ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Cyber-Physical System Programming
Università degli Studi di Bologna
A.A. 2024/25

Crazyflie Slam

Implementazione di un algoritmo SLAM in real time per drone Crazyflie.

Progetto svolto da:

- Luca Fantini
- Simone Rambelli
- Alex Vandi
- Pierfrancesco Vazzano

Contenuti della Presentazione

Panoramica del progetto

Introduzione alla struttura del Crazyflie_SLAM.

Architettura hardware

Componenti fisici e configurazione del sistema.

SLAM, Lettura dati e mappatura

Tecniche algoritmiche e implementazione pratica.

Progetto

Esempi sia in tempo reale che da analisi dei dataset.

Algoritmo SLAM

Acquisizione dati

Il FlowDeck fornisce stime di posizione relative al movimento del drone.
Il RangerDeck misura le distanze dagli ostacoli circostanti.

Elaborazione

I dati ToF vengono utilizzati per aggiornare una griglia di occupazione, rappresentando le probabilità di presenza di ostacoli nello spazio

Mappatura

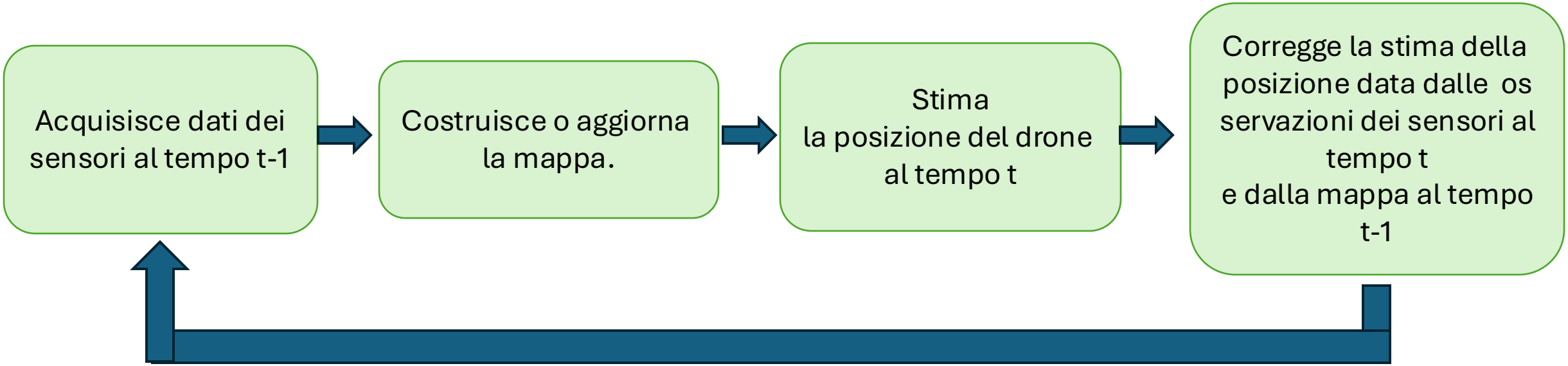
Utilizzando i dati del FlowDeck, il sistema stima la posizione corrente del drone.

Ottimizzazione

Grazie ad algorimi integrati nel codice

Cos'è lo SLAM?

Algoritmo che costruisce la mappa di un ambiente sconosciuto aggiornando contemporaneamente la posizione del drone.



💡 **Obiettivo:**

Trovare il movimento che spiega meglio la differenza tra le osservazioni dei sensori al tempo t-1 e t, dato ciò che si sa sull'ambiente

Mapping e Localizzazione

Mapping:

- Usa la stima di states, attitude e obstacles per identificare i target.
- Calcola le coordinate delle celle occupate e quelle libere lungo il percorso del sensore (algoritmo di Bresenham).
- Applica regole di aggiornamento per le celle.

Localizzazione:

- Utilizzo di un particle filter e stima un punteggio
- Aggiunge rumore casuale per differenziare le particelle (non usato nel nostro caso)
- Aggiorna i pesi delle particelle e stima lo stato attuale con la particella con il punteggio più alto.

Struttura Hardware

Crazyflie 2.1

Piattaforma principale.

Pc

Computer per l'elaborazione dei dati inviati dal Crazyflie.

Multiranger Deck

Rilevamento ostacoli con sensori ToF.

Flow deck

Sensore ottico per il rilevamento del movimento e della distanza dal suolo.

Struttura Software

[SLAM DATA]
state: {x,y,z}
attitude: {yaw}
obstacles: {
  front, right,
  back, left, up
}

python

Components:
- CFlib: crazyflie communication over the appchannel.
- SLAM: algorithm implementation that processes the received data and updates the model with each iteration.

freeRTOS

Task:
- using the Logging system, fetch the data and send them over the appchannel.

