Università degli Studi di Palermo

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

Cybersterilizer

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

A cura di:

Ambrogi Federico Ennio
Burgio Gabriele
Masi Luca
Sollazzo Ivan

Docente

Prof. Antonio Chella

Introduzione

Cybersterilizer nasce da un'idea del team composto da **Ambrogi Federico Ennio**, **Burgio Gabriele**, **Masi Luca** e **Sollazzo Ivan**, sulla base di un caso di studio al fine di rilevare ed eliminare la carica batterica da una ferita o un'area delimitata utilizzando la tecnologia a doppia luce UV. Il sistema robotico si presenta non come un completo sostituto delle pratiche standard, ma come un rafforzamento intelligente delle procedure di sterilizzazione, puntando su due elementi chiave:

- **Ripetibilità del processo:** il robot esegue ogni ciclo di sterilizzazione seguendo sequenze programmate e ottimizzate, prive delle 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.

Il robot si avvale di un sistema di visione artificiale basato sui marcatori **ArUco** per identificare con precisione l'area da delimitare e suddividere in una **griglia virtuale**, la quale permette al robot di analizzare ogni singola cella in maniera sistemistica ed applicare i protocolli di rilevamento dei batteri, tramite la luce **UV-A (405 nm)**, e di sterilizzazione degli stessi tramite i raggi **UV-C (222 nm**).

Architettura del sistema

L'architettura del sistema Cybersterilizer è costituita sostanzialmente da due principali sottosistemi:

- **Robot base:** la struttura che sostiene l'intera piattaforma;
- **End effector:** costruito ad hoc per gli scopi finali del progetto.

Il controllore, simulato tramite l'ambiente **Webots**, include il software di intelligenza artificiale per il coordinamento degli algoritmi di computer vision e dell'hardware, in modo tale da gestire il comparto cinematico, di pianificazione delle traiettorie e dell'operazione di sterilizzazione.

Specifiche tecniche del robot

Per la realizzazione di questo progetto è stato scelto il manipolatore **IRB 4600/40** prodotto da **ABB**.

Figura 1: ABB IRB 4600/40

Si tratta di un braccio antropomorfo a **6 gradi di libertà** (DOF, degrees of freedom). Appare subito evidente che questa tipologia di manipolatori richiami la struttura del braccio umano, tant'è che è possibile trovare una perfetta analogia con riferimento a:

- I primi tre giunti per la spalla, il braccio e l'avambraccio;
- Gli ultimi tre giunti per il polso.

Esattamente come il braccio umano, il robot può compiere rotazioni congiunte che permettono di raggiungere tutti i punti appartenenti al suo spazio di lavoro. Infatti, tutti i giunti sono di tipo **rotoidale**.

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

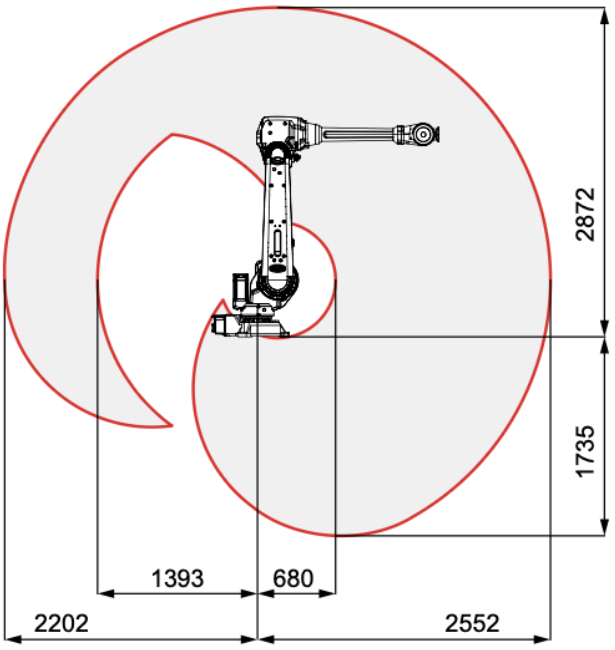

Figura 2: Rappresentazione dell'area raggiungibile.

Nel dettaglio:

- **2552 mm** e **2202 mm** indicano i raggi massimi a destra e sinistra rispetto al centro base;
- **2872 mm** è l'altezza massima raggiungibile dal braccio;
- **1735 mm** è l'altezza rispetto al piano base del robot;
- **680 mm** e **1393 mm** indicano i raggi interni dell'area non raggiungibile.

Per un elenco completo delle specifiche tecniche è possibile consultare il datasheet del produttore.

Si precisa che la scelta della struttura in questione è di carattere sperimentale. Ai fini della commercializzazione e dell'utilizzo finale possono essere impiegate tranquillamente altre strutture antropomorfe a sei giunti, utilizzando il medesimo end effector.

Tabella di Denavit-Hartenberg

Per l'implementazione del comparto cinematico ci si avvale di una convenzione che prende il nome di **convenzione di Denavit-Hartenberg**. Tramite essa è possibile assegnare a ciascun link della struttura una **terna di coordinate**. Ciascun parametro descrive la relazione spaziale (posizione e orientamento) tra la terna di un link i e quella del link precedente $i - 1$. Poiché la convenzione è standardizzata a tutte le tipologie di giunti (rotoidali e prismatici), si definiscono i seguenti parametri geometrici:

- a_i è la distanza tra le origini di due terne lungo l'asse di x_i;
- α_i: rappresenta l'**angolo** tra z_{i-1} e z_i attorno a x_i;
- d_i: rappresenta la **distanza** tra le origini di due terne lungo z_{i-1}. Variabile di giunto se il giunto è prismatico.
- θ_i: rappresenta l'**angolo del giunto** (joint angle). Variabile di giunto se il giunto è rotoidale.

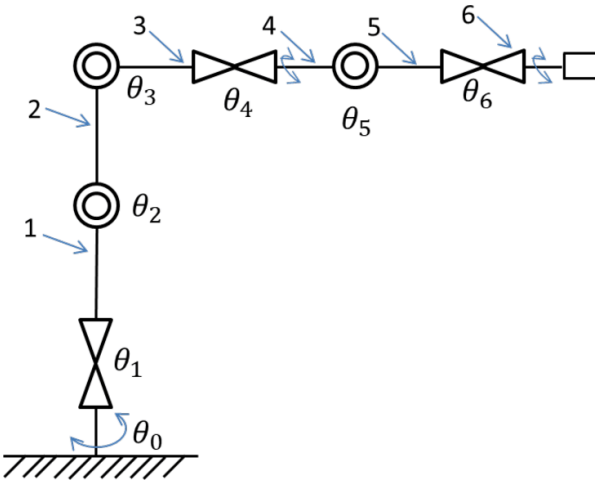

Figura 1: Applicazione della convenzione D-H all'ABB IRB 4600. Determinazione delle variabili di giunto.

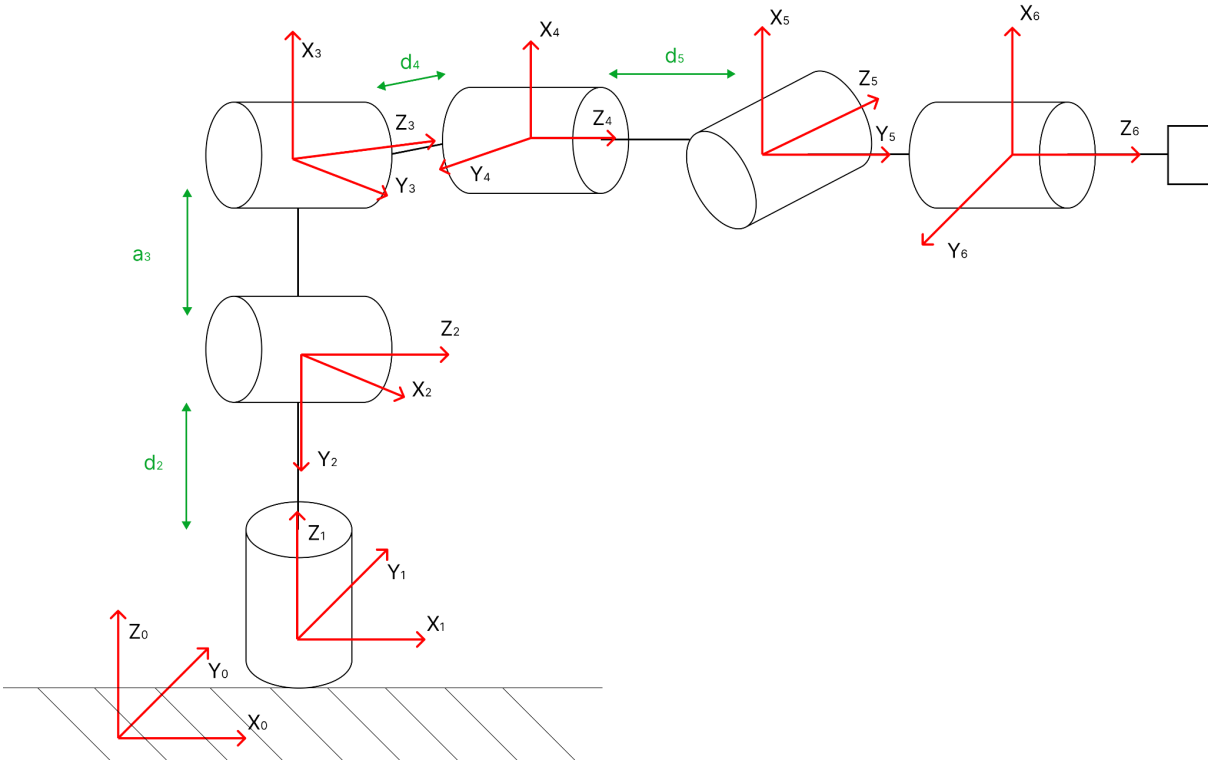

Figura 2: Applicazione della convenzione D-H all'ABB IRB 4600. Determinazione delle terne.

