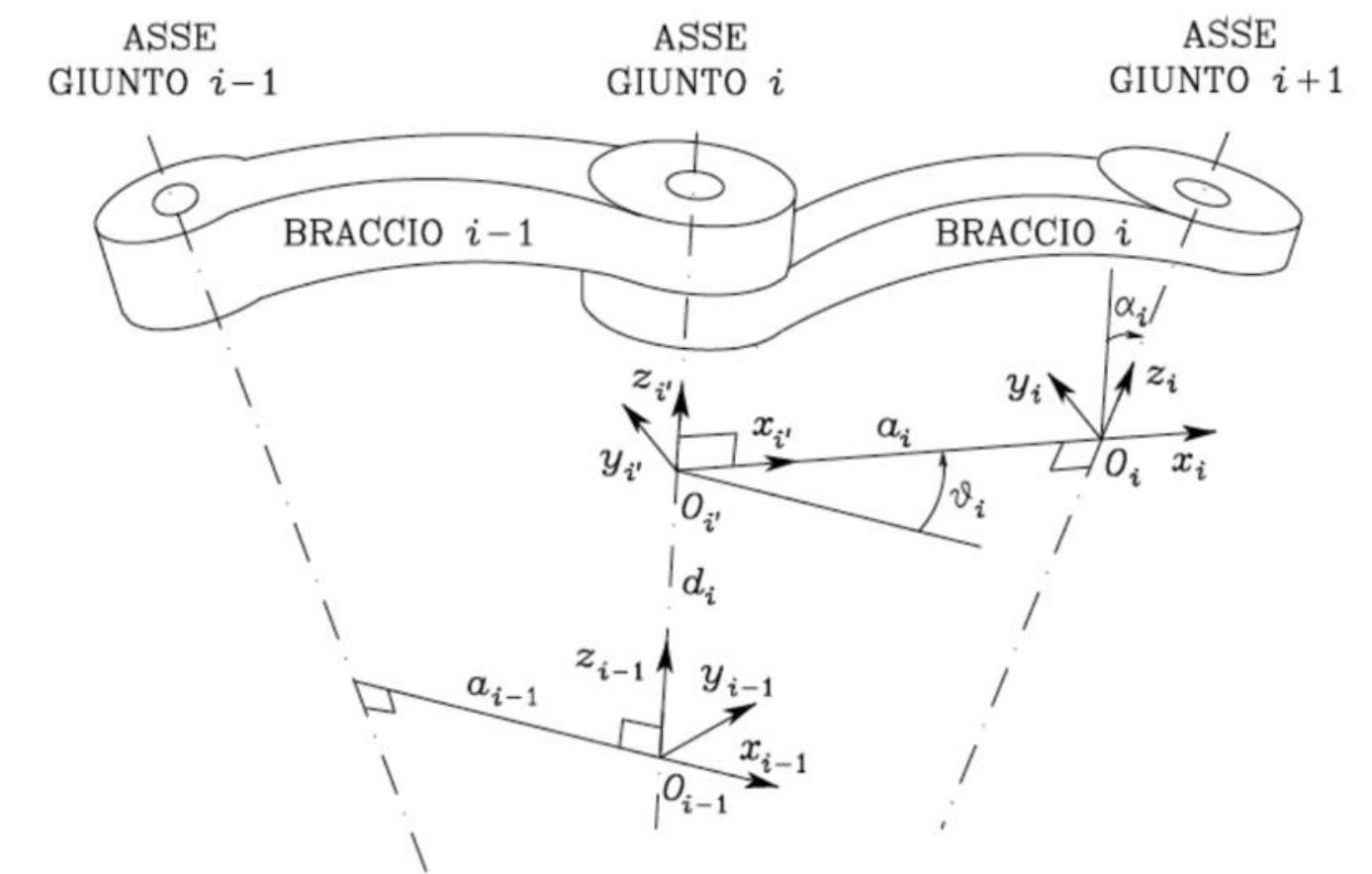
Consente di ridurre il numero di parametri per passare da una terna all'altra (quattro invece di sei). Si associa

a ciascun giunto una terna levogira in cui:

- L'asse **z** è diretto lungo l'asse di rotazione (giunto rotoidale) o di traslazione (giunto prismatico);
- L'asse **x** è diretto lungo la normale comune (qualora univoca) agli assi **z** del giunto **i-1** e del giunto **i**, preso positivo dal giunto **i** al giunto **i+1**;
- Si sceglie l'asse **y** in modo da completare la terna levogira.

$$M_i^{i-1} = \begin{pmatrix} c_{\theta_i} & -c_{\alpha_i}s_{\theta_i} & s_{\alpha_i}s_{\theta_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\alpha_i}c_{\theta_i} & -s_{\alpha_i}c_{\theta_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_i^{i-1} = T(0, 0, d_i) \cdot R(z, \theta_i) \cdot T(a_i, 0, 0) \cdot R(x, \alpha_i)$$



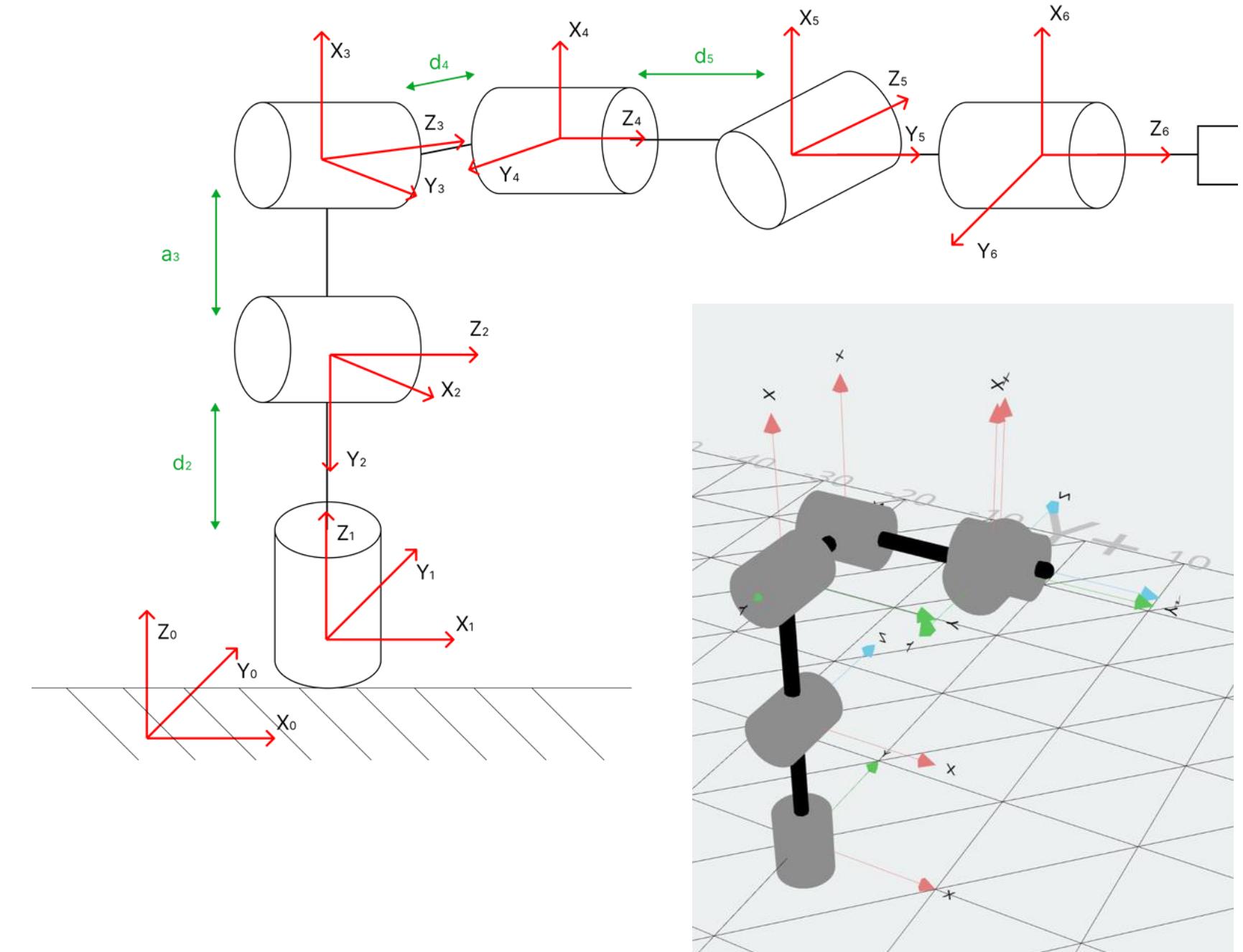
1. Traslazione lungo l'asse z_{i-1} di una quantità d_i ;
2. Rotazione attorno l'asse z_{i-1} di un angolo θ_i
3. Traslazione lungo l'asse x_i di una quantità a_i
4. Rotazione attorno l'asse x_i di un angolo α_i

α_i è preso positivo nel caso di rotazione antioraria

Tabella di Denavit-Hartenberg

Giunto	θ_i	α_i (dg.)	a_i (m)	d_i (m)
1	θ_1	0	0	0
2	θ_2	-90	0	0.175
3	θ_3	0	1.095	0
4	θ_4	-90	0	0.175
5	θ_5	90	0	1.270
6	θ_6	-90	0	0

Valori determinati secondo le specifiche tecniche fornite dal datasheet.
Tabella popolata secondo le convenzioni scelte per le terne.



Matrice di trasformazione

$$M_6^0 = \begin{pmatrix} c_6\sigma_2 - c_{12}s_{34}s_6 & s_{12}c_5 - c_{12}c_{34}s_5 & -s_6\sigma_2 - c_{12}s_{34}c_6 & 1.095c_{12}c_3 - 1.27c_{12}s_{34} - 0.175s_{12} \\ -c_6\sigma_1 - s_{12}s_{34}s_6 & -c_{12}c_5 - c_{34}s_{12}s_5 & s_6\sigma_1 - s_{12}s_{34}c_6 & 0.175c_{12} - 1.27s_{12}s_{34} + 1.095s_{12}c_3 \\ -s_6\sigma_3 - c_5c_6\sigma_4 & s_{34}s_5 & c_5s_6\sigma_4 - c_6\sigma_3 & 0.175 - 1.095s_3 - 1.27c_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\sigma_1 = c_{12}s_5 - c_{34}s_{12}c_5$$

$$\sigma_2 = s_{12}s_5 + c_{12}c_{34}c_5$$

$$\sigma_3 = c_3c_4 - s_3s_4$$

$$\sigma_4 = c_3s_4 + c_4s_3$$

Per compattare la matrice, si usano le seguenti definizioni

$$c_{ij} = \cos(\theta_i + \theta_j)$$

$$s_{ij} = \sin(\theta_i + \theta_j)$$

Viene utilizzata come riferimento la seguente matrice di trasformazione tra un giunto e l'altro

$$M_i^{i-1} = \begin{pmatrix} c_{\theta_i} & -c_{\alpha_i}s_{\theta_i} & s_{\alpha_i}s_{\theta_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\alpha_i}c_{\theta_i} & -s_{\alpha_i}c_{\theta_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

04

Risoluzione della cinematica inversa

Algoritmo di risoluzione base

Dobbiamo trovare quel **vettore delle configurazioni** tale che in funzione di tale vettore otteniamo la posizione target.

$$f(\vec{q}) = T_{\text{target}}$$

Si definisce una funzione obiettivo che misura l'errore.

$$E(\vec{q}) = \|J\Delta\vec{q} - \vec{e}\|^2$$

L'obiettivo è minimizzare questo errore. A tale scopo, si utilizza un algoritmo iterativo che si avvale della pseudoinversa dello jacobiano analitico. In particolare, ad ogni passo si calcola la seguente variazione del vettore delle configurazioni:

$$\Delta\vec{q}_k = J_k^+ \vec{e}_k$$

Si aggiorna il vettore di configurazione con:

$$\vec{q}_{k+1} = \vec{q}_k + \Delta\vec{q}_k$$

Si ripete l'algoritmo iterativamente per un numero finito di passi oppure fino a quando l'errore non si trova al di sotto di una certa soglia.

$$J = \begin{pmatrix} \frac{\partial m_{11}}{\partial \theta_1} & \dots & \frac{\partial m_{11}}{\partial \theta_6} \\ \frac{\partial m_{21}}{\partial \theta_1} & \dots & \frac{\partial m_{21}}{\partial \theta_6} \\ \vdots & \dots & \vdots \\ \frac{\partial m_{34}}{\partial \theta_1} & \dots & \frac{\partial m_{34}}{\partial \theta_6} \end{pmatrix}$$

La libreria ikpy

Consente di costruire una catena cinematica partendo da un file URDF o manualmente prendendo i dati dalla tabella di Denavit-Hartenberg.

La catena cinematica viene usata poi per risolvere la cinematica diretta e inversa tramite l'algoritmo precedentemente mostrato.

Implementazione in Cybersterilizer

Il robot usato in Cybersterilizer dispone di un file URDF contenente già le specifiche sulla catena cinematica pensate per la simulazione in Webots.

Estrazione del file URDF dal modello

Webots

```
filename = None
with tempfile.NamedTemporaryFile(suffix='.urdf', delete=False) as file:
    filename = file.name
    file.write(supervisor.getUrdf().encode('utf-8'))
```

Istanziazione della catena

cinematica

```
end_effector_offset = [0.0, 0.0, 0.3]

chain = Chain.from_urdf_file(filename, last_link_vector=end_effector_offset,
active_links_mask=[False, True, True, True, True, True, True, False, False, False])
```

Il link di basamento è False poiché non deve ruotare. Gli ultimi tre False sono dei link di offset (non realmente visibili) usati ai fini della simulazione.