Equipment:
- ranger deck
- flow deck

Lettura Dati

Dati richiesti dal'algoritmo SLAM

```
typedef struct
{

//Flow Deck

    float yaw;          // attitude
    float x;            //state
    float y;


//Ranger Deck

    uint16_t front;     //obstacles
    uint16_t back;
    uint16_t left;
    uint16_t right;
    uint16_t up;
} SensorData_t;
```

Logging system

```
//Flow Deck

logVarId_t logIdStateEstimateYaw = logGetVarId("stateEstimate", "yaw");

logVarId_t logIdStateEstimateX = logGetVarId("stateEstimate", "x");
logVarId_t logIdStateEstimateY = logGetVarId("stateEstimate", "y");


//Ranger Deck

logVarId_t logIdRangeFront  = logGetVarId("range", "front");
logVarId_t logIdRangeBack   = logGetVarId("range", "back");
logVarId_t logIdRangeLeft   = logGetVarId("range", "left");
logVarId_t logIdRangeRight  = logGetVarId("range", "right");
logVarId_t logIdRangeUp     = logGetVarId("range", "up");
```

Versione Asincrona

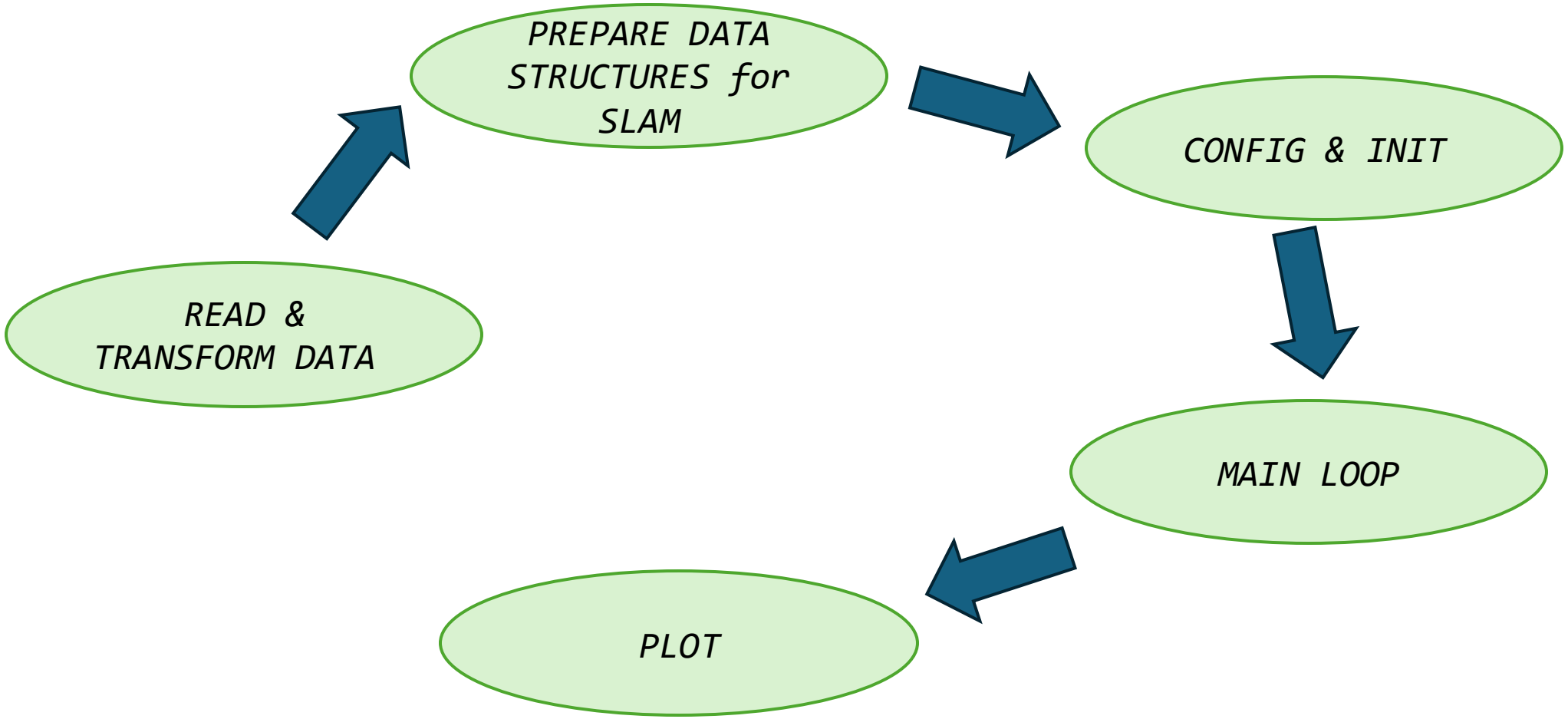
Comunicazione con il PC client

```
# |         HANDLER            |

def custom_sensor_data_handler(packet):
    """
    Callback to handle incoming CRTP packets from the Crazyflie.
    """
    # The packet is a byte array, so we need to unpack the data

    format_string = "f f f H H H H H bb"

    if len(packet) == (struct.calcsize(format_string)):
        # Unpack the data from the packet
        yaw, x, y, front, back, left, right, up, padding1, padding2 = struct.unpack(format_string, packet)


        data = [front, right, back, left, x, y, yaw]
        log_data(data)
    else:
        print(f"Received unknown packet: {packet.decode('utf-8', errors='ignore')}")
        print(f"Raw packet: {packet}")
```

```
# |          CONFIG              |

def log_data(data):
    global log_id
    data = [log_id, *data]
    log_id += 1

    with open(log_file, mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(data)
```

```
# |          MAIN              |

if __name__ == '__main__':
    logging.basicConfig(level=logging.ERROR)

    # Create a Crazyflie instance
    cf = Crazyflie(rw_cache='./cache')

    # Initialize the drivers for Crazyflie
    init_drivers(enable_debug_driver=False)

    # Register the callback for handling incoming CRTP packets
    cf.appchannel.packet_received.add_callback(custom_sensor_data_handler)
```

