

# Progetto Robotica e Intelligenza Artificiale 2 Rescuerbot



**Componenti del Gruppo:**

**Bongiorno Rosario**  
**Giambanco Alessandro**  
**Barca Antonio**



**Università  
degli Studi  
di Palermo**



**Webots**  
robot simulation

# Sommario

1. Introduzione.....	3
1.1. Descrizione Generale.....	3
1.2. Scopo del Progetto.....	3
1.3. End-Users.....	3
2. Progettazione del Sistema.....	4
2.1. Analisi dei Requisiti.....	4
2.1.1. Requisiti Funzionali.....	4
2.1.2. Requisiti Non Funzionali.....	4
2.1.3. Requisiti Software.....	5
2.2. Descrizione delle componenti del sistema.....	6
2.2.1. Architettura del robot.....	6
2.2.2. Stati Emozionali.....	8
2.2.3. Mappa.....	9
2.3. Diagramma FSM.....	10
3. Algoritmi.....	11
3.1. Mapping.....	11
3.1.1. Ricerca Oggetti stato Felice.....	14
3.1.2. Ricerca Oggetti stato Normale.....	15
3.1.3. Ricerca Oggetti stato Triste.....	16
3.2. Collect.....	17
4. Architettura del Programma.....	19
4.1. Rescue_controller.py.....	19
4.2. Salvataggio.py.....	20
4.3. Movement.py.....	21
4.4. Mapping.py.....	25
4.5. Collect.py.....	27
4.6. Cam.py.....	29
4.7. Particle_filter.py.....	30
4.8. TTS.py.....	32
5. Simulazione.....	33
5.1. Mapping.....	33
5.2. Collect.....	33

# 1. Introduzione

## 1.1. Descrizione Generale

Il nostro Progetto consiste nella realizzazione di un robot capace di muoversi in un ambiente post-terremoto, dove per un umano sarebbe pericoloso entrare a causa della struttura pericolante.

Il robot è in grado di mappare l'area, individuare e comunicare la presenza di umani, riconoscere e successivamente raccogliere oggetti smarriti.

Abbiamo realizzato un sistema che simula delle emozioni che prova il robot, esse si manifestano durante la mappatura, dove in base al numero di oggetti trovati il robot risulta felice o triste, e quindi ciò comporta che se si trova nello stato Triste il robot ha precedentemente trovato pochi oggetti quindi sarà più attento, mentre nello stato Felice avrà trovato più oggetti o umani, quindi sarà meno attento con il rischio di tralasciarne qualcuno.

Come ulteriore funzionalità è stato implementato un filtro particolare che permette, nei casi di slittamento a causa di scosse o altri eventi, al robot di rilocalizzarsi.

Inoltre è stata implementata la capacità di riconoscere, dopo aver effettuato la mappatura, eventuali incongruenze tra l'ambiente rilevato e quello descritto dalla mappa, come ad esempio la caduta di detriti dal soffitto.

L'implementazione è stata effettuata mediante il software di simulazione Webots che ci ha permesso di realizzare un ambiente domestico nel quale oltre agli arredi sono presenti rocce e detriti causati dal terremoto.

## 1.2. Scopo del Progetto

Lo scopo principale del progetto è quello di agevolare l'uomo in situazioni che potrebbero metterlo in pericolo, permettendo al robot di esplorare l'area colpita dal terremoto e individuare eventuali umani agevolandone il soccorso.

## 1.3. End-Users

Gli utilizzatori finali del robot saranno tutti quei gruppi impegnati nel soccorso e nella ricerca nei casi di calamità, come Vigili del Fuoco, Protezione Civile ecc.