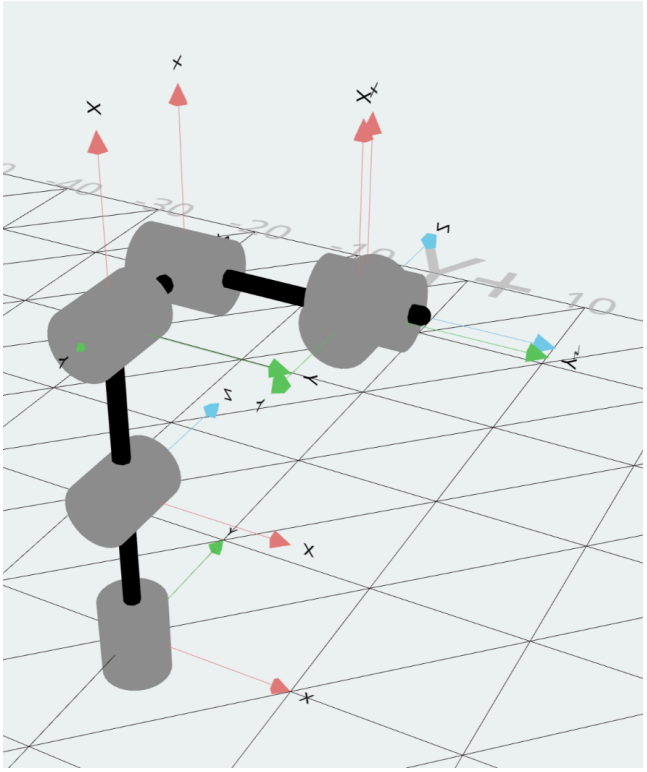

Figura 3: Applicazione della convenzione D-H all'ABB IRB 4600. Modello 3D della convenzione realizzato tramite il software disponibile al link: https://robotics101-viscircuit.web.app/robotics/dh-model

Applicando questa convenzione al manipolatore scelto individuiamo i parametri di D-H per i giunti:

Joint	θi	α_i (gradi)	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

Tabella 1: Tabella di Denavit-Hartenberg per l'IRB 4600/40.

La relazione spaziale di ciascun giunto rispetto al precedente è descritta dalla seguente matrice di trasformazione:

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

Risoluzione della cinematica diretta

La **risoluzione della cinematica diretta** consiste nel problema di determinare quali sono le coordinate dell'end effector nello spazio di lavoro, con riferimento alla terna di basamento. Con riferimento alla precedente tabella di Denavit-Hartenberg, nonché alla matrice di rotazione, è necessario costruire le matrici di trasformazione tra i giunti $i-1$ e i, per poi moltiplicarle al fine di ottenere la matrice di trasformazione che descrive posizione e orientamento dell'end effector rispetto alla terna di basamento:

$$M_6{}^0 = M_1{}^0 M_2{}^1 M_3{}^2 M_4{}^3 M_5{}^4 M_6{}^5$$

Eseguendo i prodotti matriciali, si ottiene l'opportuna matrice di trasformazione.

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

Al fine di poter compattare la matrice, si definiscono le seguenti uguaglianze:

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

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

$$\sigma_3 = c_3c_4 - s_3s_4$$

$$\sigma_4 = c_3s_4 + c_4s_3$$

N.B. I calcoli sono stati eseguiti tramite il software **MATLAB**. Inoltre, le definizioni s_{ij} e c_{ij} indicano rispettivamente $\sin(\theta_i + \theta_j)$ e $\cos(\theta_i + \theta_j)$.

Risoluzione della cinematica inversa

Il problema della **risoluzione della cinematica inversa** consiste nel determinare gli angoli dei giunti, note le coordinate dell'end effector rispetto al basamento. Per un robot a sei giunti, tale problema non è risolvibile con lo stesso grado di difficoltà della cinematica diretta, motivo per cui esistono dei metodi di ottimizzazione analitici e geometrici utili alla causa.

Il metodo approfondito per questo progetto, anche in funzione delle librerie utilizzate, consiste nel trovare quel **vettore delle configurazioni** $\vec{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$ tale che:

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

In particolare:

- $f(\vec{q})$ è la **funzione di cinematica diretta** che calcola la trasformazione finale, dato il vettore delle configurazioni;
- T_{target} è la **matrice di trasformazione omogenea di posizione e orientamento** target dell'end effector.

In generale, definiamo l'errore $\vec{e}$ come un vettore a 6 dimensioni che contempla gli errori di posizione e di orientamento dell'end effector. L'obiettivo è minimizzare tale errore con riferimento ad una **funzione di costo** come l'errore quadratico:

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

A tale scopo, si utilizza un algoritmo iterativo che si avvale della pseudoinversa dello **jacobiano analitico**.

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

Dove i vari m_{ij} sono tutte le componenti della matrice di trasformazione senza tenere conto del vettore aumentato. In particolare, ad ogni passo k dell'algoritmo si trova la seguente variazione del vettore delle configurazioni:

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

Si aggiorna il vettore delle configurazioni 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 determinata soglia.

La libreria ikpy

ikpy è una libreria Python progettata per la modellazione e la risoluzione della cinematica di robot articolati tramite catene cinematiche. È particolarmente adatta a robot con bracci manipolatori e consente di risolvere sia la **cinematica diretta**, che la **cinematica inversa**, per mezzo dei metodi precedentemente discussi.

Tramite ikpy è possibile costruire un oggetto di tipo **Chain** (ovvero la catena cinematica) per mezzo di un file URDF, contenente già i parametri necessari, oppure manualmente prendendo i dati dalla tabella di Denavit-Hartenberg.

A titolo di esempio, nel caso di Cybersterilizer è stato ricavato il file URDF direttamente dalla simulazione Webots e poi istanziata la catena cinematica col metodo `.from_urdf_file`.

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

Possiamo notare inoltre la presenza di un parametro active_links_mask. In particolare, questo parametro è un vettore che definisce come `True` quei link che contribuiscono alla risoluzione della cinematica, mentre come `False` quelli fissi. Ad esempio, il link di basamento è `False` poiché non deve ruotare. Gli ultimi tre `False` sono legati alla simulazione, poiché sono dei link di offset non realmente visibili, ma necessari per definire una distanza arbitraria tra il paziente e l'end effector.

Il metodo `forward_kinematics` restituisce la **matrice di trasformazione** 4×4 di posizione e orientamento dell'effettore, con riferimento alla terna di basamento.

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

Per quanto riguarda invece la cinematica inversa, si utilizza il metodo `inverse_kinematics`, il quale accetta le coordinate nello spazio di lavoro e dei parametri come il numero di iterazioni massime, la posizione iniziale dei giunti, nonché la configurazione dell'orientamento dell'end effector e quali assi sono contemplati nell'orientamento.

```
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_posi
tions)
```

Controllore

Il **controllore** del robot è costituito da una macchina a stati, in cui ciascuno stato rappresenta un compito isolato che il robot è in grado di svolgere.