Implementazione in Cybersterilizer

Risoluzione della cinematica

diretta

Il metodo forward_kinematics richiamato sulla catena restituisce la matrice di trasformazione 4x4 di posizione e orientamento dell'end effector con riferimento alla terna base.

```
fk =  
chain.forward_kinematics(joint_angles)
```

Risoluzione della cinematica

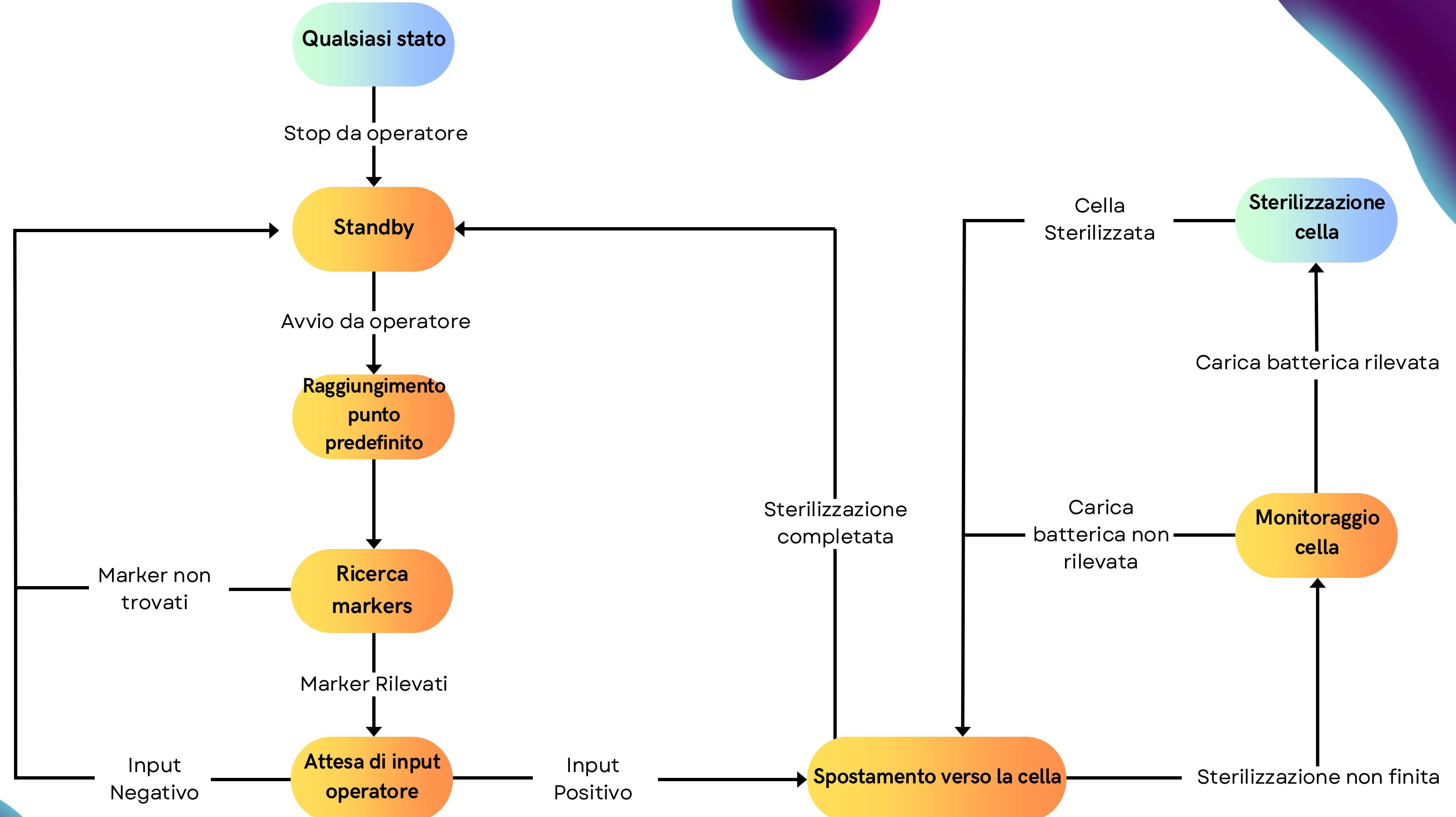
inversa

Il metodo inverse_kinematics accetta le coordinate nello spazio di lavoro e altri parametri come il numero di iterazioni da eseguire per minimizzare l'errore.

```
target_joint_positions = chain.inverse_kinematics(ik_target_robot,  
target_orientation=target_orientation_matrix_down, orientation_mode="all",  
max_iter=100, initial_position=initial_joint_positions)
```

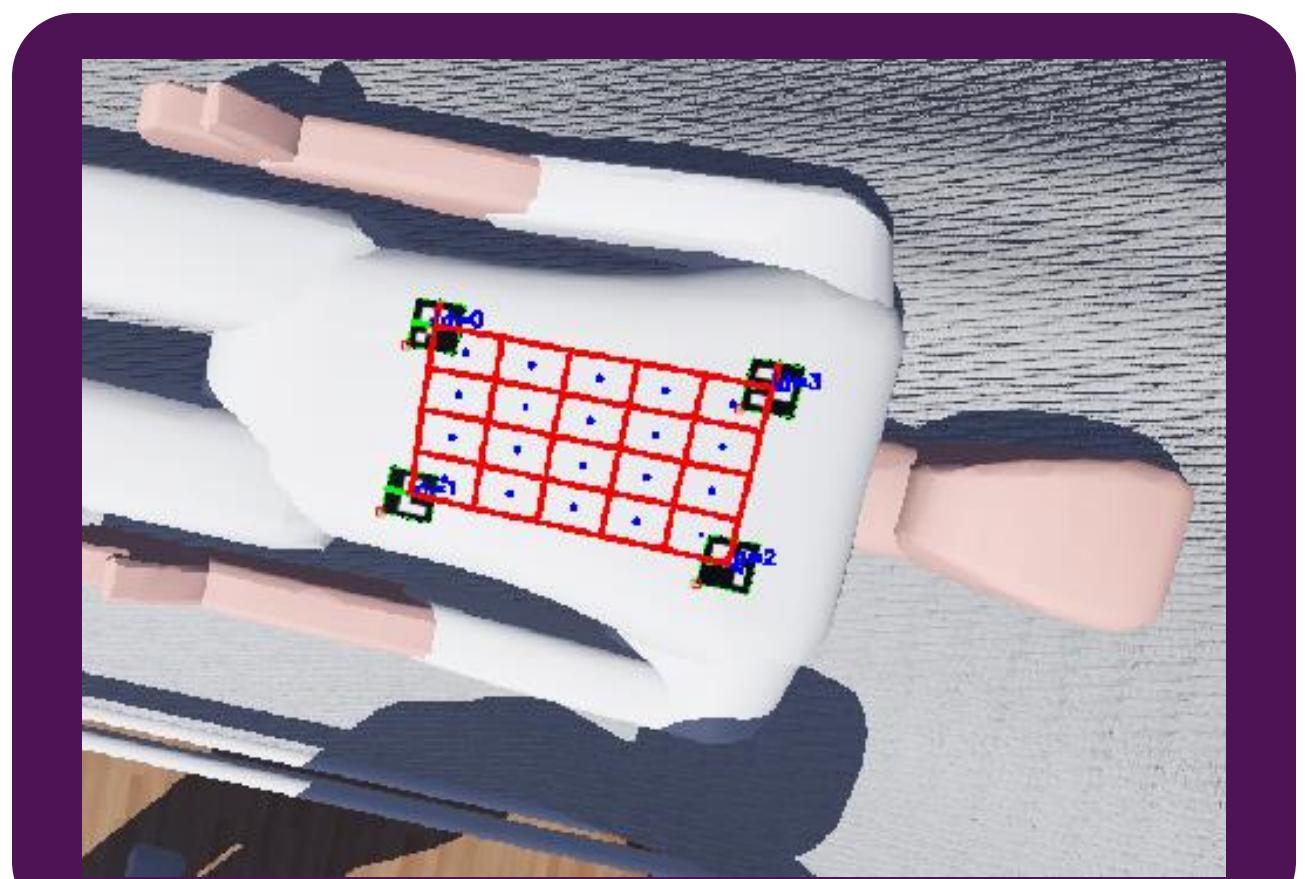
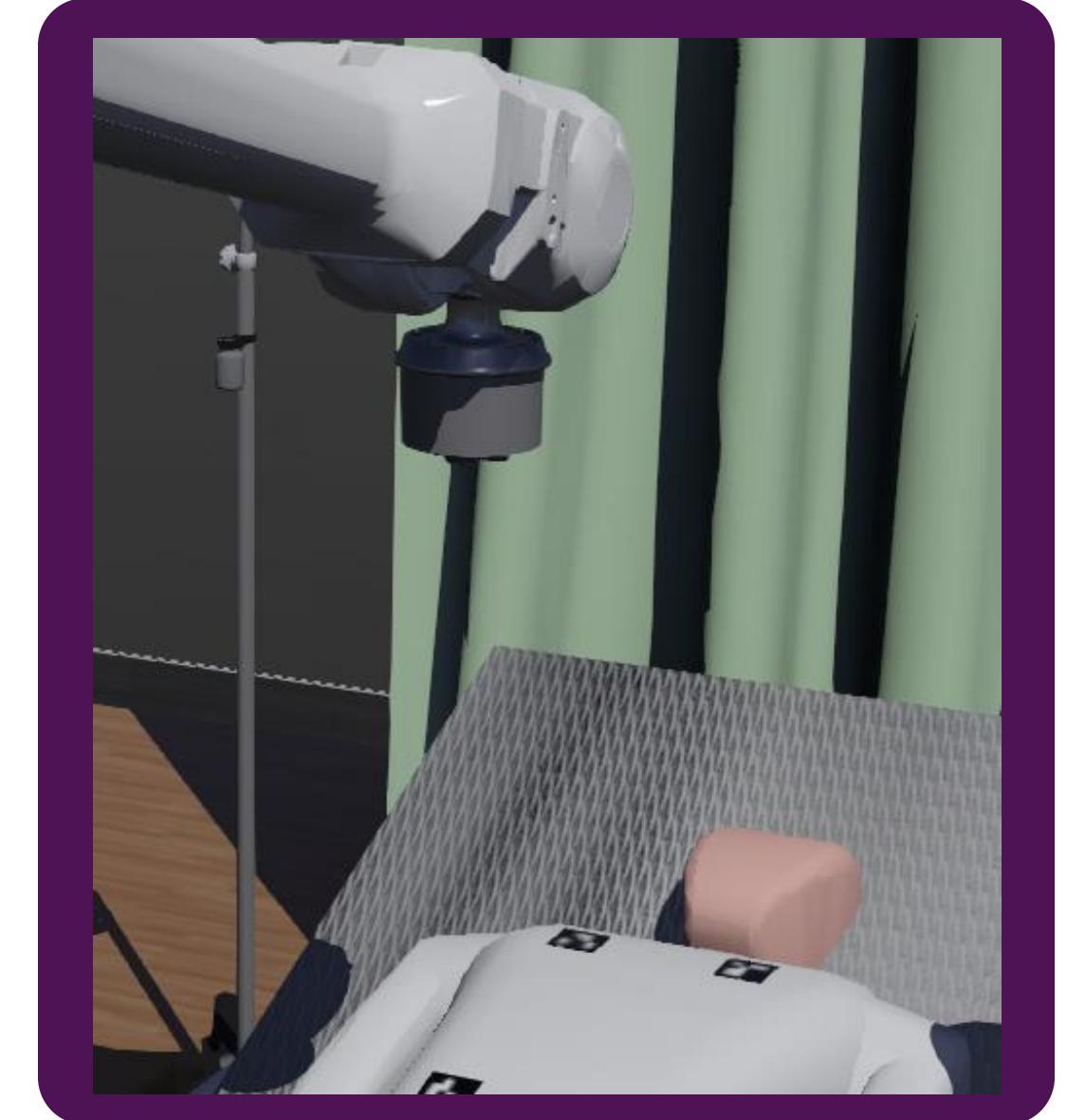
06

Controllore



Posizionamento e ricerca dell'area sterile

In questa fase il robot si colloca in una posizione ottimale seguendo una traiettoria articolare per individuare, tramite telecamera, l'area delimitata dai marcatori.