## 2. Progettazione del Sistema

### 2.1. Analisi dei Requisiti

#### 2.1.1. Requisiti Funzionali

- Movimento: Il robot è in grado di muoversi in avanti e indietro e ruotare in direzione dei 4 punti cardinali Nord Sud Est Ovest
- Misurazioni: Il robot dispone di 4 sensori lidar che gli permettono di misurare la distanza dagli oggetti nelle 4 direzioni, sensori di odometria posti in corrispondenza delle due ruote, sensore inerziale che gli permette di conoscere la propria direzione.
- Mappatura: il robot è in grado di mappare ogni ambiente in cui viene posto riconoscendo gli spazi liberi e quelli occupati.
- Riconoscimento: Il robot è in grado di riconoscere gli oggetti che si trovano davanti a lui e distingue quali sono importanti al fine della ricerca e quali no.
- Recupero: il robot è in grado di recuperare oggetti sparsi nella stanza e consegnarli in un punto di recupero, questo avviene mediante una pinza di cui è dotato.
- Localizzazione: il robot è in grado di localizzarsi mediante il sensore di odometria e recuperare la propria posizione in caso di eventi che potrebbero provocare delle traslazioni improvvise grazie ad un filtro particellare.
- Individuazione caduta rocce: il robot è in grado, in fase di raccolta, di individuare nuove rocce cadute non presenti nella fase di mappatura.
- Illuminazione: il robot dispone di una luce che illumina gli ambienti e gli permette il riconoscimento anche in situazioni di scarsa illuminazione.
- Text to Speach: il robot è in grado di comunicare vocalmente il proprio stato emotivo, eventuali riconoscimenti di oggetti o umani e il loro avvenuto recupero.
- Emozioni: il robot è in grado di simulare degli stati emotivi che gli permettono di applicare più attenzione nella ricerca quando non ha trovato oggetti/umani e di essere più veloce dopo averne trovati.

#### 2.1.2. Requisiti Non Funzionali

- Affidabilità: il robot risulta affidabile in quanto dispone di un secondo sistema di localizzazione dato dal filtro particellare
- Prestazioni: il robot deve disporre di sensori lidar al fine di una più accurata lettura della distanza tra robot e oggetti, evitando i problemi di affidabilità e di letture errate legati ai sensori sonar
- Usabilità: il robot necessita di uno speaker al fine di una maggiore interazione con l'esterno

### 2.1.3. Requisiti Software

- Interprete Python: il quale permette la traduzione e compilazione del codice scritto in linguaggio Python
- Librerie esterne:
  - numpy
  - math
  - random
  - pathfinding: importiamo principalmente Grid e AStarFinder , il quale implementa l'algoritmo di ricerca A\*
  - matplotlib.pyplot

## 2.2. Descrizione delle componenti del sistema

### 2.2.1. Architettura del robot

Abbiamo scelto per la nostra implementazione il Pioneer 3-DX di Adept; esso è un robot multiuso, utilizzato per la ricerca e applicazioni che coinvolgono mappatura, localizzazione, monitoraggio, ricognizione e altri comportamenti. La piattaforma base Pioneer 3-DX è assemblata con motori dotati di encoder da 500 tick, ruote da 19 cm, robusto corpo in alluminio, 8 sensori a ultrasuoni (sonar) rivolti in avanti, 8 sonar posteriori opzionali.

Il robot può raggiungere velocità di 1,6 metri al secondo e trasportare un carico utile fino a 23 kg.

Nella nostra realizzazione non abbiamo utilizzato i sonar già installati nel Pioneer, ma per una maggiore attendibilità dei dati abbiamo scelto di usare 4 sensori lidar posti ai 4 punti cardinali del robot, che avendo un'apertura conica riescono a coprire Nord Sud Est e Ovest.

Per misurare gli spostamenti del robot abbiamo inserito due sensori di odometria posti in corrispondenza delle due ruote.



Inoltre abbiamo aggiunto una camera grandangolare posta nella parte superiore del robot che ci permette di riconoscere gli oggetti.

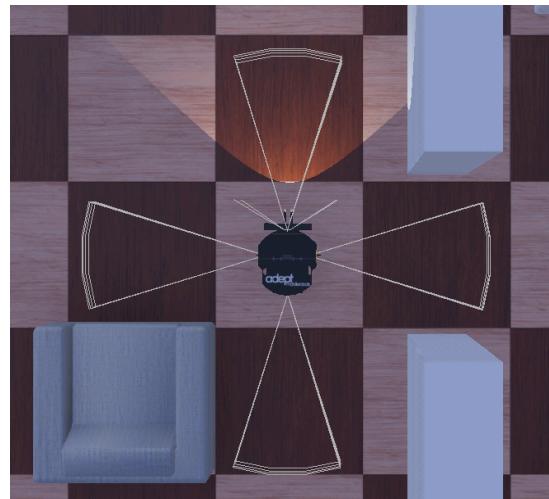
Nella parte superiore abbiamo aggiunto anche una luce che ci permette di illuminare gli ambienti.