SLAM Asincrono da file CSV



SLAM Asincrono da file CSV

```
# |    READ & TRANSFORM DATA     |

# Read CSV file
data_frame = pd.read_csv(file_path)

# Extract columns and apply the transfromations
id = data_frame['id']
front = data_frame['front'] / 1000 # from [mm] to [m]
right = data_frame['right'] / 1000
back = data_frame['back'] / 1000
left = data_frame['left'] / 1000
x = data_frame['x']
y = data_frame['y']
yaw = data_frame['yaw'] * np.pi / 180.0 # from [deg] to [rad]
```

```
# |  PREPARE DATA STRUCTURES for SLAM  |

# Create "ranges" that contains front, right, back and left
ranges = np.array([front, right, back, left])

# Create "scanangles" that contains the four   direction of information
scan_angles = np.array([0, 1/2*np.pi, np.pi, 3/2*np.pi]).T

# Create "states" that contains x, y and yaw
states = np.array([x, y, yaw])

# Get Delta of the states
motion_updates = np.diff(states, axis=1, prepend=np.zeros((3, 1)))

states = np.cumsum(motion_updates, axis=1)
```

```
# |      CONFIG & INIT          |

# Useful values
system_noise_variance = np.diag([0, 0, 0])
correlation_matrix = np.array([
    [0, -1],
    [-1, 10],
])
slam_states = np.zeros_like(states)

# Init the SLAM agent
slam_agent = SLAM (
    params=init_params_dict(size=1, resolution=100),
    n_particles=int(args.n_particles),
    current_state=states[:, 0],
    system_noise_variance=system_noise_variance,
    correlation_matrix=correlation_matrix,
)

# Main loop
plt.ion()
# Update the plot every <skip> iteration
skip = 40

fig = plt.figure(figsize=(7,7))
  (old_front, old_right, old_back, old_left) = ranges[:, 0]
```

```
# |          MAIN LOOP          |

for t in range(states.shape[1]):
    # filter values
    ...code that filters zero values and full scale values...

    slam_states[:, t]  = slam_agent.update_state(
        ranges[:, t],
        scan_angles,
        motion_updates[:, t], # un lista di delta nelle posizioni
    )
```

```
# |          PLOT               |

if t % skip == 0:
    plt.clf()
    slam_map = slam_agent.map
    idx_slam = discretize(slam_states[:2, :], slam_agent.params)
    idx_noise = discretize(states[:2, :], slam_agent.params)


    plt.imshow(slam_map, cmap="gray") # vmin=-50, vmax=100)
    plt.plot(idx_slam[1, :t+1], idx_slam[0, :t+1], "-r", label="slam")
    plt.title(f"Iterazione #{t}")
    plt.legend()
    plt.draw()
    plt.pause(0.01)
```