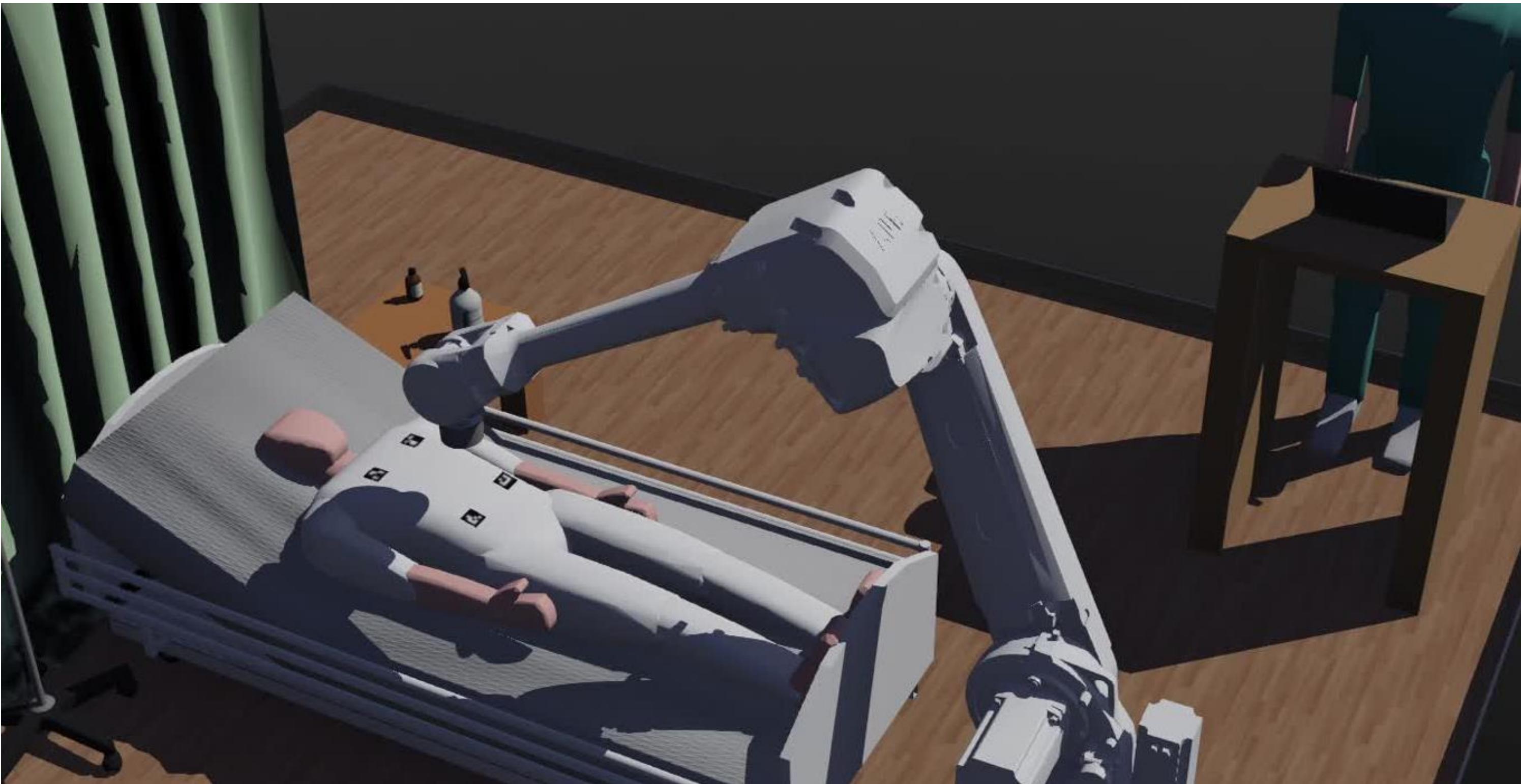


Sterilizzazione

In questa fase, il robot si sposta progressivamente da una cella all'altra della griglia delimitata attivando di volta in volta la luce UV-A per rilevare presenza batterica, e in caso di rilevamento, viene disattivata la luce UV-A e attivata la luce UV-C per procedere alla sterilizzazione.



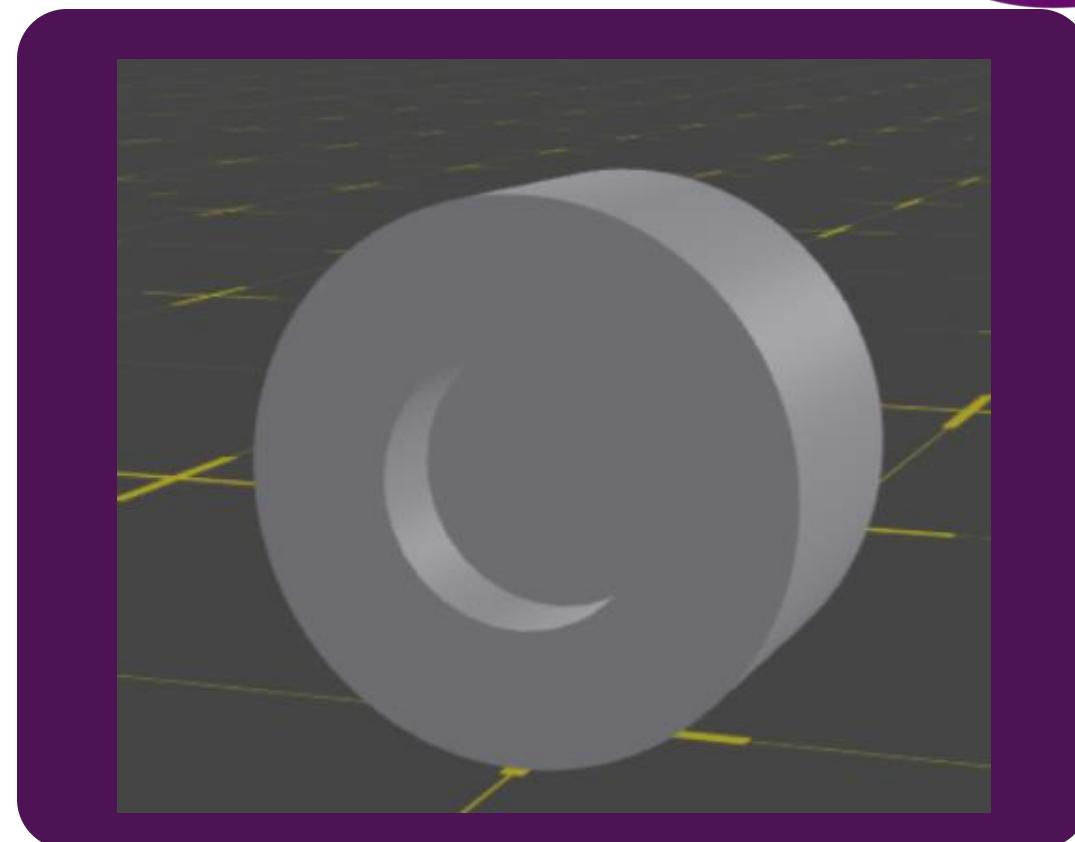
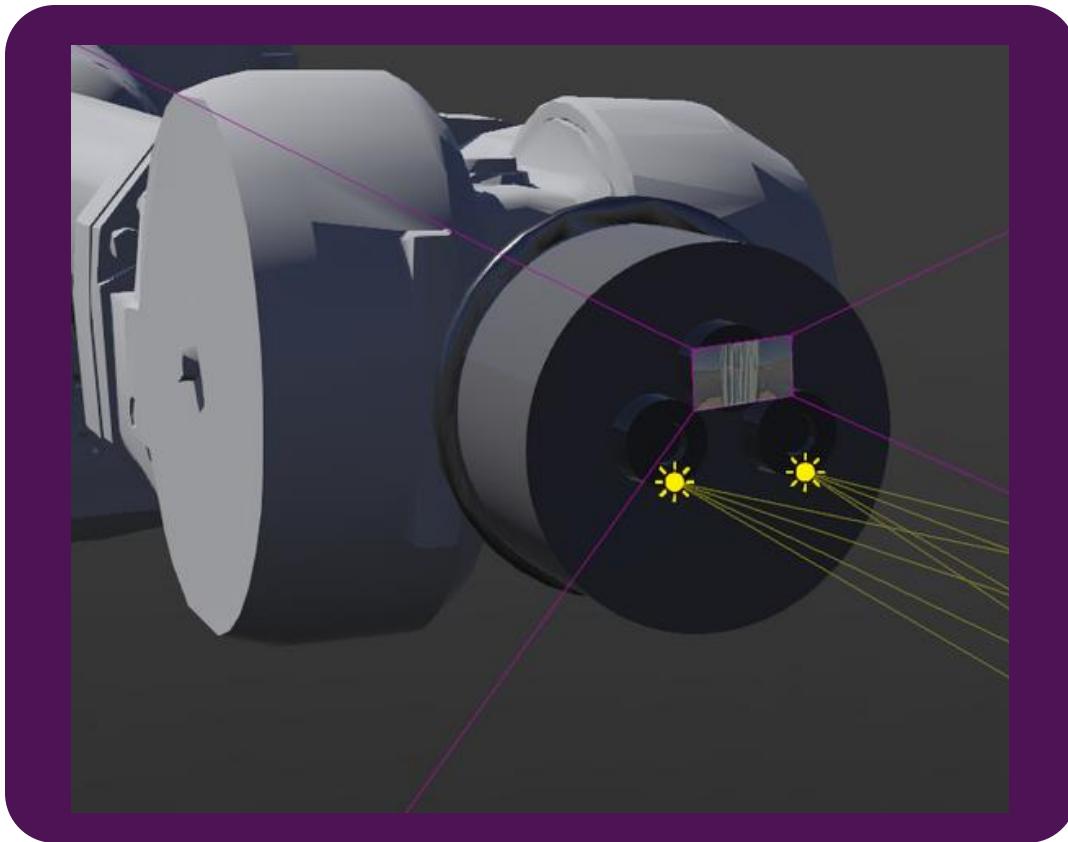
Procedura di Sicurezza



07

End effector

L'end effector è il modulo compatto a forma cilindrica montato saldamente sul polso dell'ABB IRB 4600. La progettazione del modello 3D è stata effettuata tramite il software CAD Shapr3D



Architettura dell'end effector

Videocamera

Utilizzata per il rilevamento degli appositi marker per delimitare con precisione la zona operativa. In fase di sterilizzazione, invece, viene utilizzata per la rilevazione della presenza batterica attraverso algoritmi di Computer Vision.

Lampada UV-A

Utilizzata per emettere radiazioni ultraviolette con lunghezza d'onda di 405 nanometri, che causa fluorescenza nei batteri.

Lampada UV-C

Utilizzata per emettere radiazioni ultraviolette con lunghezza d'onda di 222 nanometri, che ha una funzione germicida.

08

Il caso studio

Rischio di Infezioni del Sito Chirurgico

Ancora oggi, il rischio di infezioni chirurgiche in ospedale, rappresenta una sfida quotidiana negli ospedali.

Nonostante protocolli sempre più rigorosi di asepsi, in molti reparti chirurgici europei e italiani si registrano ancora tassi di infezione del sito operatorio che variano dal 2 al 5% nei casi di chirurgia elettiva, e arrivano fino al 15-20% negli interventi maggiori e di emergenza.

Lampada UV-A

La lampada UV impiegata per la rilevazione opera a 405 nm. Il suo utilizzo si basa sul fenomeno della “autofluorescenza”. Diversi microrganismi quando esposti a questa radiazione, emettono luce visibile grazie alla presenza di molecole fotosensibili nel loro metabolismo.

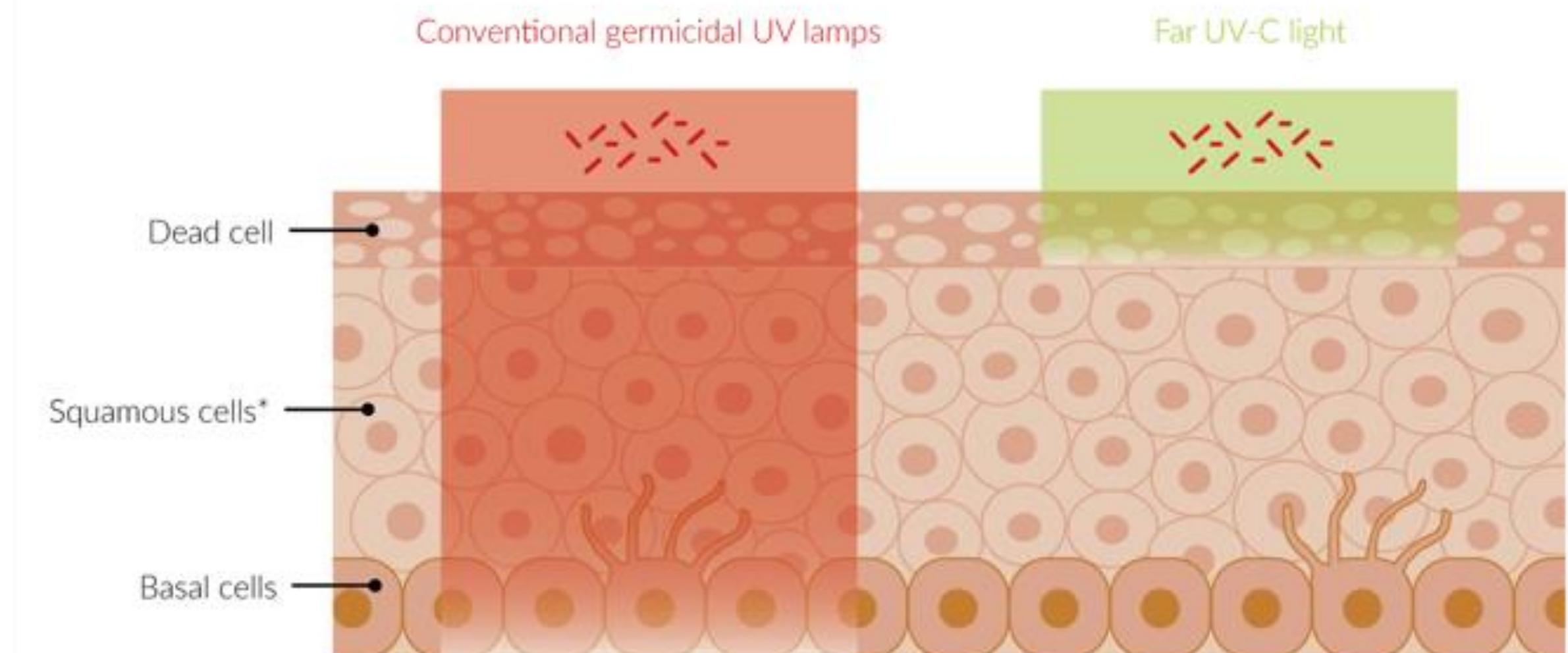
La radiazione a 405 nm non rappresenta un pericolo significativo per la salute umana, poiché rientra nel campo della luce visibile e non possiede energia sufficiente per danneggiare il DNA o provocare effetti mutageni, a differenza delle radiazioni a lunghezza d'onda inferiore (UV-B e UV-C). Tuttavia, un'esposizione diretta e prolungata agli occhi è sconsigliata

Recenti studi scientifici, hanno dimostrato la lunghezza d'onda 222 nm appartiene alla cosiddetta categoria “Far-UVC”, che presenta un'elevata efficacia germicida, mantenendo una capacità di penetrazione nei tessuti biologici estremamente limitata.