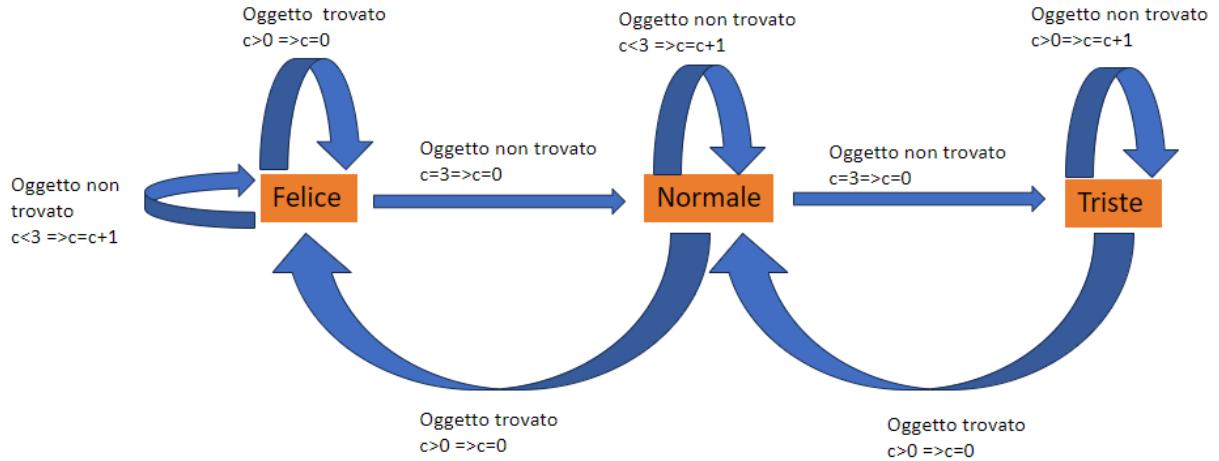
Frontalmente abbiamo infine aggiunto l'accessorio Pioneer 3D Gripper che ci permette di sollevare e agganciare oggetti. Dopo alcuni test abbiamo deciso inoltre di inserire una piccola zavorra posteriore per bilanciare il peso del gripper.



Altri dispositivi che abbiamo inserito nel robot sono il sensore inerziale che ci fornisce l'angolo di rotazione attorno all'asse z del nostro robot (Yaw) ed uno speaker che permette al robot di riprodurre vocalmente dei testi.



## 2.2.2. Stati Emozionali



Abbiamo deciso di implementare tre stati emozionali nel robot, legati alla fase di ricerca e mappatura degli oggetti e degli umani.

In ognuno di questi stati il robot cambia approccio alla ricerca aumentando o meno la precisione; il cambio di stato, analizzato nella FSM precedente, avviene nei casi in cui il robot individua un oggetto/umano o quando per più di 3 celle non ne trova.

Di seguito vengono analizzati gli stati:

- **Triste:** in questo stato il robot effettua scansioni con la camera in tutte le direzioni per cercare eventuali oggetti anche in posizioni più lontane; se trova un oggetto passa allo stato Normale
- **Normale:** in questo stato il robot effettua scansioni con la camera esclusivamente nelle celle dove mediante il sensore di distanza ha individuato qualcosa; se trova un oggetto passa allo stato felice, se per più di tre celle non trova oggetti passa allo stato Triste
- **Felice:** in questo stato il robot effettua scansioni con la camera, solo in una porzione casuale di celle tra quelle dove mediante il sensore di distanza ha individuato qualcosa; se per più di tre celle non trova oggetti passa allo stato Normale

### 2.2.3. Mappa

Per la realizzazione dell'ambiente, abbiamo deciso di simulare uno scenario domestico terremotato composto da due stanze, un soggiorno e una cucina, dove possiamo notare rocce e mobili caduti a seguito del terremoto.