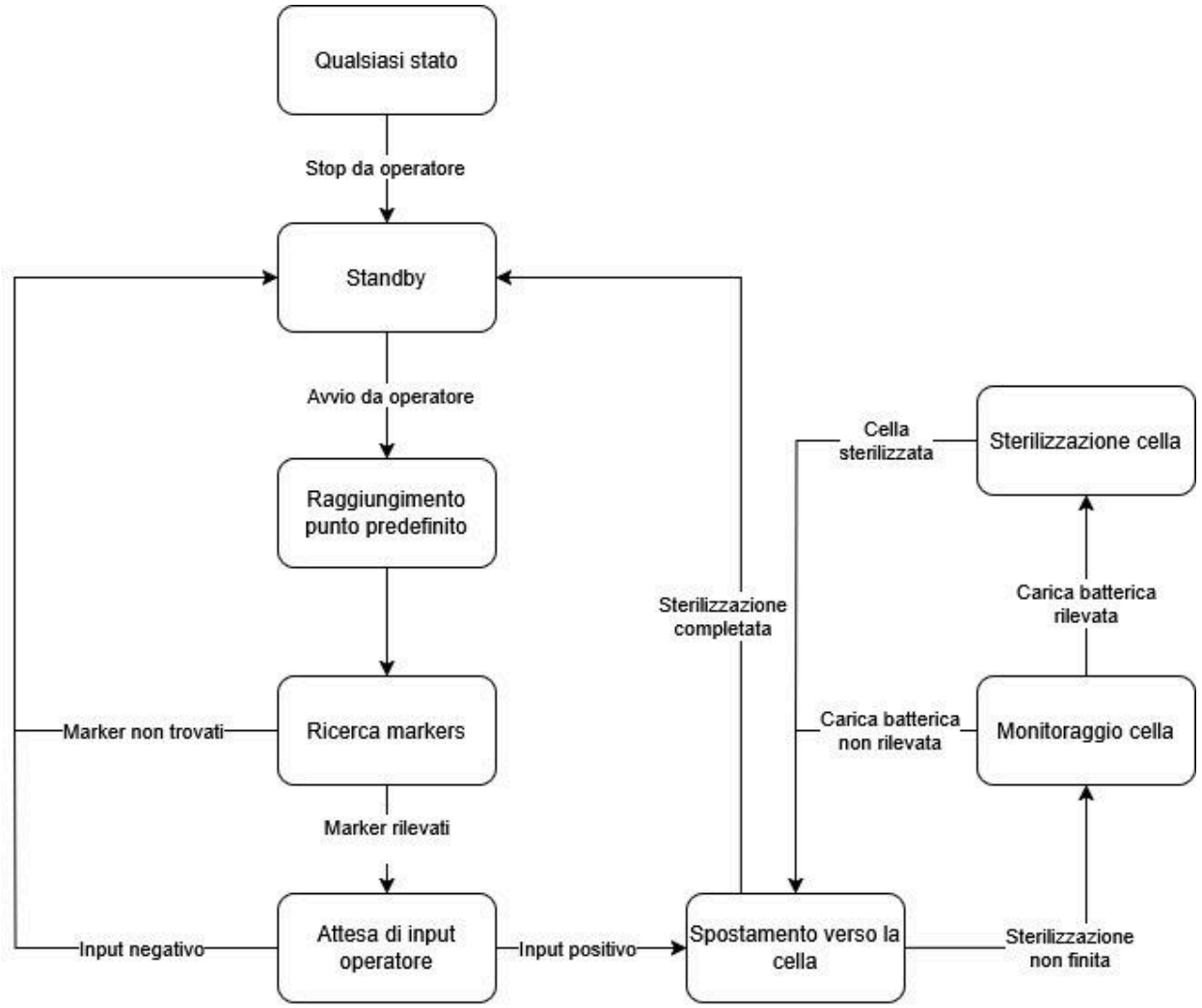

Figura 1: macchina a stati del robot.

Fase di standby

Si assume che durante questa fase il robot pronto per ricevere l'input di avvio da parte dell'operatore. Prima di avviare la sterilizzazione, è necessario che vengano posizionati sul paziente quattro marcatori **ArUco**, in maniera tale da delimitare l'area operatoria.

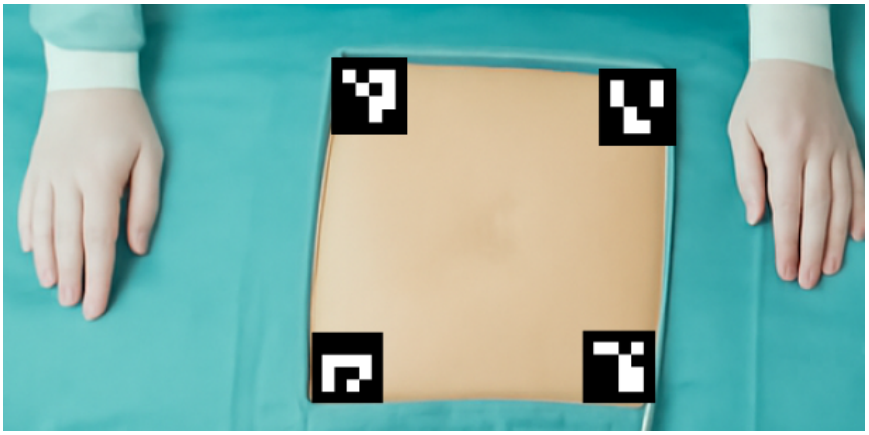

Figura 2: Esempio area operatoria con marker ArUco

Una volta preparata l'area operatoria bisogna premere quindi il pulsante di avvio (**S** sulla tastiera nell'ambiente simulativo Webots). In un ambiente reale si potrebbe implementare l'avvio grazie ad un telecomando o qualsiasi altro dispositivo di input.

Posizionamento e ricerca area sterile

In questo stato, il robot si colloca in una posizione ottimale per individuare l'area delimitata dai marcatori tramite la telecamera. Una volta individuati i marcatori **ArUco**, il sistema procede a delimitare l'area di interesse in base alla loro disposizione geometrica. Contestualmente, questa area viene suddivisa in una griglia di elementi con una superficie prestabilita.

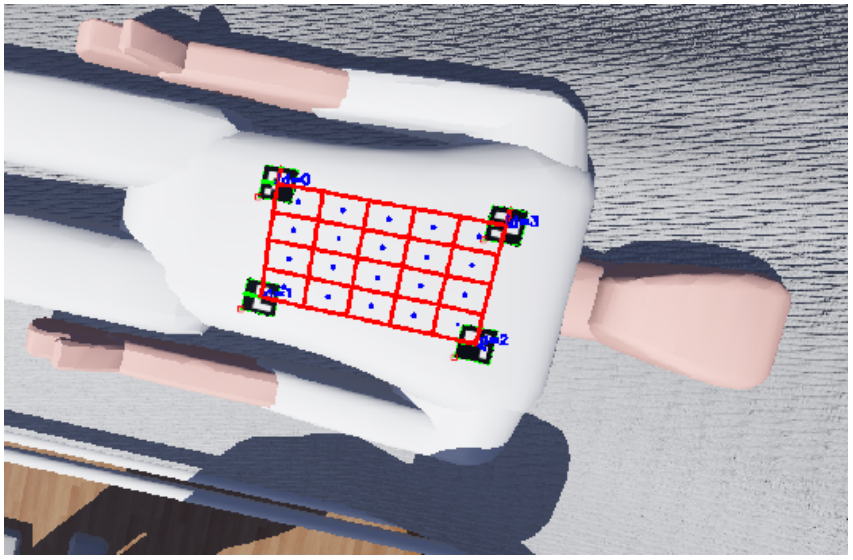

Figura 3: Griglia generata sul paziente in ambiente simulativo

Attesa di conferma pre-sterilizzazione

Al termine dell'operazione di rilevamento, il sistema notifica all'operatore l'avvenuta identificazione di un'area da sterilizzare, entrando successivamente in uno stato di attesa per ricevere un input. L'operatore può confermare l'avvio della procedura di sterilizzazione tramite un input positivo oppure annullare l'operazione con un input negativo, facendo così tornare il sistema allo stato iniziale. Nella simulazione su Webots, questo avviene attraverso la pressione dei tasti "**y**" o "**n**" sulla tastiera: "**y**" per confermare e avviare la sterilizzazione, "**n**" per annullare e resettare il sistema.

Sterilizzazione

In questa fase, il robot si sposta progressivamente da una cella all'altra, attivando di volta in volta la luce UV dedicata alla rilevazione dei batteri. Se all'interno di una cella viene rilevata la presenza di agenti patogeni, viene immediatamente attivata la luce UV germicida per procedere alla sterilizzazione. In caso contrario, il robot prosegue verso la cella successiva, ripetendo il processo fino a quando tutte le celle non saranno state analizzate e, se necessario, sterilizzate. Al termine del ciclo, il robot torna automaticamente alla posizione iniziale.

N.B. Le sorgenti UV impiegate per la rilevazione e la sterilizzazione sono differenti, poiché assolvono a funzioni distinte. In particolare:

- La fase di rilevazione utilizza raggi **UV-A** a 405 nm, i quali inducono fluorescenza in alcuni composti presenti nei batteri, rendendoli rilevabili dalla telecamera;

- La fase di sterilizzazione impiega invece raggi **UV-C** a 222 nm, una lunghezza d'onda in grado di danneggiare il DNA e l'RNA dei microrganismi, impedendone la replicazione e, di conseguenza, eliminando il rischio di infezioni per il paziente.

Nel simulatore **Webots**, non è possibile rappresentare fisicamente i raggi UV. Per questo motivo, la rimozione dei batteri viene gestita a livello di

codice: i batteri vengono considerati eliminati dopo un'esposizione simulata alla luce UV-C per un intervallo di tempo definito.

N.B. in qualsiasi momento deve essere garantita la possibilità di arrestare completamente il robot. A tal fine, mediante la pressione di un tasto specifico (tasto **X** sulla tastiera, nella simulazione Webots), viene attivato l'arresto di emergenza.