Real-time SLAM

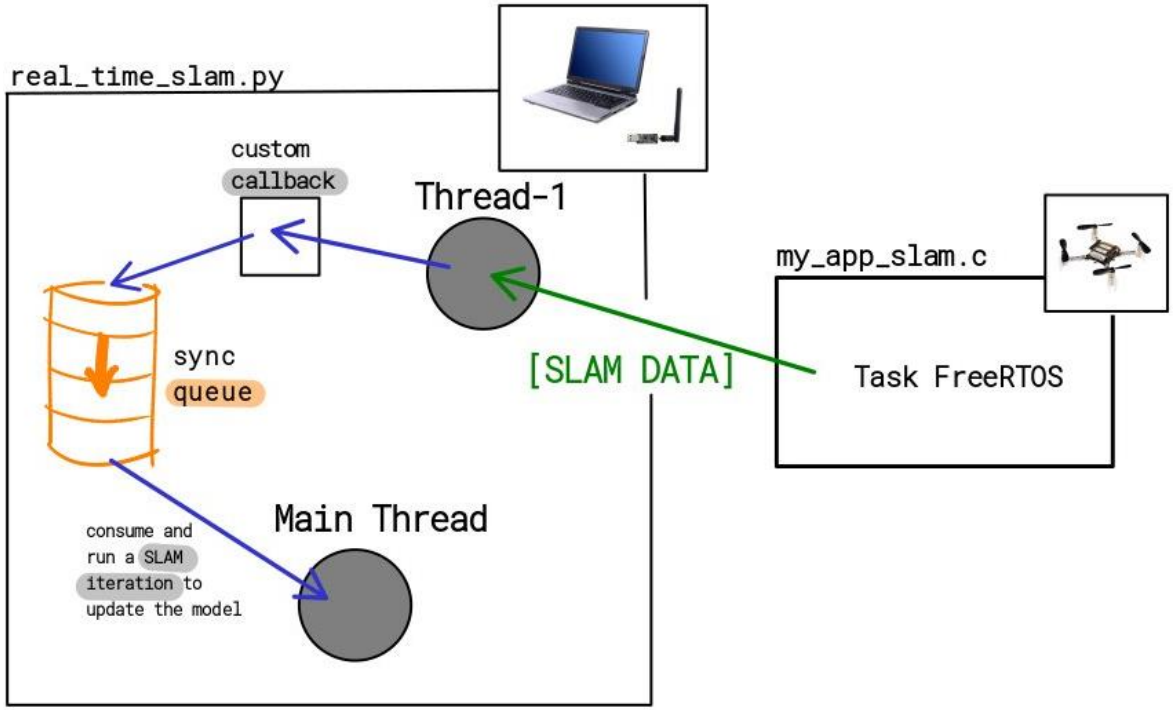
CONFIGURAZIONE INIZIALE

ACQUISIZIONE DATI SENSORE

INIZIALIZZAZIONE SLAM AGENT

LOOP– AGGIORNAMENTO SLAM

MAPPING E LOCALIZZAZIONE



real_time_slam.py

Configurazione Iniziale

- Lettura dei parametri da linea di comando
- Inizializzazione dei driver Crazyflie
- Connessione via radio al drone
- Impostazione della funzione di callback per i dati dal sensore

```
parser = argparse.ArgumentParser()
parser.add_argument("--n_particles", default=400)
parser.add_argument("--cf_uri", default="radio://0/83/2M/E7E7E7E7EA")

init_drivers(enable_debug_driver=False)
cf = Crazyflie(rw_cache='./cache')
cf.appchannel.packet_received.add_callback(custom_sensor_data_handler)
```

Acquisizione Dati dal Sensore

- Decodifica del pacchetto radio in arrivo
- Estrazione di: yaw, posizione, distanze (front, back, etc)
- Inserimento dei dati nella coda condivisa per l'elaborazione

```
# common data structure to exchange and buffer data
data_queue = queue.Queue()

def custom_sensor_data_handler(packet):

 format_string = "f f f H H H H H bb"

if len(packet) == (struct.calcsize(format_string)):
        # Unpack the data from the packet
        yaw, x, y, front, back, left, right, up, padding1, padding2 = struct.unpack(format_string, packet)
        data = [front, right, back, left, x, y, yaw]

        data_queue.put([front, right, back, left, x, y, yaw])
```

Inizializzazione SLAM Agent

- Definizione della mappa vuota con dimensione e risoluzione
- Stato iniziale: posizione (x, y) e orientamento (yaw)
- Impostazione del rumore del sistema (matrice di covarianza)
- Creazione dello SLAM agent con le particelle richieste

```
init_state = np.zeros(shape=(3,1))
system_noise_variance = np.diag([0.0, 0.0, 0.0])

freq = 10 # sample frequency [Hz]
skip = freq
size = 1
resolution = 100

correlation_matrix = np.array([
[0, -1],
[-1, 10],
])

# Init the SLAM agent
slam_agent = SLAM (
            params=init_params_dict(size, resolution),
            n_particles=int(args.n_particles),
            current_state=init_state,
            system_noise_variance=system_noise_variance,
            Correlation_matrix=correlation_matrix,
)
```