Abbiamo deciso di dividere l'ambiente con una serie di celle di dimensioni 1m x 1m e rappresentarlo con una matrice dove ogni posizione corrisponde ad una cella, anche i muri sono inclusi nella mappa.

La rappresentazione degli elementi nella mappa segue la seguente legenda:

- -1 cella non ancora visitata,
- 0 cella libera,
- 1 cella occupata da oggetto sconosciuto,
- 2 cella occupata da umano,
- 3 cella occupata da un oggetto da recuperare,
- 4 cella occupata da muro o oggetto da non recuperare;

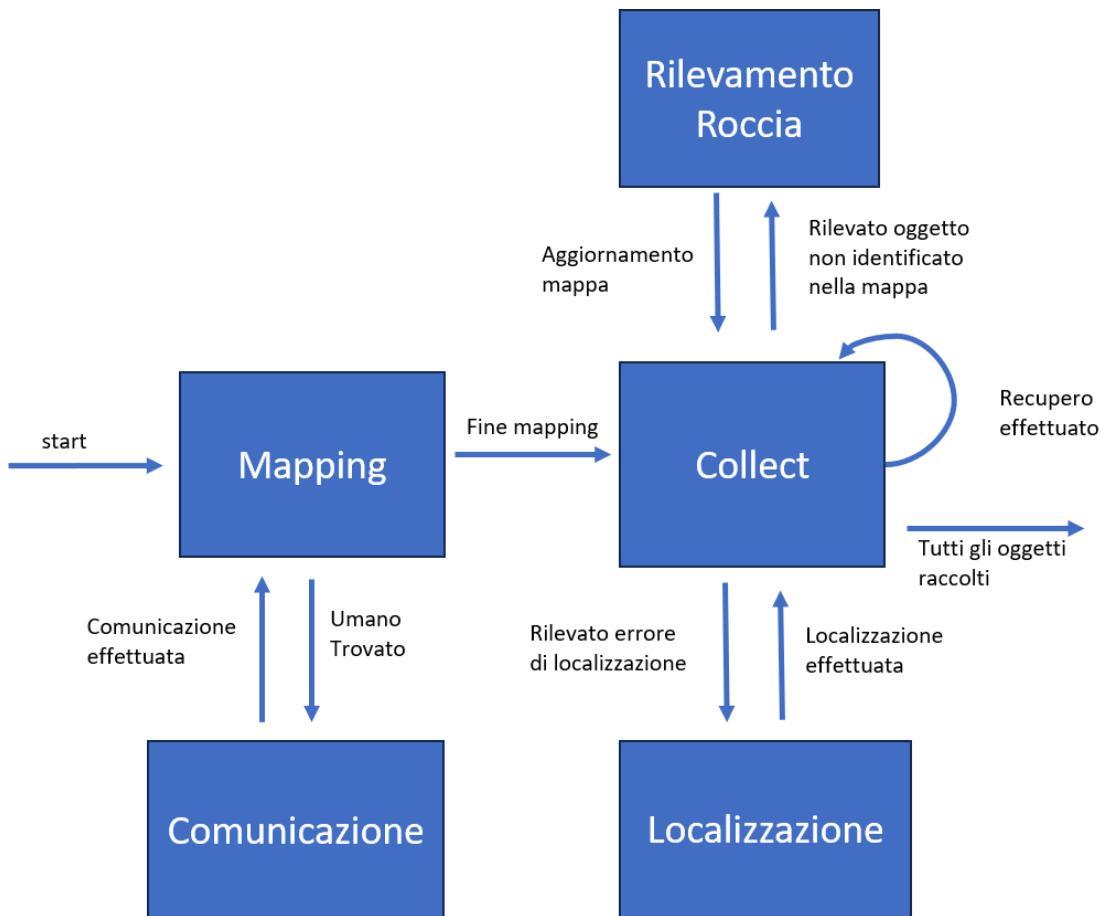
Di seguito una possibile rappresentazione della mappa popolata da umani e oggetti da recuperare:



Di seguito la rappresentazione matriciale:

```
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4],  
[4, 0, 3, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4],  
[4, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4],  
[4, 4, 3, 4, 4, 0, 0, 0, 0, 0, 4, 4, 2, 4, 3, 4],  
[4, 0, 4, 4, 0, 0, 4, 0, 4, 0, 0, 0, 4, 2, 0, 4, 4],  
[4, 0, 0, 0, 0, 2, 2, 0, 4, 4, 0, 0, 3, 4, 0, 4, 4],  
[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]])
```

### 2.3. Diagramma FSM



Il diagramma rappresenta una macchina a stati finiti delle principali funzionalità di cui dispone il Rescuerbot.