In definitiva, si hanno i seguenti stati:

- ***STANDBY***: stato di attesa, il robot è stato fermato dell'operatore / è appena stato acceso;
- ***MOVING_TO_DETECT_SPOT***: il robot si muove verso la posizione ottimale dalla quale determinare i marcatori ArUco;
- ***DETECTING_AT_SPOT***: il robot ricerca i marker ArUco, i quali delimitano l'area da sterilizzare;
- ***AWAITING_CONFERMATION***: il robot ha rilevato i marker ed attende che l'operatore dia conferma per iniziare la sterilizzazione;
- ***MONOTIORING_SELECT_CELL***: il robot verifica qual è la prossima cella della griglia da ispezionare e risolve la cinematica inversa per raggiungere il centro della cella;
- ***MOVING_TO_CELL***: il robot segue la traiettoria della cinematica inversa, arrivando quindi al punto target;
- ***STERILIZING_CELL***: il robot attiva la lampada UV-A per verificare la presenza di batteri nella cella, nel caso in cui venissero rilevati si accenderà una lampada UV-C.

```
while supervisor.step(timeStep) != -1:
      if stato_corrente == STANDBY:
            if in ritorno dalla sterilizzazione:
                  # Torna alla posizione di detection spot
```

```
            # Attendi input dell'utente
            if input_utente == 'S' or input_utente == 's':
                  # Calcola la cinematica inversa per la posizione di
      detection spot
                  # Pianifica la traiettoria verso la posizione di
      detection spot
                  stato_corrente = MOVING_TO_DETECT_SPOT

      elif stato_corrente == MOVING_TO_DETECT_SPOT:
            if not robot_arrivato_posizione_ottimale:
                  # Imposta gli angoli di giunto calcolati dalla
      cinematica inversa
                  pass
            if robot_arrivato_posizione_ottimale:
                  stato_corrente = DETECTING_AT_SPOT

      elif stato_corrente == DETECTING_AT_SPOT:
            # Rileva i marcatori ArUco
            if marcatori_rilevati:
                  # Delimita l'area e suddividila in celle aventi la
      medesima area
                  # Salva in memoria la griglia
                  stato_corrente = AWAITING_CONFIRMATION

            elif stato_corrente == AWAITING_CONFIRMATION:
            # Attendi input dell'utente
            if input_utente == 'S' or input_utente == 's':
                  # Imposta come prima cella da controllare la cella
      di posto 0
                  stato_corrente = MONITORING_SELECT_CELL
            elif input_utente == 'N' or input_utente == 'n':
                  # Resetta il sistema
                  stato_corrente = STANDBY

      elif stato_corrente == MONITORING_SELECT_CELL:
            # Calcola la cinematica inversa per spostare l'end
      effector alla cella successiva
            # Pianifica la traiettoria per spostare l'end effector
      alla cella successiva
            stato_corrente = MOVING_TO_CELL

      elif stato_corrente == MOVING_TO_CELL:
```

```
            # Imposta gli angoli di giunto calcolati dalla cinematica
      inversa
            if robot_arrivato_prossima_cella:
                  stato_corrente = STERILIZING_CELL

            elif stato_corrente == STERILIZING_CELL:
                  # Accendi il detector
                  # Aspetta i tempi di rilevamento
                  # Rileva i contorni dei batteri
                  if contorni_batteri_rilevati:
                        # Accendi lo sterilizzatore
                        # Aspetta i tempi di sterilizzazione
                        # Passa alla cella successiva
                  stato_corrente = MONITORING_SELECT_CELL
                  if non ci sono più celle da sterilizzare:
                        # Calcola la cinematica inversa per tornare
                        alla posizione di detection spot
                        # Pianifica la traiettoria per tornare alla
                        posizione di detection spot
                        stato_corrente = STANDBY

      # Aggiorna il display alla fine di ogni iterazione
```

N.B. Il controllore realizzato è stato pensato per far funzionare Cybersterilizer all'interno dell'ambiente simulativo Webots. Di conseguenza, non è utilizzabile sul modello reale senza aver apportato le opportune modifiche. Alcune delle scelte di design del software sono state fatte esclusivamente per scopi della simulazione, con il fine di mostrare al meglio quello che è il progetto.

End effector

L'**end effector** è il cuore operativo del sistema, essendo progettato specificamente per individuare ed eliminare batteri presenti nella zona chirurgica da trattare. Si tratta di un modulo compatto, a forma cilindrica, montato saldamente sul polso del braccio robotico **ABB IRB 4600**, garantendo precisione, stabilità e ripetibilità durante tutte le fasi del processo. Il dispositivo è composto da **tre componenti principali**, ciascuno con una funzione specifica e sinergica:

- **Videocamera:** funzionale sia alla navigazione che al monitoraggio in tempo reale. Durante la fase di *posizionamento e ricerca dell'area sterile*, la videocamera rileva appositi marker per delimitare con precisione la zona operativa, suddividendola in una griglia di celle. In fase di sterilizzazione, invece, consente la visualizzazione dei batteri, evidenziati grazie all'illuminazione fornita dalla lampada UV-A rilevatrice;

- **Lampada UV-A (rilevatrice):** emette raggi ultravioletti tipo A a 405 nm. Questa sorgente induce fenomeni di fluorescenza nei composti batterici, rendendoli visibili alla videocamera;

- **Lampada UV-C (germicida):** emette raggi ultravioletti di tipo C a 222 nm, noti per la loro capacità di danneggiare irreversibilmente DNA e RNA dei microrganismi. Questa lampada viene attivata **solo nelle celle in cui è stata rilevata la presenza di batteri**.

Design del modello

La progettazione del modello 3D dell'end effector è stata effettuata tramite il software CAD **Shapr3D**. Tale modello racchiude le tre componenti chiave.

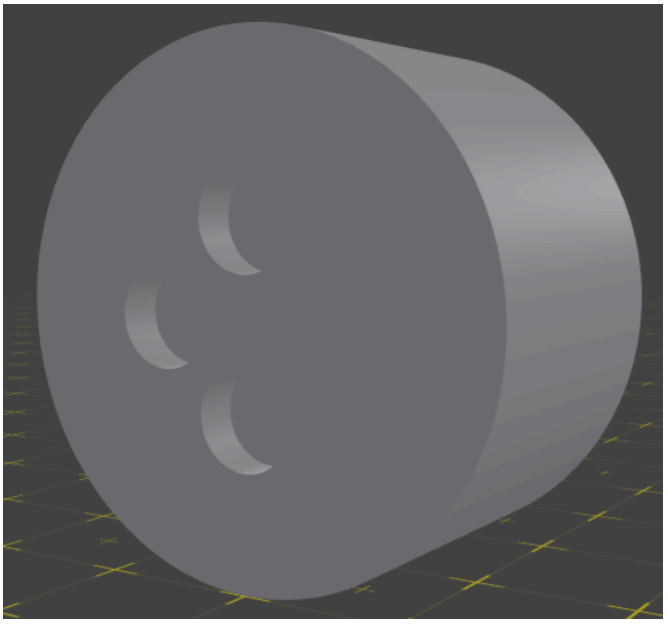

Figura 1: modello 3D della base dell'end effector

Con riferimento alla **Figura 1**, l'end effector è composto da un **corpo principale** e **tre moduli strumentali** integrati. La base presenta una **struttura cilindrica** dotata di **tre cavità simmetriche**, all'interno delle quali sono alloggiati il **sensore ottico (videocamera)** e le **due sorgenti luminose UV**.

Per semplificare la produzione e ridurre i costi di fabbricazione, è stato adottato un **design modulare comune** per i tre componenti. Questo design, mostrato in dettaglio nella **Figura 2**, consente un assemblaggio più rapido e una maggiore intercambiabilità tra i moduli.

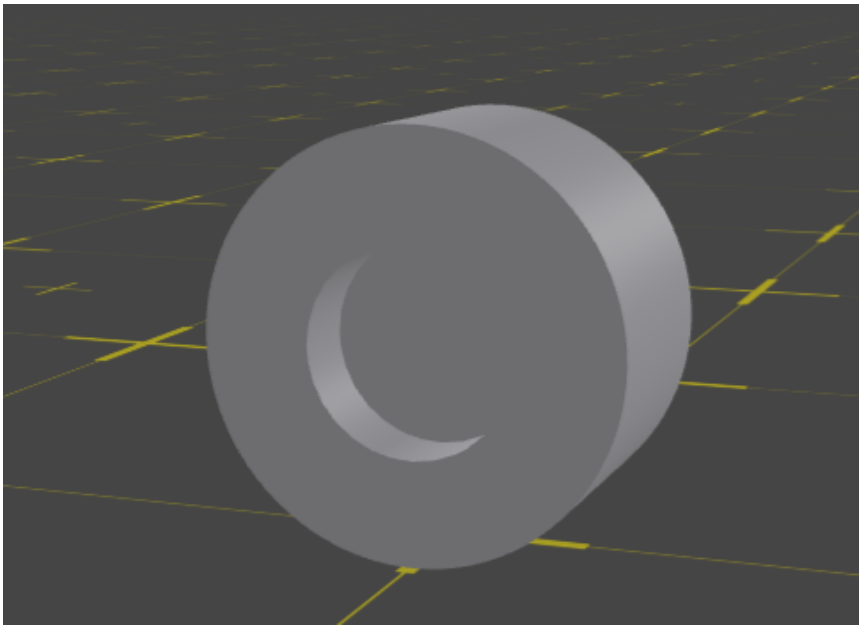

Figura 2: modelli 3D strumento end effector

Una volta realizzati i singoli componenti, l'end-effector viene assemblato integrando i tre moduli strumentali nelle rispettive cavità della base.

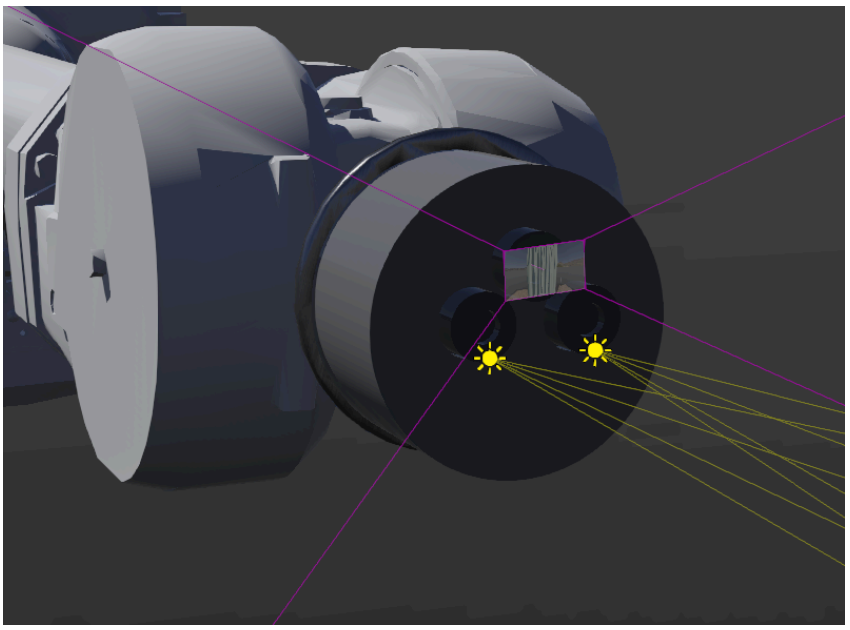

Figura 3: end effector posizionato sul manipolatore, simulazione Webots

Scelta delle lampade UV

Durante il processo di sterilizzazione, il robot impiega due differenti sorgenti UV, ciascuna con funzioni specifiche e proprietà distinte. È quindi fondamentale analizzarne nel dettaglio **l'efficacia contro i batteri** e **la sicurezza per l'uomo**, al fine di garantire un impiego responsabile e conforme agli standard medico-sanitari.