Lampada UV-C

Absorption of Far UV-C

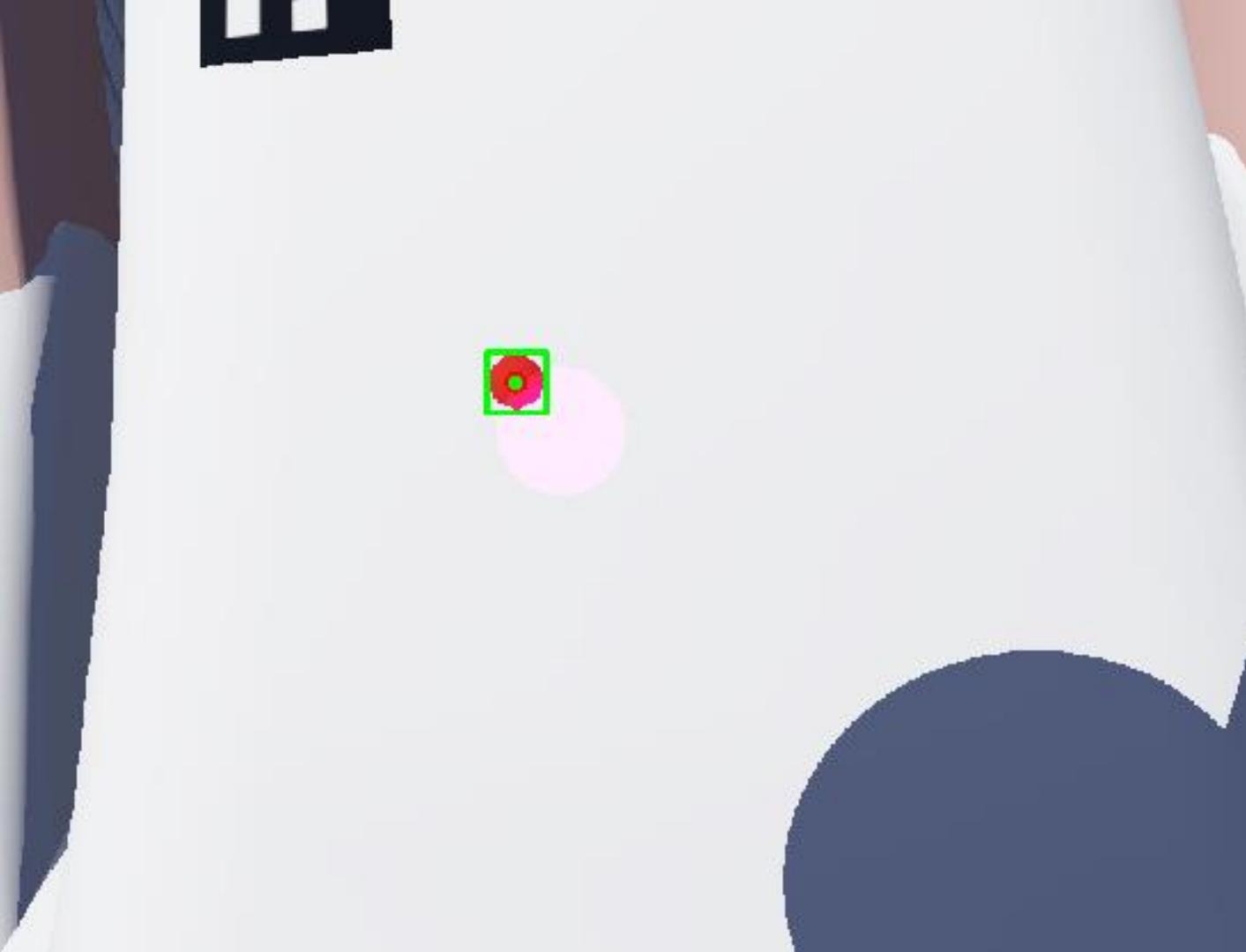
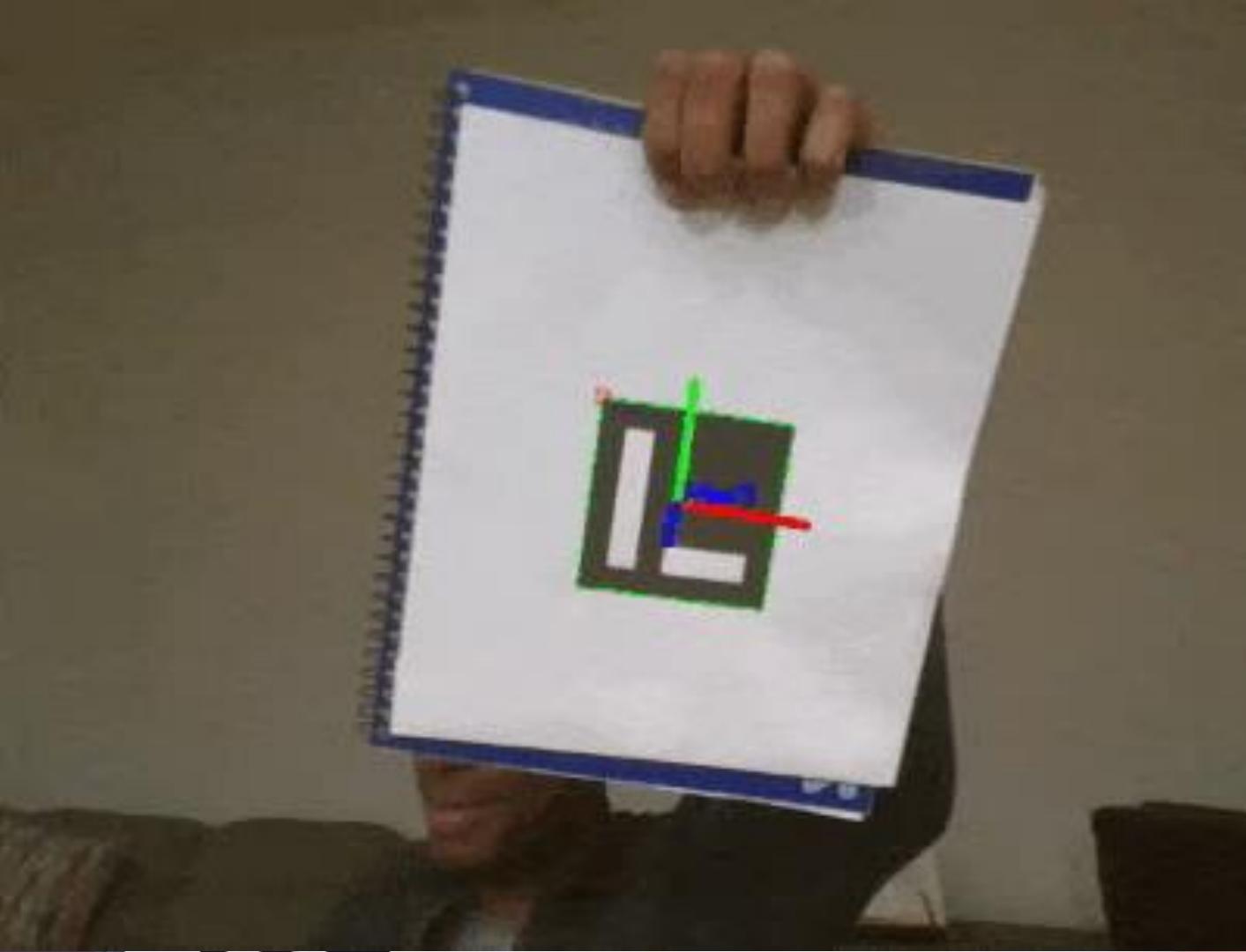
Far UV-C light (222 nm) is absorbed by the top layer of the skin (made of dead cells).



*Squamous cells are the thin flat cells that make up the epidermis or the outermost layer of the skin.

09

Computer Vision Algorithms



OpenCV

(Open Source Computer Vision Library) è una libreria open source di visione artificiale e machine learning, progettata per analizzare immagini e video in tempo reale. È ampiamente utilizzata in robotica, automazione e intelligenza artificiale.

Marker Aruco

OpenCV consente di rilevare e identificare marker ArUco in tempo reale

Rilevamento Batteri

Grazie all'elaborazione delle immagini, OpenCV permette di rilevare i batteri tramite la fluorescenza indotta dalla luce UVA, evidenziandoli come macchie rosse e tracciandone i contorni.

Algoritmo di rilevamento dei marker

```
# Detect the markers in the image  
corners, ids, _ = aruco.detectMarkers(img_bgr, aruco_dict, parameters=detector_params)
```