Il Rescuerbot inizia con la fase di mappatura dell'ambiente circostante ignoto, qualora trovi un umano si dirige immediatamente su di esso al fine di comunicare tempestivamente la posizione dell'umano per il recupero da parte dei soccorsi, successivamente continua a mappare l'area per ricercare altri eventuali oggetti o umani.

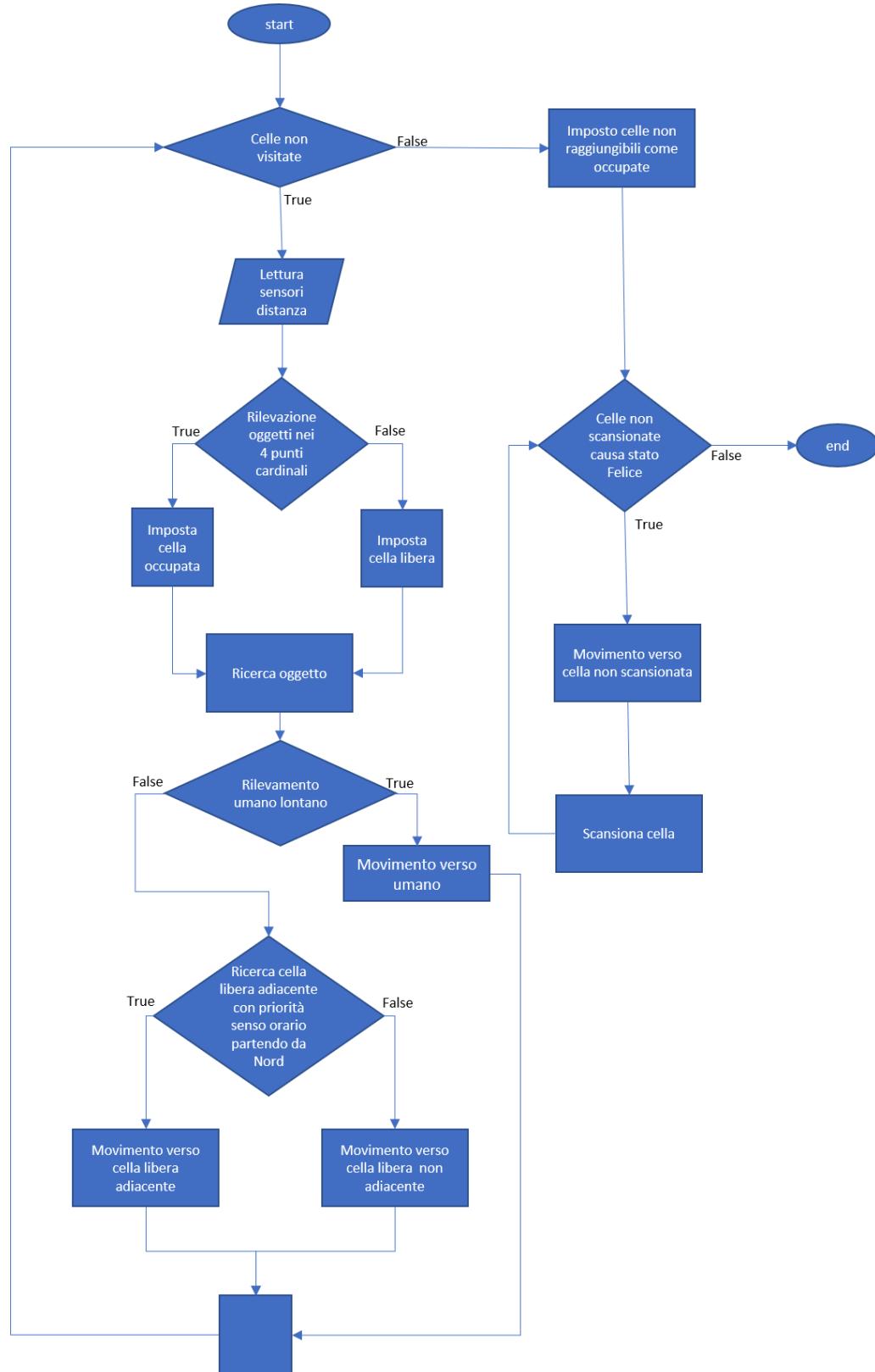
Al termine della fase di mappatura il robot inizia la fase di recupero degli oggetti trovati precedentemente; nel caso in cui il robot rileva incongruenze tra ciò che percepisce con i sensori e la mappa valuta se ciò è dovuto ad una scossa di terremoto che porterebbe il Rescuerbot a traslare involontariamente, e quindi avvia l'algoritmo di localizzazione; oppure se si tratta di un nuovo elemento comparso nell'ambiente, come ad esempio la caduta di una roccia, quindi aggiorna la mappa; una volta recuperati tutti gli oggetti il Rescuerbot avrà completato la sua missione.