Le lampade impiegate sono le seguenti:

- **Lampada UV-C "Killer"** a **222 nm**
- **Lampada UV-A "Rilevatrice"** a **405 nm**

Di seguito viene analizzato ciascun componente, con particolare attenzione agli aspetti funzionali e di biosicurezza.

Lampada UV-C Killer a 222 nm

La lunghezza d'onda di **222 nm** appartiene alla categoria del cosiddetto **"Far-UV-C"**. Recenti studi scientifici, che si sono intensificati soprattutto durante la pandemia da COVID-19, hanno dimostrato che questa lunghezza d'onda permette un'elevata efficacia germicida, mantenendo tuttavia un

profilo di sicurezza alto per la salute dell'uomo, in quanto la loro capacità di penetrazione nei tessuti biologici è estremamente limitata. I raggi vengono infatti assorbiti quasi interamente dagli strati più superficiali della pelle e della cornea, i quali sono costituiti da cellule anucleate (cioè prive di nucleo), e quindi non soggette a danni genetici.

Alla luce di questo, l'esposizione è **sicura per l'uomo**, a differenza delle comuni lampade UV-C a 254 nm, che richiedono schermature protettive e assenza di persone durante il funzionamento.

Allo stesso tempo i raggi UV-C a 222 nm sono in grado di **danneggiare il DNA e l'RNA** dei microrganismi, impedendone la replicazione e portando alla loro inattivazione permanente. Studi scientifici dimostrano che questa lunghezza d'onda è **altamente efficace** contro una vasta gamma di patogeni, inclusi batteri, virus, spore e ceppi resistenti agli antibiotici, come per esempio lo *Staphylococcus aureus*.

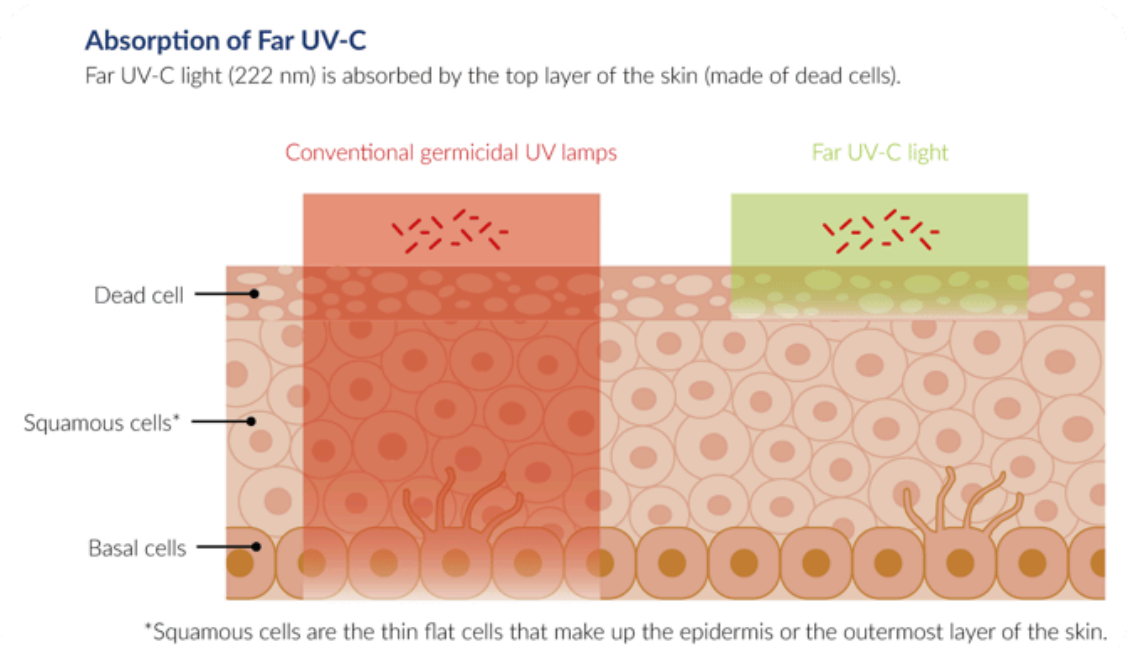

Figura 1: Assorbimento dei raggi Far UV-C sulla pelle

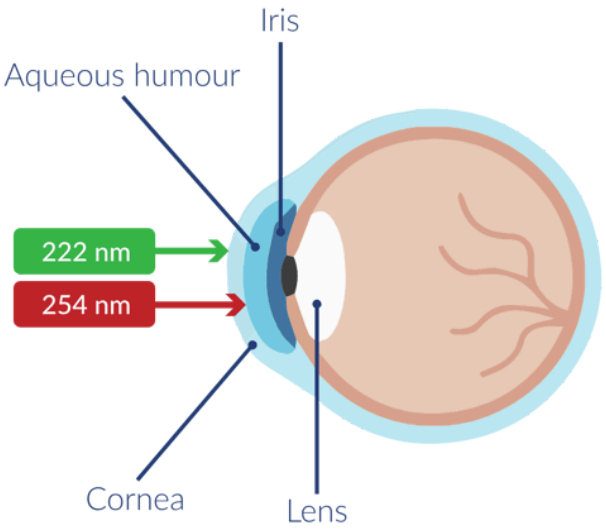

Figura 2: Assorbimento dei raggi Far UV-C sull'occhio

Ad ogni modo, Cybersterilizer è munito di un filtro per radiazioni con lunghezza d'onda superiore ai 222 nm e gli algoritmi software precedentemente citati riducono l'esposizione al tempo necessario per la sterilizzazione.

Lampada UV-A rilevatrice a 405 nm

La lampada UV impiegata per la rilevazione opera a **405 nm**, una lunghezza d'onda al margine tra l'ultravioletto e il visibile (luce blu-violetto). 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, rendendoli identificabili tramite videocamera o persino a occhio nudo.

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**, poiché potrebbe indurre **stress retinico** o danni fototossici alla retina, in particolare in condizioni di alta intensità luminosa.

Gli UV-A a 405 nm sono particolarmente efficaci nel rilevare **presenze batteriche ad alta concentrazione**. Molti batteri e funghi, infatti, sintetizzano **porfirine endogene** durante il proprio metabolismo. Quando queste molecole vengono irradiate con luce UV-A, **emettono fluorescenza visibile**, permettendo una chiara identificazione delle aree contaminate.

È importante notare, tuttavia, che **non tutti i microrganismi producono porfirine**: di conseguenza, alcuni ceppi potrebbero **non risultare visibili** mediante fluorescenza indotta.

Di seguito una tabella con dei microrganismi comuni e la loro eventuale fluorescenza, presa da uno studio condotto con degli UV a 407 nm. Gli effetti sono paragonabili a quelli che ti possono ottenere con i 405nm, quindi quello usato da Cybersterilizer.

Microorganismo	**Picco di fluorescenza**	**Fluorescenza sotto 407 nm**
Actinomyces odontolyticus	635 nm	Sì
Bacteroides intermedius	636 nm, 708 nm	Sì
Pseudomonas aeruginosa	618 nm, 636 nm, 703 nm	Sì
Streptococcus mutans	—	No
Streptococcus faecalis	—	No
Lactobacillus casei	—	No
Lactobacillus acidophilus	—	No

Candida albicans (fungo)	620 nm	Sì
Corynebacterium spp.	620 nm	Sì

Nella **figura 3** si può constatare come aumenti la fluorescenza dei batteri sotto dosi di 405nm, si nota come l'e. coli aumenta la sua fluorescenza notevolmente, mentre lo S. aureus non subisce quasi alcun cambiamento e quindi non risulta visibile.