Preprocessing
dell'immagine

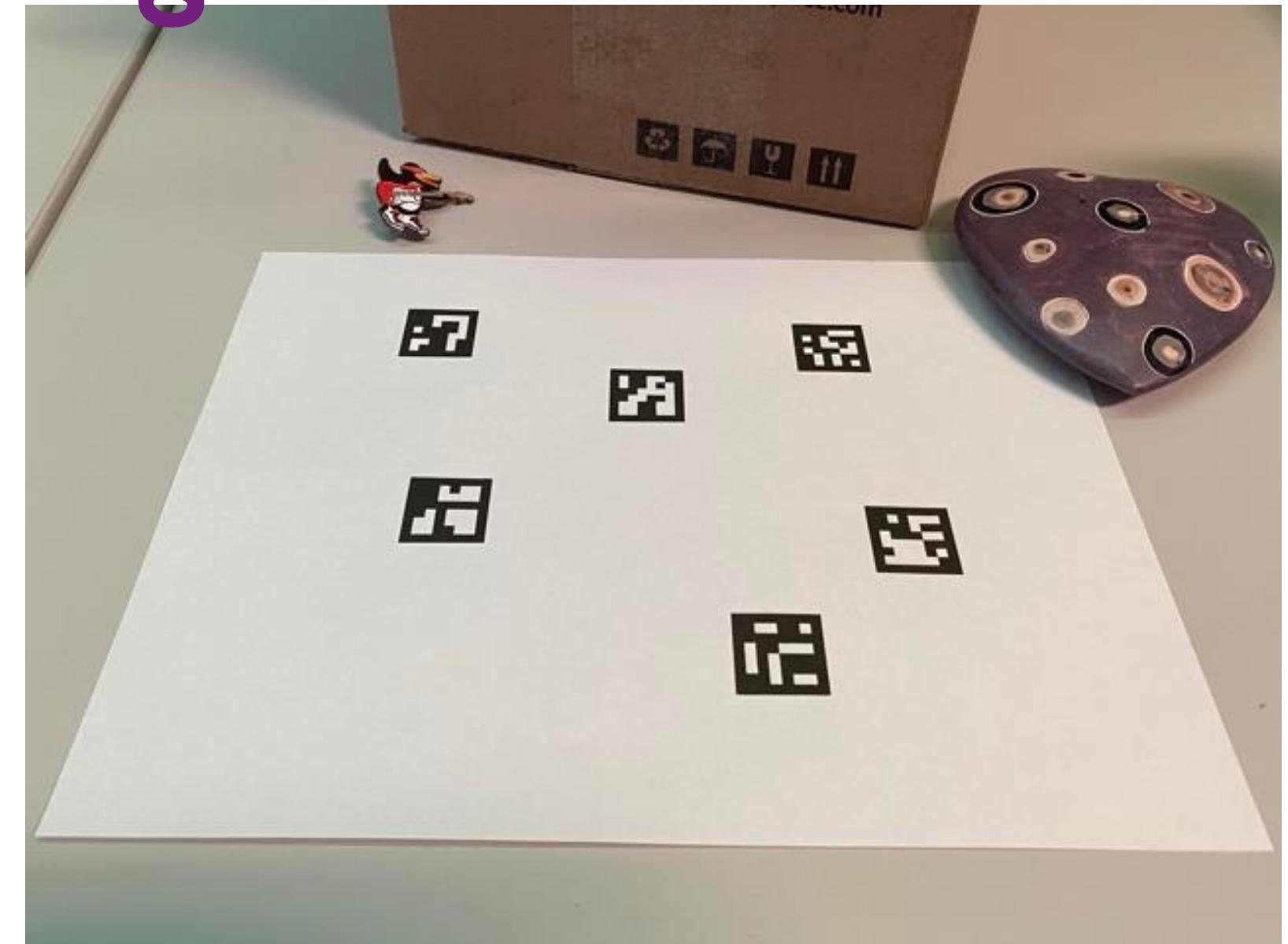
Estrazione del
contenuto

Rilevamento dei
quadrati

Matching con
dizionario

Preprocessing dell'immagine

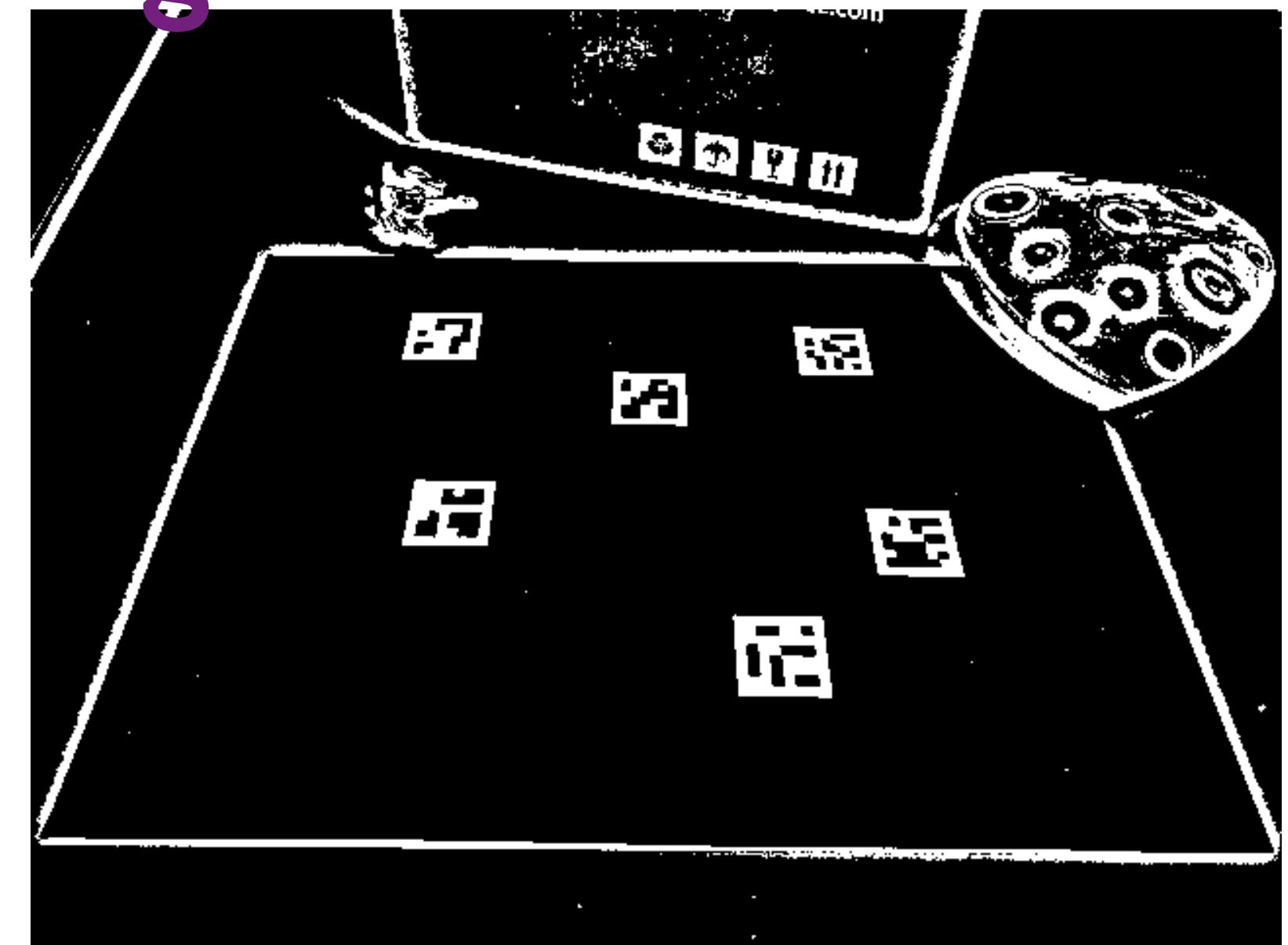
L'immagine viene convertita in bianco e nero per evidenziare i contorni dei marker



Preprocessing dell'immagine

Si effettua una suddivisione in finestre di pixel

Per ogni finestra si calcola una soglia

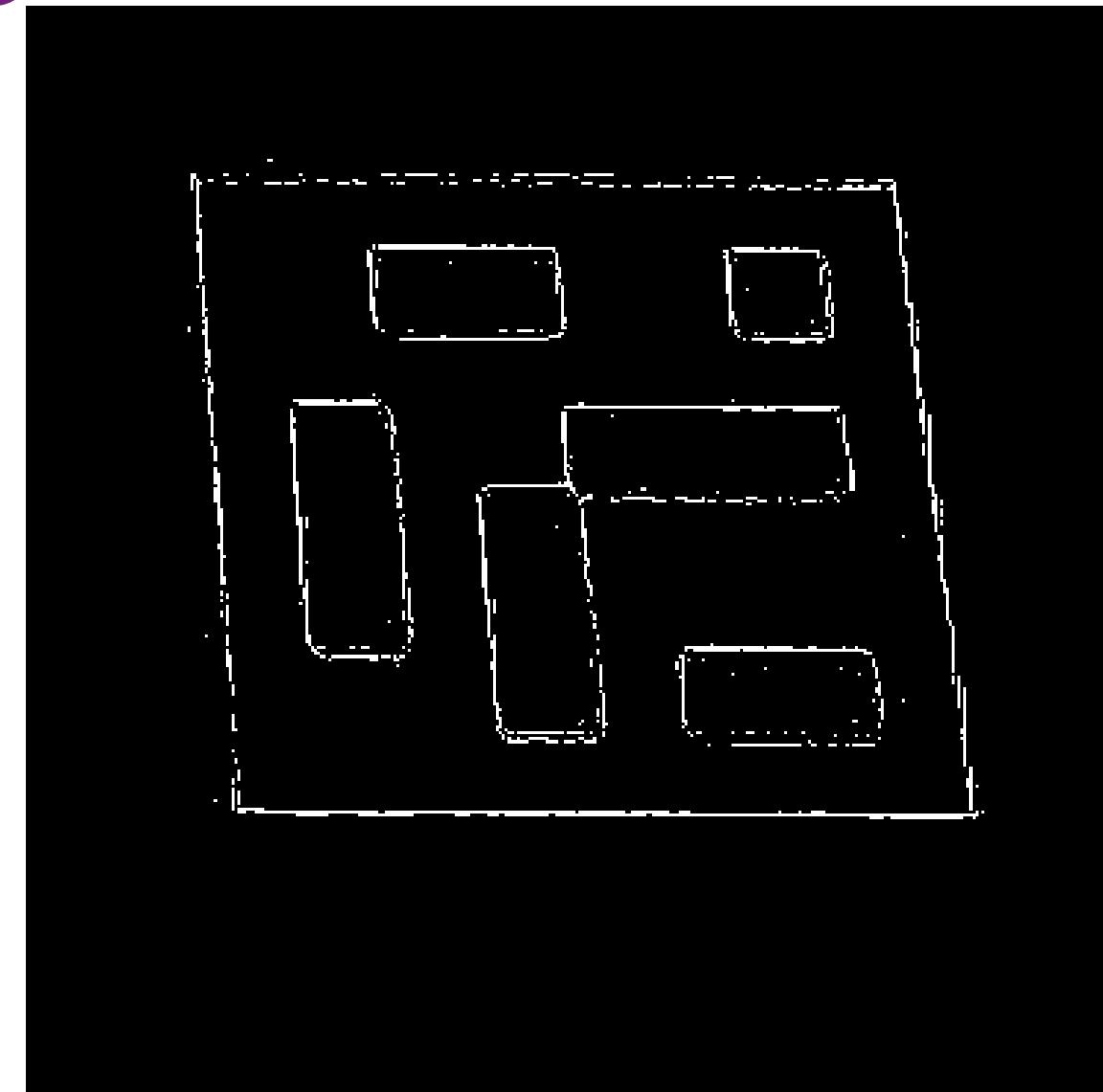


Preprocessing dell'immagine



Attenzione ai parametri

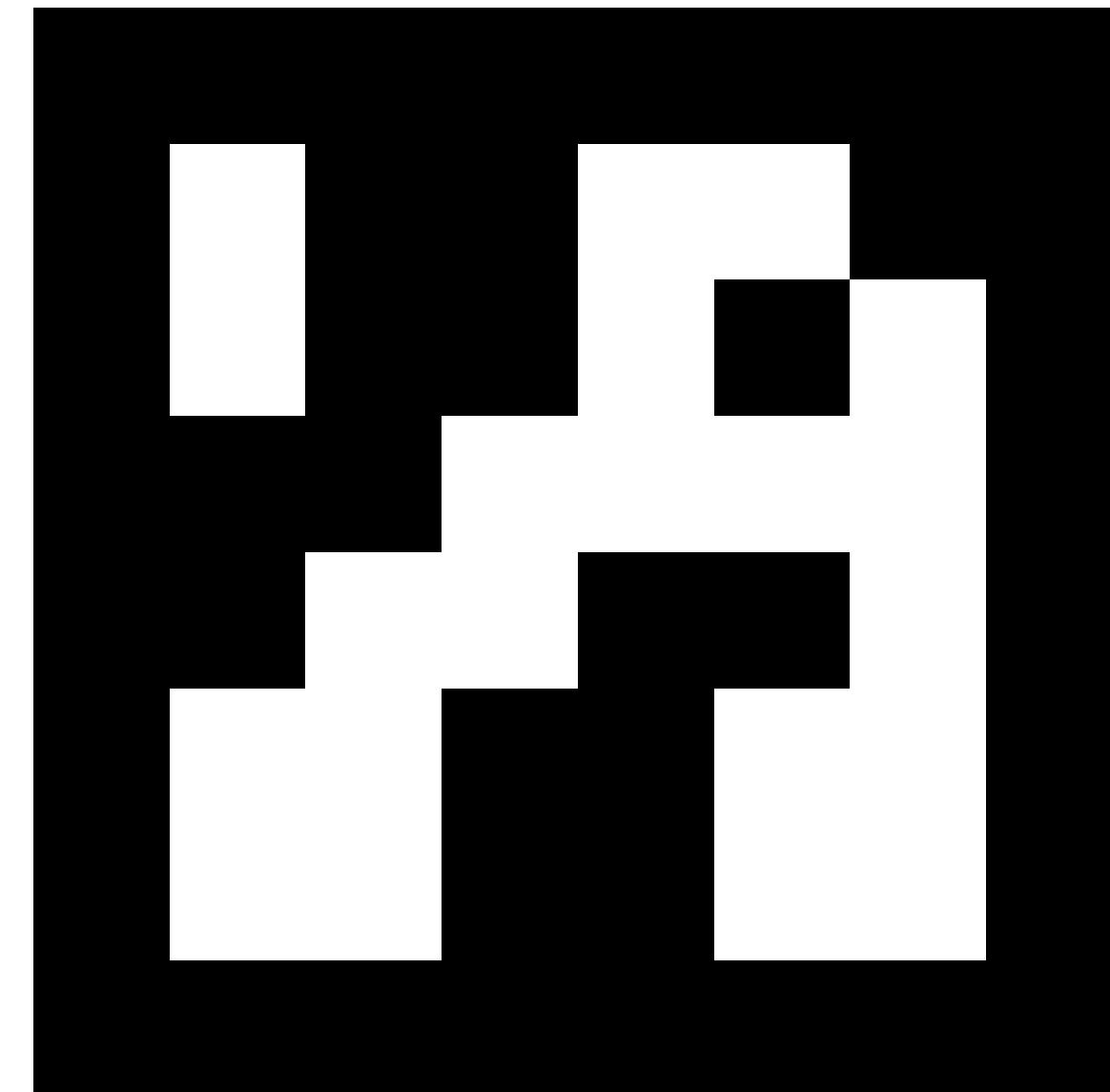
- Finestra troppo piccola → bordi spezzati
- Finestra troppo grande → perdita di dettaglio



Rilevamento dei quadrati

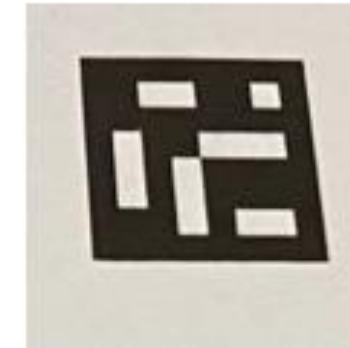
Countour filtering, parametri:

- Dimensioni del contorno
- Forma del contorno
- Distanze fra gli angoli
- Distanza dal bordo dell'immagine