Loop Principale - Aggiornamento SLAM

- Lettura dei dati sensore dalla coda
- Calcolo del movimento relativo tra iterazioni
- Aggiornamento dello stato stimato e della mappa
- Salvataggio dello stato stimato corrente per visualizzazione

```
while True: # [Main thread]
            (front, right, back, left, x, y, yaw) = data_queue.get()

            # filter values

             ...code that filters zero values and full scale values...

            # prepare data for the SLAM algorithm

            # yaw in rad
            yaw = yaw * np.pi / 180.0

            # ranges in meters
            ranges = np.array([front / 1000, right / 1000, back / 1000, left / 1000])

            # motion delta
            motion_update = np.array([x - old_x, y - old_y, yaw - old_yaw])

            slam_states.append (
                slam_agent.update_state (
                    ranges,
                    scan_angles,
                    motion_update
                )
            )

            # Update old values
            old_x = x
            old_y = y
            old_yaw = yaw
```

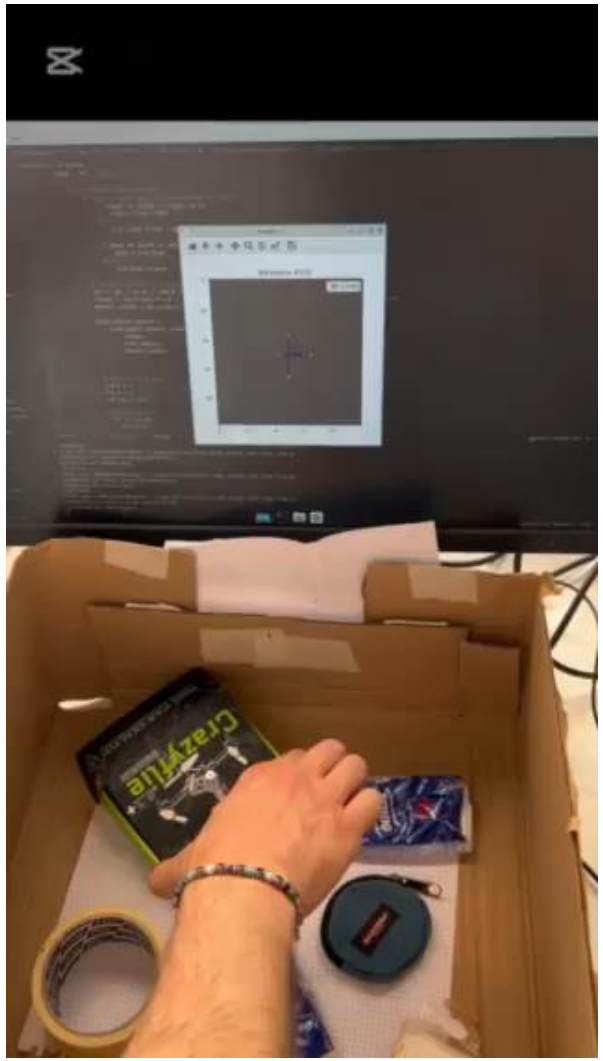
Mapping e Visualizzazione SLAM

- Conversione delle coordinate del drone in coordinate mappa
- Aggiornamento della mappa di occupazione durante l'esecuzione
- Utilizzo della funzione `discretize` per localizzazione su griglia

- Visualizzazione della mappa aggiornata ogni *skip* iterazioni
- Posizione stimata del drone mostrata come punto rosso
- Freccia blu per indicare la direzione del drone
- Interfaccia live con Matplotlib

```
# Visualization
if k % skip == 0:
    plt.clf()
    slam_map = slam_agent.map

    plt.imshow(slam_map, cmap="gray")
    idx_slam = discretize(slam_states[k][:2], slam_agent.params)
    px, py = idx_slam

    plt.plot(py, px, "ro", label="slam")
    dx = arrow_length * np.cos(yaw)
    dy = arrow_length * np.sin(yaw)
    plt.arrow(py, px, dx, dy, head_width=3, head_length=3, fc='blue', ec='blue')

    plt.title(f"Iterazione #{k}")
    plt.legend()
    plt.draw()
    plt.pause(0.01)
    # plt.show(block=False)
k += 1
```

Real Time



Asincrono