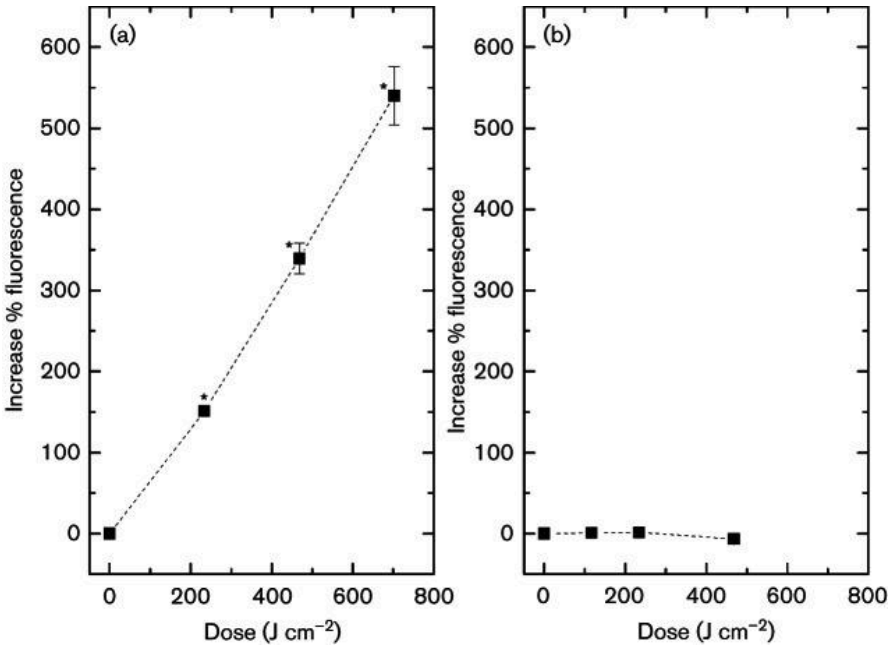

Figura 3: aumento della fluorescenza a seguito dell'esposizione a UV-A. e.coli (a) e S. aureus (b)

Nonostante non tutti i batteri siano rilevabili mediante fluorescenza a 405 nm, studi recenti hanno dimostrato che questa radiazione possiede anche un'efficace azione battericida. Pertanto, la fase di rilevazione con la lampada UV-A si configura anche come una fase parziale di sterilizzazione.

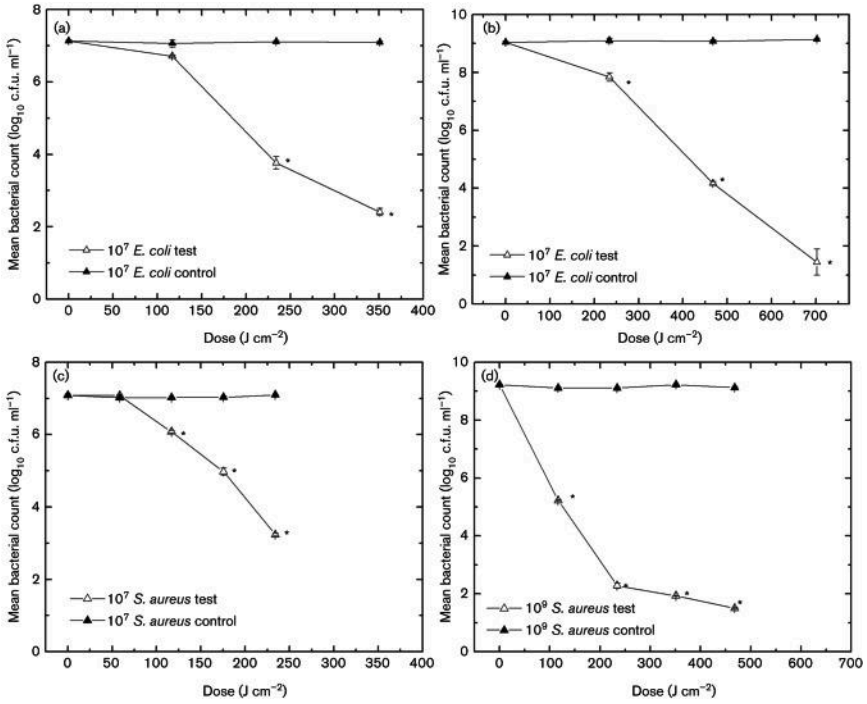

Figura 4: effetti degli UV-A 405nm su E. coli e S. aureus.

Nella **figura 4** è possibile constatare come i batteri reagiscano quando esposti agli UV-A rispetto a quando non lo sono. Si può infatti notare come il numero di batteri scenda considerevolmente, contribuendo ad una riduzione dei rischi di infezione.

Cybersterilizer utilizza la lampada UV-A per scansionare l'intera area designata, inducendo l'autofluorescenza dei microrganismi, così da consentire alla fotocamera di rilevare la presenza di eventuali cariche batteriche. Tuttavia, come evidenziato da numerose ricerche, l'autofluorescenza non è l'unico effetto prodotto da questa radiazione: la lampada UV-A esercita anche un'azione germicida parziale.

Questa duplice funzionalità rende il processo di Cybersterilizer ancora più efficace, in quanto la fase di rilevazione si traduce contemporaneamente in una riduzione attiva della carica batterica, potenziando così l'efficacia complessiva del sistema nella sterilizzazione pre-operatoria.

OpenCV

Cybersterilizer utilizza la libreria **OpenCV** per interpretare le immagini acquisite dalla telecamera sull'end-effector. Questa libreria ricopre quindi un ruolo cruciale ai fini del funzionamento del progetto.

Una delle sue prerogative principali è quella di rilevare i marker ArUco, ciò avviene in particolar modo grazie a **`cv2.aruco`**, il quale identifica i marker e ne stima la posizione nello spazio, permettendo la costruzione della griglia virtuale che suddivide l'area pre-operatoria in celle (la cui area è fissa).

Un altro ruolo importante di OpenCV è quello della **rilevazione dei batteri**. In particolare, la telecamera riesce ad individuare i batteri come delle macchie rosse, ne definisce i contorni e permette quindi la successiva **sterilizzazione** grazie all'ausilio della lampada "killer".

Tutte le immagini elaborate vengono anche mostrate su un **display** (virtuale nel caso della simulazione Webots) in modo tale da mostrare all'operatore ciò che vede Cybersterilizer e cosa ha effettivamente rilevato, questo perché sia la **griglia** che i **contorni** dei batteri **vengono mostrati anche a schermo**.

Marcatori ArUco

I marcatori **ArUco** consistono in dei **pattern binari** in bianco e nero, progettati per essere facilmente rilevabili da una telecamera e per fornire informazioni riguardanti la posizione e l'orientamento del marker nello spazio tridimensionale tramite un'immagine bidimensionale.

Nel caso di OpenCV, esistono diversi tipi di pattern ArUco, dove ognuno di essi appartiene ad un **dizionario**, ovvero un insieme predefinito di marker, ciascuno con un **ID** univoco e un pattern binario. Cybersterilizer utilizza il dizionario `DICT_4X4_250` il nome del dizionario fornisce tutte le informazioni riguardo i marker presenti al suo interno.

DICT_4X4_250

- DICT: nome del dizionario;
- 4X4: dimensione del pattern, indica quante celle binarie compongono il marker;
- 250: numero di marker disponibili, quindi il numero di ID univoci.

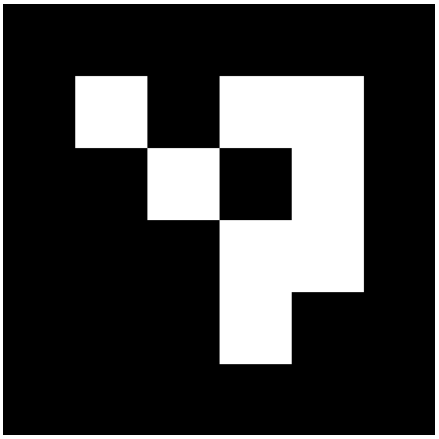

Figura 1: ArUco 4x4 con id=0

Teoricamente il numero di marker realizzabili in una griglia 4x4 è di gran lunga maggiore in quanto si avranno 2^{16} pattern (ovvero 65 546 pattern). Tuttavia si opta per un numero di gran lunga più limitato di pattern, questo per evitare di avere pattern troppo simili tra loro i quali aumenterebbero il rischio di errore, inoltre alcuni pattern solo esclusi, per esempio non vengono utilizzati dei marker simmetrici o troppo ambigui.

Algoritmo di rilevamento dei marker

I marker ArUco sono stati progettati per essere facilmente rilevati e identificati da algoritmi di visione artificiale. Di seguito un'analisi delle funzioni principali utilizzate nel controller di Cybersterilizer in particolare prestando attenzione ai passaggi cruciali per la rilevazione dei marker.

Inizializzazione del dizionario ArUco

Come si può vedere nel **code snippet 1** viene inizializzato il dizionario già menzionato precedentemente, ovvero utilizzando DICT_4X4_250.

```
# Initialize the ArUco marker dictionary and parameters
```

```
aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_250)
detector_params = aruco.DetectorParameters()
```

Code snippet 1: inizializzazione ArUco

Rilevamento ArUco

Quando viene richiamata la funzione aruco.detectMarkers, i marcatori vengono di fatto rilevati.

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

Code snippet 2: Funzione che rileva gli ArUco

All'interno della funzione vengono eseguiti i seguenti passaggi:

- **Preprocessing dell'immagine**: il fotogramma viene convertito in scala di grigi per limitare il possibile rumore dell'immagine e definisce una sogia che definisce un pixel bianco da quello nero;
- **Rilevamento dei quadrati**: OpenCV cerca le cornici nere dei marker;
- **Estrazione del contenuto**: ogni quadrato viene corretto prospetticamente e si estrae la griglia binaria;
- **Matching con dizionario**: viene confrontata la matrice di 0 ed 1 e si trova il match più vicino possibile e se ne estrae l'ID.