Estrazione del contenuto

Bits extraction



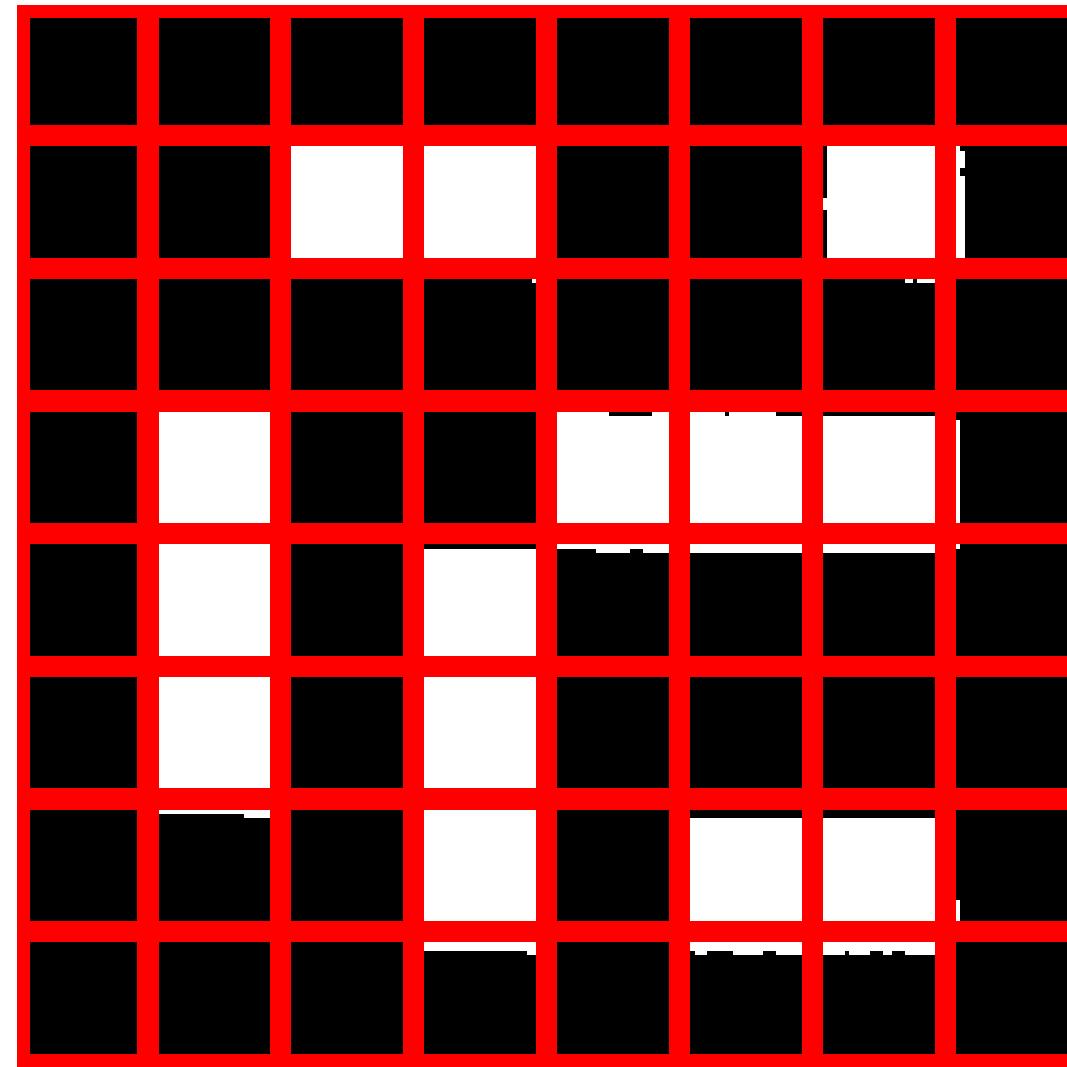
**Si effettua una correzione prospettica per
eliminare la distorsione**



Estrazione del contenuto

Bits extraction

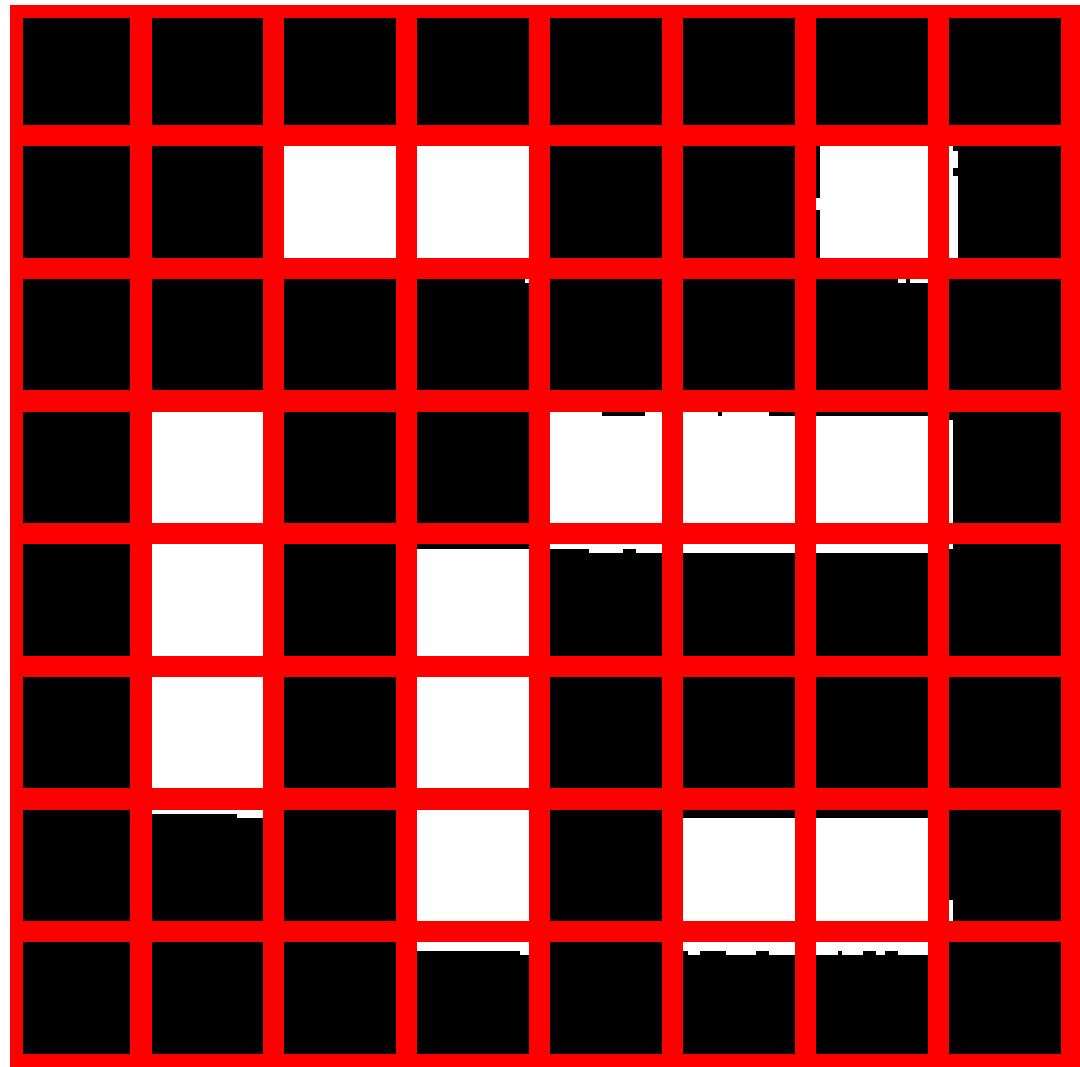
L'immagine viene suddivisa in celle per permettere l'assegnazione del valore di ogni pixel in bit



Matching con il dizionario

Identificazione marker

- Si verifica se il codice binario è presente nel dizionario
- Si possono correggere alcuni bit errati
- Si effettua un ulteriore controllo riguardo i bordi del marker, i quali devono essere neri



Marker ArUco

Vengono posizionati i marker ArUco, per delimitare l'area dell'intervento. Questa tipologia di marker è molto utile in quanto vengono elaborati facilmente da Opencv e possono essere disposti in modi diversi, anche per adattarli bene all'area operativa.



Marker ArUco

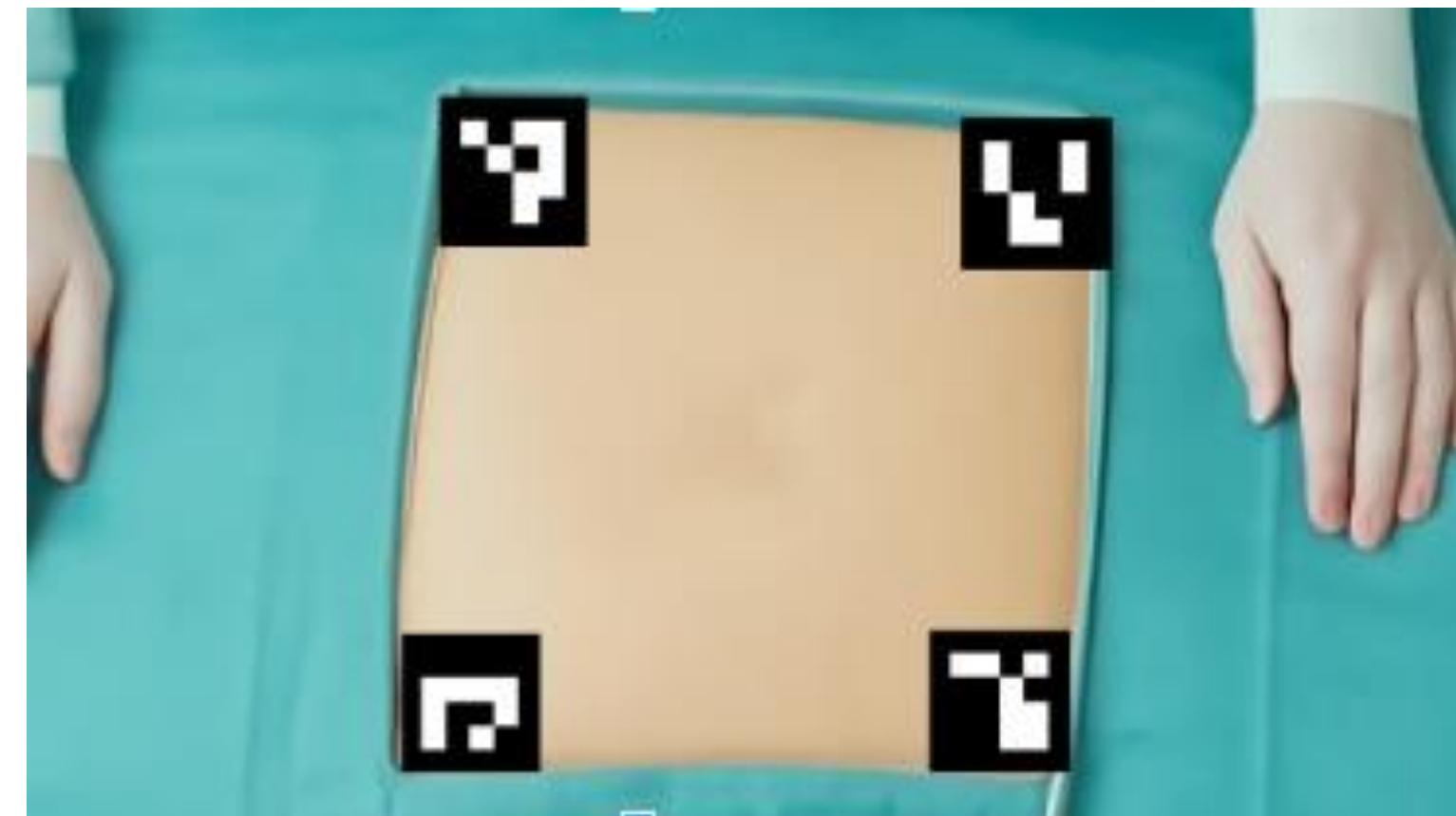
```
def detect_and_draw_markers(frame_raw):

    # Convert the raw image to a numpy array and reshape it
    img_bgra = np.frombuffer(frame_raw, np.uint8).reshape(img_height, img_width, 4)
    img_bgr = cv2.cvtColor(img_bgra, cv2.COLOR_BGRA2BGR)

    # Detect the markers in the image
    corners, ids, _ = aruco.detectMarkers(img_bgr, aruco_dict, parameters=detector_params)

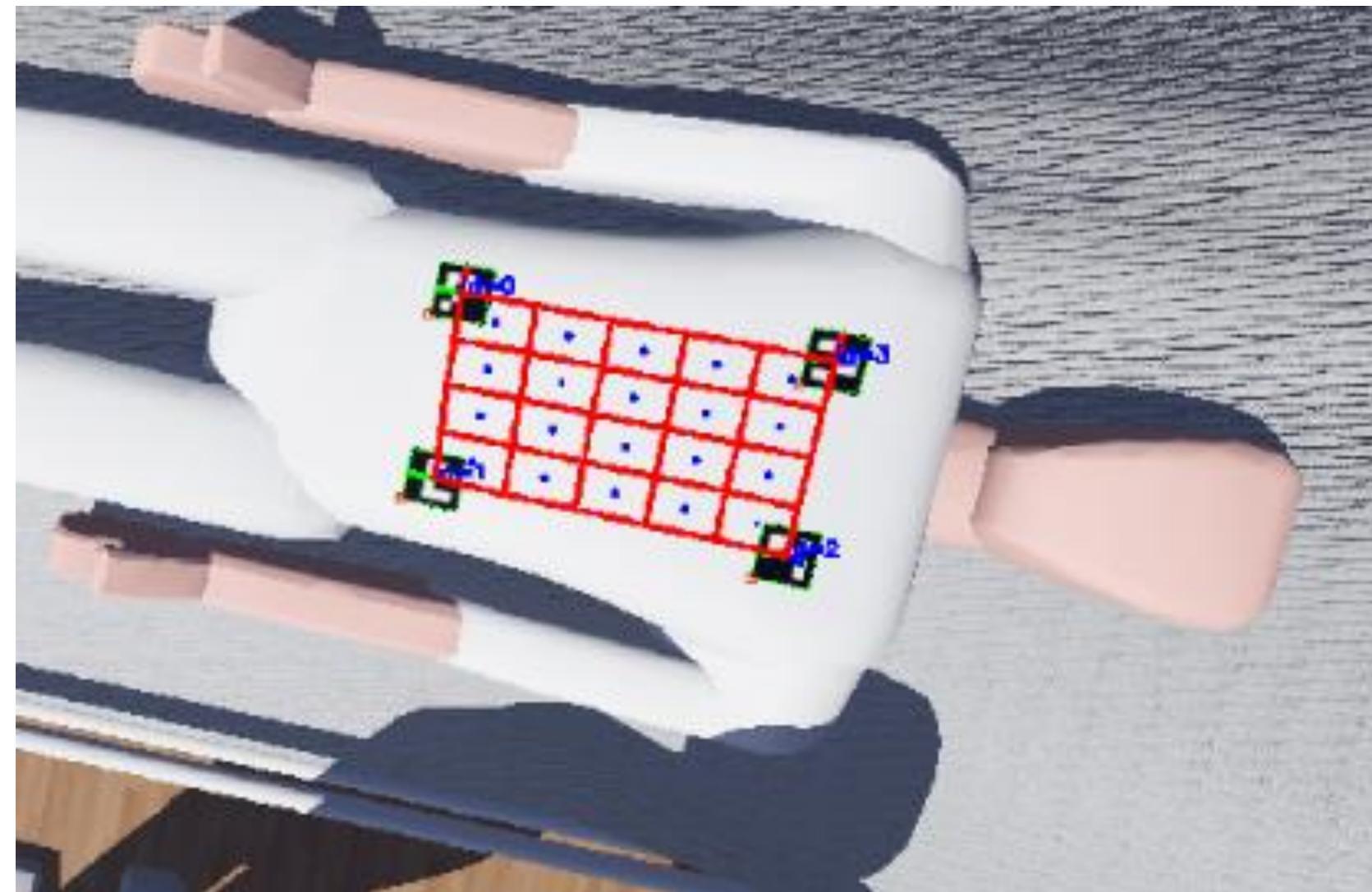
    # If there are detected markers, draw them on the image
    if ids is not None:
        rvecs, tvecs, _ = aruco.estimatePoseSingleMarkers(
            corners, MARKER_SIZE, camera_matrix, dist_coeffs)
        for rv, tv in zip(rvecs, tvecs):
            cv2.drawFrameAxes(img_bgr, camera_matrix, dist_coeffs, rv, tv, MARKER_SIZE/2)
    img_marked = aruco.drawDetectedMarkers(img_bgr.copy(), corners, ids)

    return img_marked, corners, ids
```