Parametri intrinseci della camera

Per riuscire a stimare la posizione 3D dei marker, l'algoritmo necessita dei parametri intrinseci della telecamera del robot, i quali vengono inseriti all'interno di una matrice (di calibrazione) che verrà presa in input da OpenCV per eseguire una stima della posa del marker.

```
# Calculate camera parameters
img_width, img_height = camera.getWidth(), camera.getHeight()
```

```
fov = camera.getFov()

# Calculate the camera intrinsic matrix. Camera is assumed to be the
pinhole model.
cx, cy = img_width / 2, img_height / 2
fx = cx / np.tan(fov / 2)
camera_matrix = np.array([[fx, 0, cx], [0, fx, cy], [0, 0, 1]],
dtype=np.float32)
dist_coeffs = np.zeros((4, 1), dtype=np.float32)
```

Code Snippet 3: Estrazione e calcolo della matrice intrinseca

La matrice di calibrazione della fotocamera presenta la seguente struttura.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Dove:

- f_x e f_y sono le **lunghezze focali**;
- c_x e c_y: sono i **centri ottici**.

Non essendo necessario considerare la distorsione, si considera il coefficiente di distorsione nullo.

Pose Estimation

Grazie ai parametri intrinseci calcolati in precedenza sarà possibile ottenere la posizione 3D dei marker rispetto alla camera, i quali verranno utilizzati successivamente per la realizzazione della griglia.

```
# Do pose estimation for each marker
_, translations, _ = aruco.estimatePoseSingleMarkers(
      corners, MARKER_SIZE, camera_matrix, dist_coeffs)
```

Code snippet 4: Calcolo della posizione

I marker ArUco offrono numerosi vantaggi, soprattutto negli ambiti della robotica, in particolar modo si possono apprezzare diversi vantaggi

dell'utilizzo di questa tecnologia, motivo del quale è stata scelta per Cybersterilizer:

- **Riconoscimento robusto e veloce**: I marker sono stati progettati per essere rilevati anche in condizioni di scarsa luminosità, con parziale occlusione o con angolazioni che non permettono una visibilità perfetta. L'algoritmo riesce a rilevare velocemente i marker (praticamente in tempo reale) nonostante le possibili condizioni non ideali;
- **Identificazione univoca**: sullo schermo possono essere presenti diversi marker, nonostante ciò l'algoritmo è capace di distinguere il loro ID univoco con precisione;
- **Stima della posizione**: l'algoritmo è capace di riconoscere l'orientazione e la posizione di ogni marker, il ché è fondamentale per Cybersterilizer il quale necessita delle posizioni dei marcatori per realizzare la griglia;
- **Facili da generare e stampare**: gli ArUco sono facilmente generabili da un algoritmo e stampabili con una qualsiasi stampante, non richiedendo quindi tecnologie aggiuntive;
- **Bassa probabilità di errore**: per via delle scelte di progettazione di ogni dizionario, la probabilità di errore è minima, questo anche grazie al fatto che gli ArUco hanno una distanza di hamming minima tra loro.

Griglia dinamica e logica di generazione

Una componente fondamentale del sistema Cybersterilizer è la **griglia virtuale** che suddivide l'area operatoria in celle di superficie uniforme. Questa griglia viene generata **una singola volta** durante la fase di rilevamento del marker ArUco, e rimane **immutata** per tutta la durata del ciclo di sterilizzazione.

Motivazioni della generazione singola

La scelta di generare la griglia una singola volta è motivata da:

- **ArUco fissati**: i marker ArUco vengono utilizzati in una fase iniziale per generare la griglia e quindi definire l'area operativa, tuttavia nello stato attuale del progetto, una volta generata la griglia è **potenzialmente** possibile rimuovere i marker ArUco in quanto le coordinate di ogni cella sono state salvate;
- **Efficienza computazionale**: non dover ricalcolare la griglia per ogni singolo frame rende il codice più leggero ed efficiente;
- **Coerenza operativa**: l'operatore vedrà su schermo un frame della telecamera, in particolar modo il frame sulla quale viene generata la griglia, in base a quel frame l'operatore può decidere se iniziare la fase di sterilizzazione o meno, quindi mantenere i punti basandosi su quelli confermati dall'operatore risulta cruciale.

Nel controllore, questa logica è implementata nello stato ***DETECTING_AT_SPOT***, dove viene chiamata la funzione:

```
        # Perform detection and get the annotated image with cell
centers
        annotated_image, detected_cam_centers, detected_world_centers
= run_detection_and_get_annotated_image(raw_camera_image_buffer)
```

Code snippet 4: generazione centri delle celle

Questa funzione esegue:

- Rilevamento dei marker con `aruco.detectMarkers(...)`
- Stima della posa con `aruco.estimatePoseSingleMarkers(...)`
- Ordinamento dei marker con `sort_quad(...)`
- Calcolo delle dimensioni della griglia con `compute_grid_dims(...)`
- Interpola i centri delle celle con `interpolate_world_centers(...)`

I risultati saranno le coordinate dei centri rispetto alla telecamera e rispetto al mondo simulato di Webots.

La funzione `sort_quad()`

La funzione `sort_quad()` ordina i marker ArUco rilevati in senso orario, fa in modo che la griglia abbia la forma corretta, senza che venga distorta o invertita.

```
# Function to sort the corners of the detected markers
def sort_quad(corners, ids):

    # Sort the corners based on their IDs
    pts = np.array([c.reshape(-1,2).mean(axis=0) for c in
corners[:4]], np.float32)
    center = pts.mean(axis=0)
    angles = np.arctan2(pts[:,1]-center[1], pts[:,0]-center[0])
    order = np.argsort(angles)

    return pts[order], ids.flatten()[order]
```

Code snippet 5: La funzione sort_quad()

La funzione compute_grid_dims()

La funzione `compute_grid_dims()` calcola il numero di righe e colonne della griglia in base all'area delimitata dai marker. In particolare, permette di adattare dinamicamente la griglia alla dimensione dell'area operatoria mantenendo costante l'area di ogni singola cella.

```
    # Calculate the area of the quadrilateral by dividing it into two
triangles
    # and summing their areas.
    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]))
    # Convert total area from m^2 to cm^2
    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))
```

Code snippet 6: compute_grid_dims()

In breve:

- Divide il quadrilatero in due triangoli;
- Calcola l'area totale e la converte in cm^2;
- Divide l'area basandosi sull'area desiderata di ogni cella;
- Calcola il numero di righe e colonne.

La funzione interpolate_world_centers(storted_ids, rows, cols)

Questa funzione calcola le coordinate 3D dei centri delle celle nel sistema di riferimento partendo dai marker ordinati

```
    # Ensure we have exactly 4 markers
    p0, p1, p2, p3 = [world_positions[id_] for id_ in sorted_ids]
    # Initialize a dictionary to hold the 3D centers of the grid
cells
    centers3d = {}
    # Iterate through the grid cells and compute their 3D centers
    for i in range(rows):
        for j in range(cols):
            u, v = (j + 0.5) / cols, (i + 0.5) / rows
            # Compute the 3D center of the cell using bilinear
interpolation
            centers3d[(i,j)] = (
                (1-u) * (1-v) * p0 + u *(1-v) * p1 + u * v * p2 +
(1-u) * v * p3
            )
```

Code snippet 7: interpolate_world_centers

In breve:

- Usa l'interpolazione bilineare tra i 4 marker ordinati;
- Per ogni cella, calcola la posizione 3D.

Conversione da coordinate Webots a coordinate del robot

Nel contesto simulativo di Webots, le coordinate ottenute tramite OpenCV (relative alla camera) devono essere convertite nel sistema di riferimento del

robot. Questo passaggio è necessario per permettere al braccio robotico di muoversi correttamente verso i centri delle celle.

Nel codice questa conversione è gestita dalla funzione nel **code snippet 8**:

```
    # Get the arm position in Webots coordinates.
    armPosition = arm.getPosition()

    # Compute the position of the target relatively to the arm.
    # x and y axis are inverted because the arm is not aligned with
the Webots global axes.
    x = -(targetPosition[1] - armPosition[1])
    y = targetPosition[0] - armPosition[0]
    z = targetPosition[2] - armPosition[2]
```

Code snippet 8: conversion to robot coordinates

Questa funzione **non rappresenta un approccio realistico** per un sistema fisico, ma è una **semplificazione per la simulazione Webots**. In un contesto reale si dovrebbero implementare degli algoritmi di calibrazione della camera.

Rilevamento dei batteri tramite la funzione findContours

Il metodo findContours() utilizza un algoritmo di computer vision per rilevare gli oggetti all'interno di una scena, in questo contesto, rilevare i batteri. L'algoritmo in questione sebbene datato, offre un ottimo bilancio tra performance e precisione, evitando modelli di machine learning/deep learning che sono computazionalmente onerosi per il calcolo dell'inferenza; inoltre, bisogna anche specificare che il braccio robotico è composto da microcontrollori con risorse hardware limitate, pertanto l'utilizzo di un algoritmo con basso costo computazionale è importante tenerlo in considerazione. Lo studio è stato eseguito su immagini composte esclusivamente da pixel di valore binario: 0 o 1. Questa semplificazione permette un'analisi più rigorosa della loro struttura topologica.

Un'immagine digitale binaria viene rappresentata come una **matrice bidimensionale** di pixel, dove ogni pixel è identificato da una coppia di coordinate (i, j):

- i rappresenta l'indice di riga (aumenta dall'alto verso il basso);
- j rappresenta l'indice di colonna (aumenta da sinistra verso destra).

Per convenzione, i bordi dell'immagine (prima e ultima riga, prima e ultima colonna) sono **riempiti con 0-pixel**, costituendo quella che viene chiamata **cornice** (*frame*) dell'immagine. Questo consente di definire in modo coerente il concetto di sfondo e di garantire che tutti i buchi siano interni all'immagine.