Individuazione della griglia

Dopo aver rilevato i marker, vengono riordinati per id. Vengono collegati tutti i marker, viene suddivisa l'area individuata dai marker in due parti, formando due triangoli e all'interno di essi vengono generate dinamicamente le celle stabilendo il numero di celle



Individuazione della griglia

```
# compute_grid_dims():

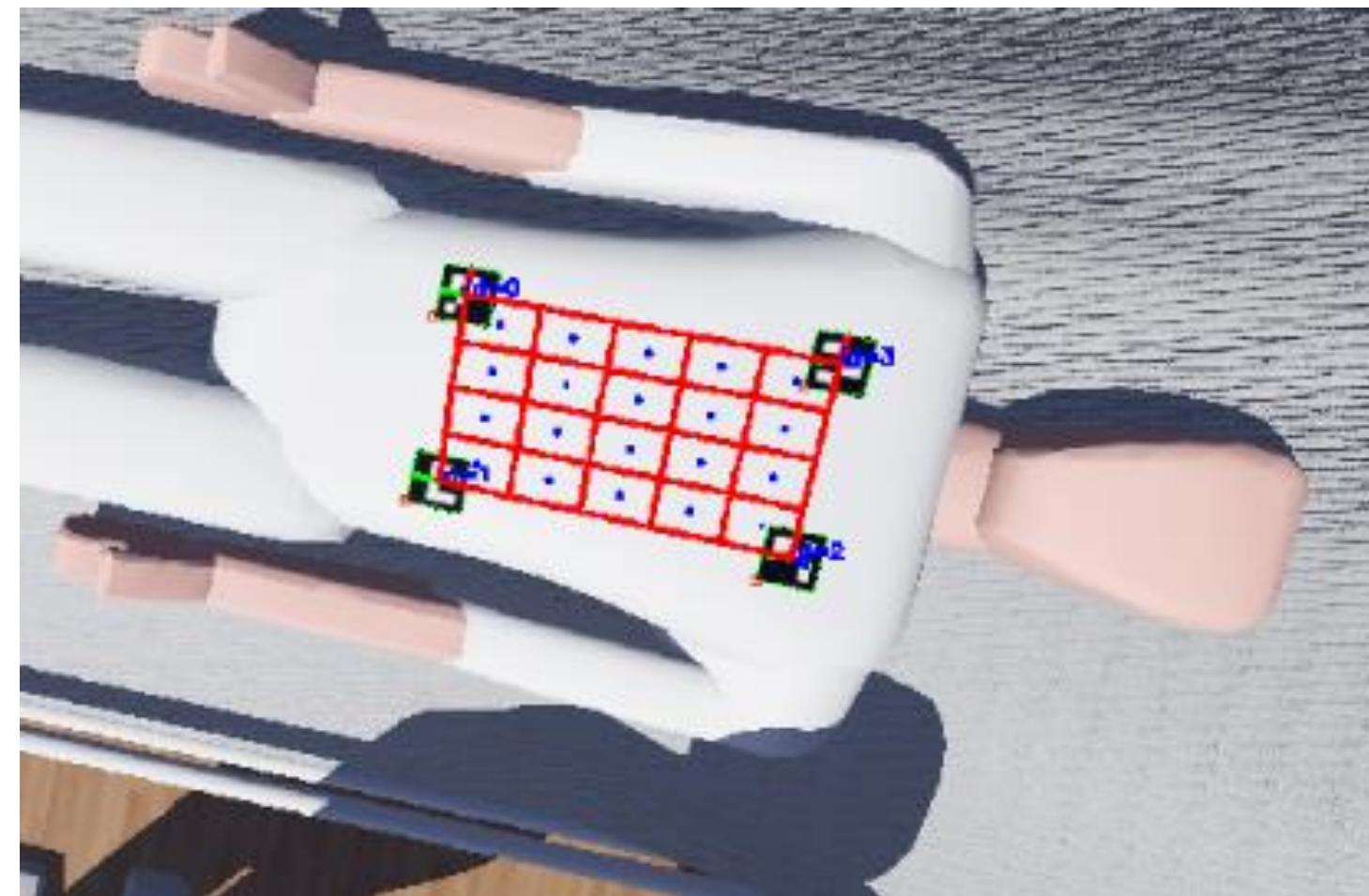
# Compute the area of the quadrilateral formed by the four markers
corners = [world_positions[i] for i in range(4)]

# Calculate the area using the cross product of two vectors
a1 = 0.5 * np.linalg.norm(np.cross(corners[1]-corners[0], corners[2]-corners[0]))
a2 = 0.5 * np.linalg.norm(np.cross(corners[2]-corners[0], corners[3]-corners[0]))
total_area_cm2 = (a1 + a2) * 1e4

# Calculate the number of cells based on the target cell area
num_cells = max(1, int(round(total_area_cm2 / CELL_AREA_TARGET)))

# Calculate the number of rows and columns for the grid
rows = int(np.floor(np.sqrt(num_cells)))
cols = int(np.ceil(num_cells / rows))

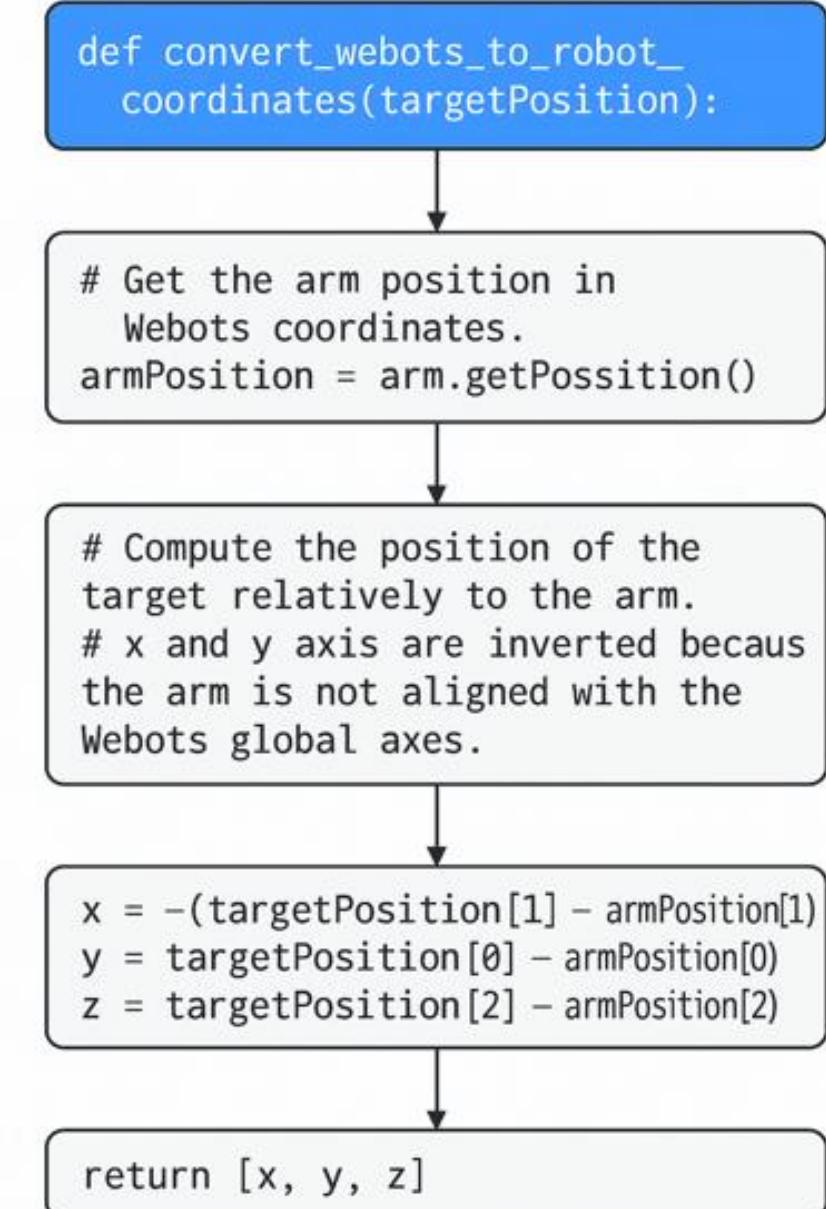
return rows, cols
```



Calcolo traiettoria e Object Detection

Matrice di conversione

Per poter proseguire nella simulazione, è necessario convertire le coordinate webots nelle coordinate del robot. Il sistema di riferimento di webots risulta invertito rispetto al sistema di riferimento del robot, quindi per una corretta pianificazione e spostamento è necessario convertirle nel sistema di riferimento corretto.



Interpolazion e

```
def interpolate_cam_centers(centers2d, rows, cols):  
  
    centers3d = {}  
  
    for (i, j), _ in centers2d.items():  
        u, v = (j + 0.5) / cols, (i + 0.5) / rows  
        top = cam_positions[0] * (1-u) + cam_positions[1] * u  
        bot = cam_positions[3] * (1-u) + cam_positions[2] * u  
        centers3d[(i,j)] = top * (1-v) + bot * v  
  
    return centers3d
```

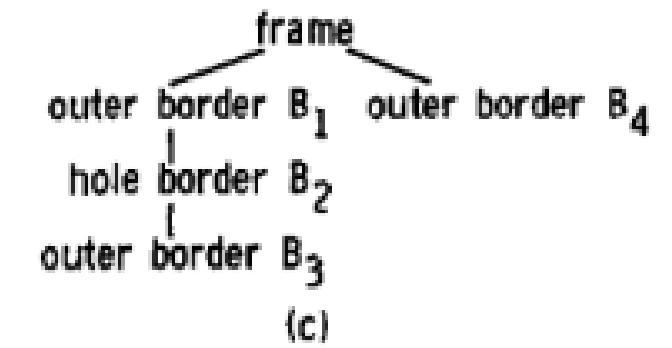
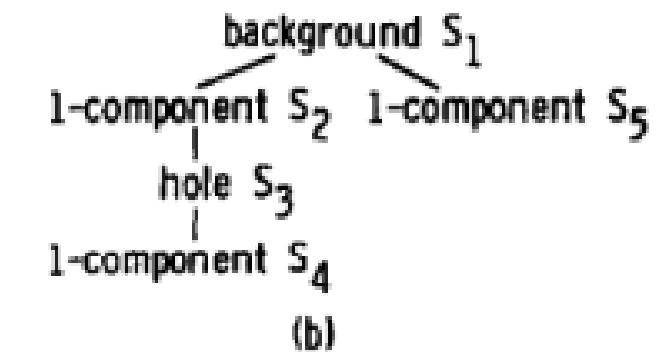
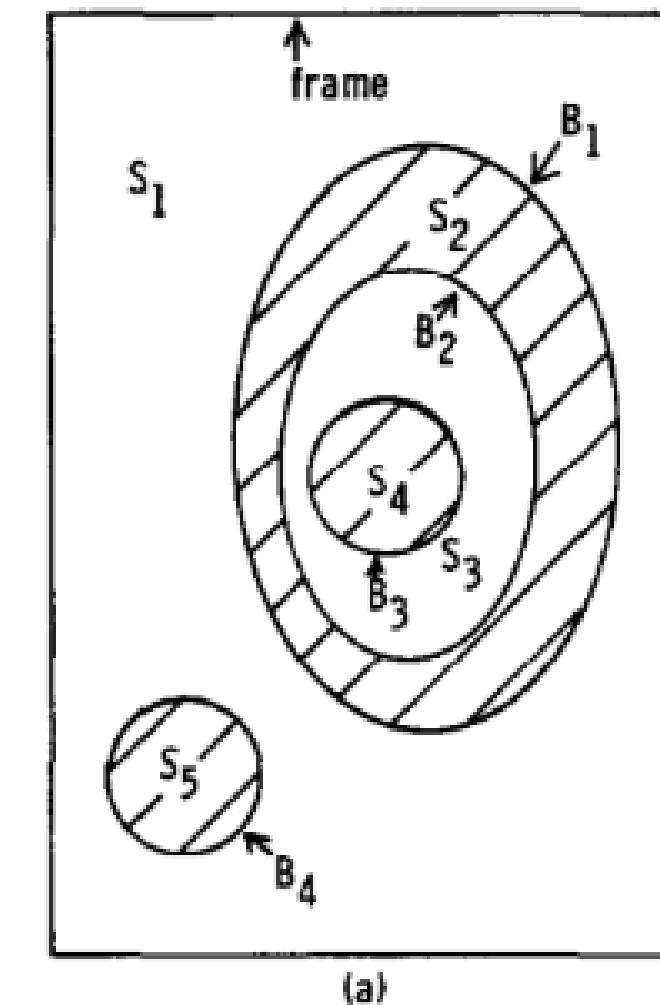
$$\text{top} = (1 - u) \cdot \text{cam0} + u \cdot \text{cam1}$$