In una immagine binaria:

- I **0-pixel** rappresentano tipicamente lo sfondo;
- I **1-pixel** rappresentano oggetti o regioni di interesse.

Le componenti connesse rappresentano quei pixel che possiedono caratteristiche simili (es. valori di intensità):

- Una **1-componente** è un insieme connesso di 1-pixel.
- Una **0-componente** è un insieme connesso di 0-pixel.

Le componenti connesse possono essere classificate in due categorie:

- **Sfondo**: una 0-componente che tocca il bordo dell'immagine;
  Buco: una 0-componente interamente circondata da 1-pixel.

Possiamo definire quindi:
- Un pixel è **4-connesso** con i suoi vicini a nord, sud, est, ovest.
- Un pixel è **8-connesso** se include anche i vicini diagonali.

Un pixel 1-pixel (i,j)(i, j)(i,j) è un **punto di bordo** se esiste almeno un 0-pixel (p,q)(p, q)(p,q) nel suo intorno (4 o 8, a seconda del caso). Questo rappresenta il confine tra una componente 1 e una componente 0. Il **bordo esterno** è l'insieme di punti di bordo tra una 1-componente e il background che la circonda direttamente. Il **bordo del buco** è l'insieme di punti di bordo tra una 1-componente e un buco (0-componente) che essa circonda direttamente.

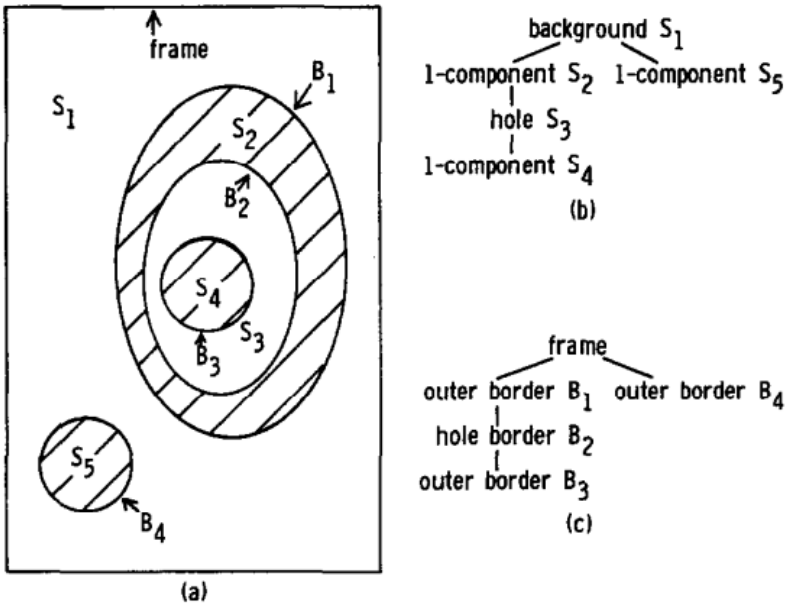

Figura 1: componenti connesse.

Entrambi i tipi di bordi sono costituiti **solo da 1-pixel**, anche nel caso del bordo del buco. La **relazione di circondamento** (*surroundness*) è una delle nozioni topologiche più importanti.

Siano S1 e S2 due componenti connesse. Si dice che **S1 circonda S2** se:

- Ogni percorso 4-connesso che va da un pixel di S2 al bordo dell'immagine passa attraverso almeno un pixel di S1;
- Se, inoltre, esiste almeno un **punto di bordo** tra S1 e S2, si dice che S1 **circonda direttamente** S2. Questa definizione permette di rappresentare in modo rigoroso la **gerarchia tra oggetti e buchi** nell'immagine.

```
    # Convert the raw image buffer to a BGR 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),2) # Verde brillante in formato
BGR


        # Draw a circle at the center of the contour
        center_x = x + w // 2
        center_y = y + h // 2
        radius = max(5, max(w, h) // 8)
        cv2.circle(img_bgr, (center_x, center_y), radius, (0, 255,
0), -1)
        cv2.circle(img_bgr, (center_x, center_y), radius + 2, (0, 0,
255), 2)
```

Code snippet 9: implementazione del metodo findContours()

Il metodo elabora l'immagine grezza catturata dalla camera, nello spazio dei colori RGB. Questa viene convertita nello spazio BGR, in quanto Opencv

elabora le immagini a colori in questo formato, e successivamente viene convertito nello spazio HSV (Hue, saturation, brightness). Lo spazio dei colori HSV riesce a separare le informazioni del croma dal valore di intensità del pixel, quindi al variare dell'intensità luminosa il colore non cambia, invece nel campo BGR (RGB) è sensibile alla variazione dell'intensità luminosa.

Essendo che l'algoritmo findContours lavora con immagini binarie è necessario utilizzare delle maschere binarie della forma del batterio. Le maschere rappresentano uno strumento molto potente in quanto si adattano alla forma geometrica dell'oggetto da rilevare. Il metodo `cv2.morphologyEx` serve a **ripulire la maschera** da **rumore** e **piccoli artefatti** che potrebbero interferire con il rilevamento dei contorni, i flag `cv2.RETR_EXTERNAL` e `cv2.CHAIN_APPROX_SIMPLE` servono a mantenere i contorni esterni e considerare i punti più importanti. Infine viene passato quanto dichiarato nel metodo findContours che restituisce le coordinate dell'oggetto interessato(batterio) che verranno evidenziate sul display mediante opportuni metodi di disegno cv2.circle.

Sviluppi futuri

L'attuale implementazione del sistema robotico rappresenta un solido punto di partenza, ma come ogni tecnologia in fase di evoluzione, esistono ampi margini di miglioramento che possono essere esplorati e sviluppati nel tempo. L'obiettivo è rendere il sistema sempre più efficiente, accessibile, intelligente e facilmente integrabile con altri strumenti tecnologici.

I futuri sviluppi puntano a potenziare sia l'infrastruttura hardware sia le capacità software del robot, con un'attenzione particolare alla **riduzione dei costi**, al **miglioramento delle prestazioni**, all'**adozione di tecniche avanzate come il Machine Learning** e alla **progettazione di API** per estendere le funzionalità in maniera modulare e scalabile.

Riduzione del costo di implementazione

Uno dei principali obiettivi futuri è la riduzione dei costi complessivi di implementazione del sistema robotico, al fine di renderlo accessibile a un numero maggiore di strutture sanitarie, comprese quelle con budget limitati. Questo traguardo può essere raggiunto tramite:

- **Ottimizzazione della componentistica hardware**, privilegiando soluzioni più economiche ma comunque affidabili, come sensori open-source o microcontrollori integrati;
- **Miniaturizzazione dei componenti** e semplificazione dell'architettura meccanica per contenere i costi di produzione e manutenzione;
- **Produzione su scala**: una progettazione orientata alla replicabilità permette economie di scala che incidono positivamente sul prezzo finale.

Adozione di hardware più potente

Per migliorare le prestazioni del sistema, si prevede l'adozione di hardware più potente in grado di gestire in modo efficiente compiti computazionalmente intensivi, quali:

- Elaborazione in tempo reale di segnali visivi e sensoriali ad alta risoluzione;
- Pianificazione dinamica del movimento con intelligenza artificiale onboard;
- Maggiore precisione e reattività grazie a motori e attuatori di ultima generazione, eventualmente dotati di feedback aptico.

Introduzione del Machine Learning

L'adozione del Machine Learning rappresenta una svolta significativa nell'autonomia e adattabilità del sistema robotico.

Le possibili applicazioni includono:

- **Riconoscimento avanzato degli oggetti e delle superfici** da trattare, migliorando la precisione dell'azione di sterilizzazione;

- **Apprendimento continuo da esperienze passate**, permettendo al robot di ottimizzare i propri movimenti e strategie operative in base all'ambiente;

- **Predizione di guasti o malfunzionamenti** tramite tecniche di manutenzione predittiva basate su modelli addestrati su dati storici;

- **Interazione uomo-robot migliorata**, grazie alla capacità del sistema di interpretare comandi vocali, gestuali o comportamenti dell'operatore umano.

L'integrazione di modelli di apprendimento supervisionato e non supervisionato, eventualmente supportati da reti neurali profonde, aumenterebbe l'intelligenza operativa del sistema.

Sviluppo di API per l'estensione delle funzionalità

Per garantire la scalabilità e l'integrazione del robot in contesti sempre più complessi, è fondamentale sviluppare un set di **API (Application Programming Interface)** ben documentato e modulare. Queste API potrebbero offrire:

- **Interfacce standard per il controllo remoto** del robot da applicazioni mobili o desktop;

- **Estendibilità del software**, consentendo a terze parti di sviluppare nuovi moduli, plugin o funzionalità aggiuntive (es. protocolli personalizzati di sterilizzazione);

- **Integrazione con sistemi informativi ospedalieri (HIS)** o altri dispositivi medicali in uso;

- **Supporto alla telemedicina e al monitoraggio remoto**, con possibilità di log, diagnosi e aggiornamenti da cloud.

Una progettazione orientata alle API favorisce lo sviluppo di un ecosistema attorno al robot, aumentandone il valore e la versatilità in ambito sanitario.

Bibliografia

- UV medico far UV-C
- 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
- Application of Fluorescence Spectroscopy for Microbial Detection to Enhance Clinical Investigations
- Paper FindContours Algorithm
- Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods
- A computer tool for simulation and analysis: the Robotics Toolbox for MATLAB