$$\text{bot} = (1 - u) \cdot \text{cam3} + u \cdot \text{cam2}$$

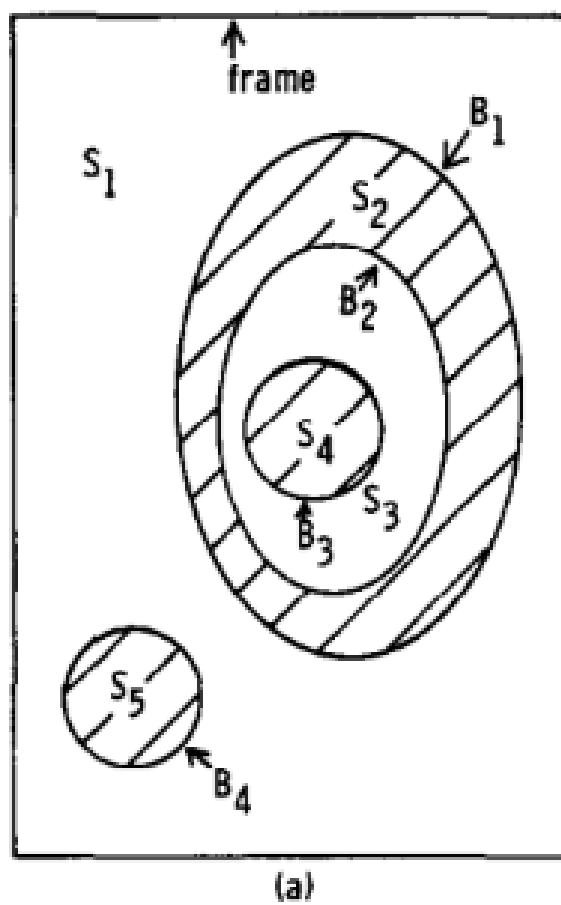
$$\text{center} = (1 - v) \cdot \text{top} + v \cdot \text{bot}$$

Algoritmo FindContours

Algoritmo inventato da Suzuki e Abe nel 1985, molto utilizzato per rilevare gli oggetti all'interno di un'immagine. Vengono utilizzate immagini binarie rappresentate come una matrice bidimensionale di pixel, definite attraverso coordinate(i,j) con differenziazione tra componenti connesse 0-pixel, 1-pixel.



Algoritmo FindContours



background S_1

1-component S_2 1-component S_5

hole S_3

1-component S_4

(b)

frame

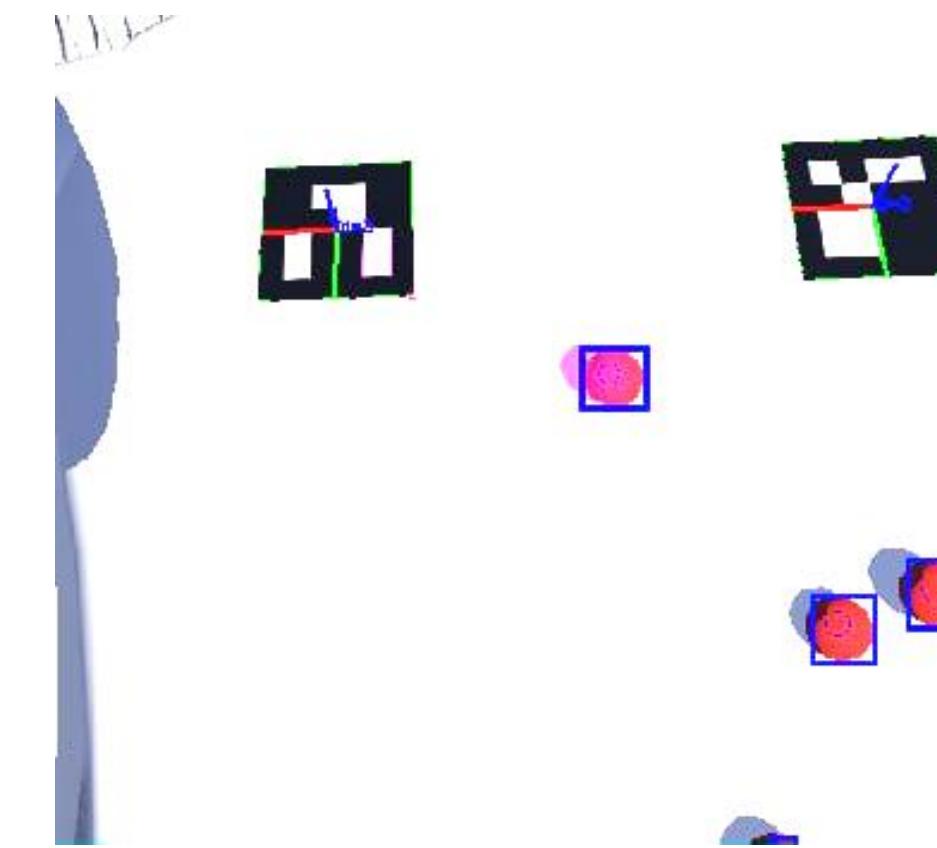
outer border B_1 outer border B_4

hole border B_2

outer border B_3

(c)

Diagram (b) shows a hierarchical tree structure for the regions and borders from diagram (a). The root node is the background S_1 , which branches into two 1-component regions, S_2 and S_5 . The region S_2 contains a hole S_3 , and the region S_5 contains a 1-component region S_4 . Diagram (c) shows a hierarchical tree structure for the borders from diagram (a). The root node is the frame, which branches into four outer borders: B_1 and B_4 , and two hole borders: B_2 and B_3 .



Algoritmo FindContours

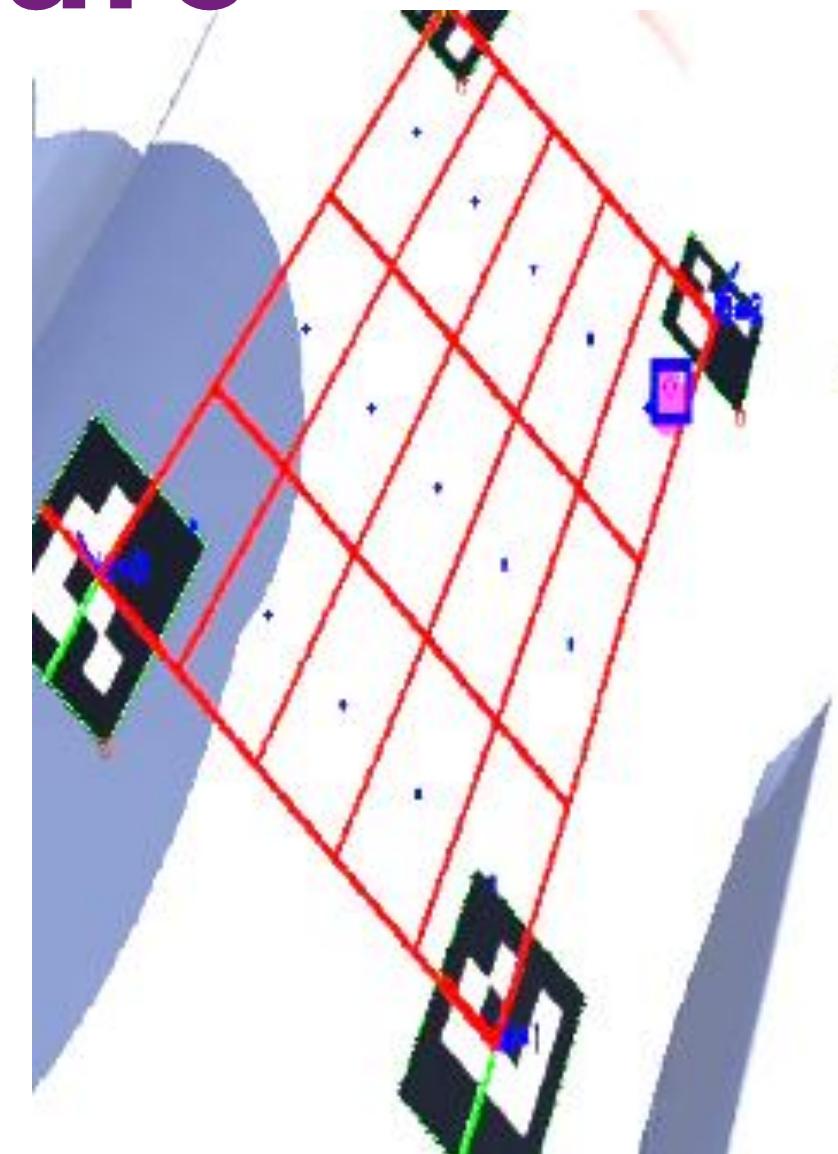
```
# Convert the raw image buffer to a numpy image
img_bgra = np.frombuffer(raw_image_buffer, np.uint8).reshape(img_height, img_width, 4)
img_bgr = cv2.cvtColor(img_bgra, cv2.COLOR_BGRA2BGR)

# Detect red bacteria using color thresholding
hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
lower_red1 = np.array([0, 100, 100])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([160, 100, 100])
upper_red2 = np.array([179, 255, 255])
mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
mask = cv2.bitwise_or(mask1, mask2)
kernel = np.ones((5, 5), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

# Find contours in the mask by using OpenCV
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw contours on the original image
for cnt in contours:
    x, y, w, h = cv2.boundingRect(cnt)

    # Draw a rectangle around the contour
    for thickness in range(1, 4):
        cv2.rectangle(img_bgr,
                     (max(0, x - thickness), max(0, y - thickness)),
                     (min(img_width, x + w + thickness), min(img_height, y + h + thickness)),
                     (0, 255, 0), # Verde brillante in formato BGR
                     2)
```



Traiettori

a

La libreria robotics toolbox fornisce un insieme di strumenti per il calcolo della traiettoria. Questa viene calcolata attraverso un polinomio di quinto grado che assicura continuità in posizione, velocità e accelerazione

$$\underline{Q} = \mathbf{M}(\underline{q})\ddot{\underline{q}} + \mathbf{C}(\underline{q}, \dot{\underline{q}})\dot{\underline{q}} + \mathbf{F}(\dot{\underline{q}}) + \mathbf{G}(\underline{q})$$

$$q(0) = q_0$$

$$\dot{q}(0) = 0$$

$$q(t_f) = q_f$$

$$\dot{q}(t_f) = 0$$

Traiettori

a

- Ingresso: configurazione iniziale q_0 , configurazione finale q_f , e numero di punti n (o un vettore di tempo).
- Uscita: traiettoria q (posizioni), qd (velocità), qdd (accelerazioni).

```
# Set the active trajectory to move to the target position
active_trajectory = jtraj(initial_joint_positions, target_joint_positions, 10).q
trajectory_step_index = 0

# Set the target for check_if_moved to the target position
g_target_for_check_if_moved = target_position_world

# Change the state to MOVING_TO_DETECT_SPOT
current_state = MOVING_TO_DETECT_SPOT
```

Sviluppi futuri

Possibili miglioramenti

Riduzione del costo
di implementazione

Hardware più
potente
API, estensioni per le
funzionalità del
robot

Deep
Learning/Machine
learning

Bibliografia

- UV medico far UVC (<https://uvmedico.com/far-uvc-light>)
- The effects of 405 nm light on bacterial membrane integrity determined by salt and bile tolerance assays, leakage of UV-absorbing material and SYTOX green labelling (<https://pmc.ncbi.nlm.nih.gov/articles/PMC5068139/>)
- Application of Fluorescence Spectroscopy for Microbial Detection to Enhance Clinical Investigations (<https://www.intechopen.com/chapters/59541>)
- Paper FindContours Algorithm (<https://d1wqtxts1xzle7.cloudfront.net/38698235/suzuki1985-libre.pdf>)
- Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods (<https://www.cs.cmu.edu/~15464-s13/lectures/lecture6/iksurvey.pdf>)
- A computer tool for simulation and analysis: the Robotics Toolbox for MATLAB (<https://www.petercorke.com/RTB/ARA95.pdf>)