### 3. Algoritmi

In seguito andremo a descrivere i due algoritmi principali del programma che abbiamo distinto in Mapping e Collect, il primo riguarda la mappatura e la ricerca degli oggetti e degli umani e il secondo riguarda la loro raccolta.

#### 3.1. Mapping

Di seguito il diagramma di flusso dell'algoritmo mapping:



Per effettuare la mappatura abbiamo deciso di creare due matrici della dimensione dell'area da esplorare che rappresentano rispettivamente la mappa della reale disposizione degli elementi e la mappa delle celle visitate, quest'ultima sarà una matrice booleana dove imposteremo True se il robot è realmente passato da quella cella e false altrimenti.

Inizialmente impostiamo come libera la cella iniziale da cui parte il robot nella mappa principale e settiamo a TRUE la stessa posizione nella mappa delle celle visitate.

Successivamente eseguiamo un ciclo che si ripete finché tutte le celle non sono state esplorate.

Ad ogni ciclo l'algoritmo di mapping si può dividere in 3 sezioni principali:

#### 1° PARTE: Scansione delle celle mediante i sensori di distanza

Tramite i sensori di distanza, analizziamo le celle adiacenti alla nostra posizione attuale poste ai 4 punti cardinali e assegniamo 0 per celle libere o 1 per cella con oggetto ancora non identificato.

#### 2° PARTE: Scansione delle celle mediante la camera

Successivamente effettuiamo la ricerca degli oggetti che è la parte fondamentale del mapping, perché è qui che riusciamo a riconoscere gli oggetti o gli umani.

Questa fase sarà differente a seconda dello stato emotivo del robot, per questo abbiamo deciso di effettuare un diagramma di flusso diverso per ogni stato e che troveremo in seguito.

In particolare:

- Stato Normale: il robot inizialmente controlla lo stato delle celle vicine, se almeno una di esse è occupata da qualche oggetto, allora effettua la scansione di quelle celle;
- Stato Triste: il robot effettua la scansione nelle 4 celle vicine, a prescindere dal loro valore, così da cercare umani anche in posizioni più distanti;
- Stato Felice: il robot controlla lo stato delle celle vicine ed effettua la scansione esclusivamente in una porzione casuale delle celle in cui ha rilevato oggetti;

Il cambio da uno stato all'altro avviene secondo le seguenti regole:

- Da Triste a Normale, se è riuscito a trovare un oggetto e vi rimane così per almeno 3 celle anche se non trova oggetti;
- Da Normale a Felice, se è riuscito a trovare un oggetto e vi rimane così per almeno 3 celle anche se non trova oggetti;
- Da Felice a Normale, se non è riuscito a trovare un oggetto e vi rimane così per almeno 3 celle anche se non trova oggetti;
- Da Normale a Triste, se non è riuscito a trovare nessun oggetto.

### 3° PARTE: Movimento verso la prossima cella

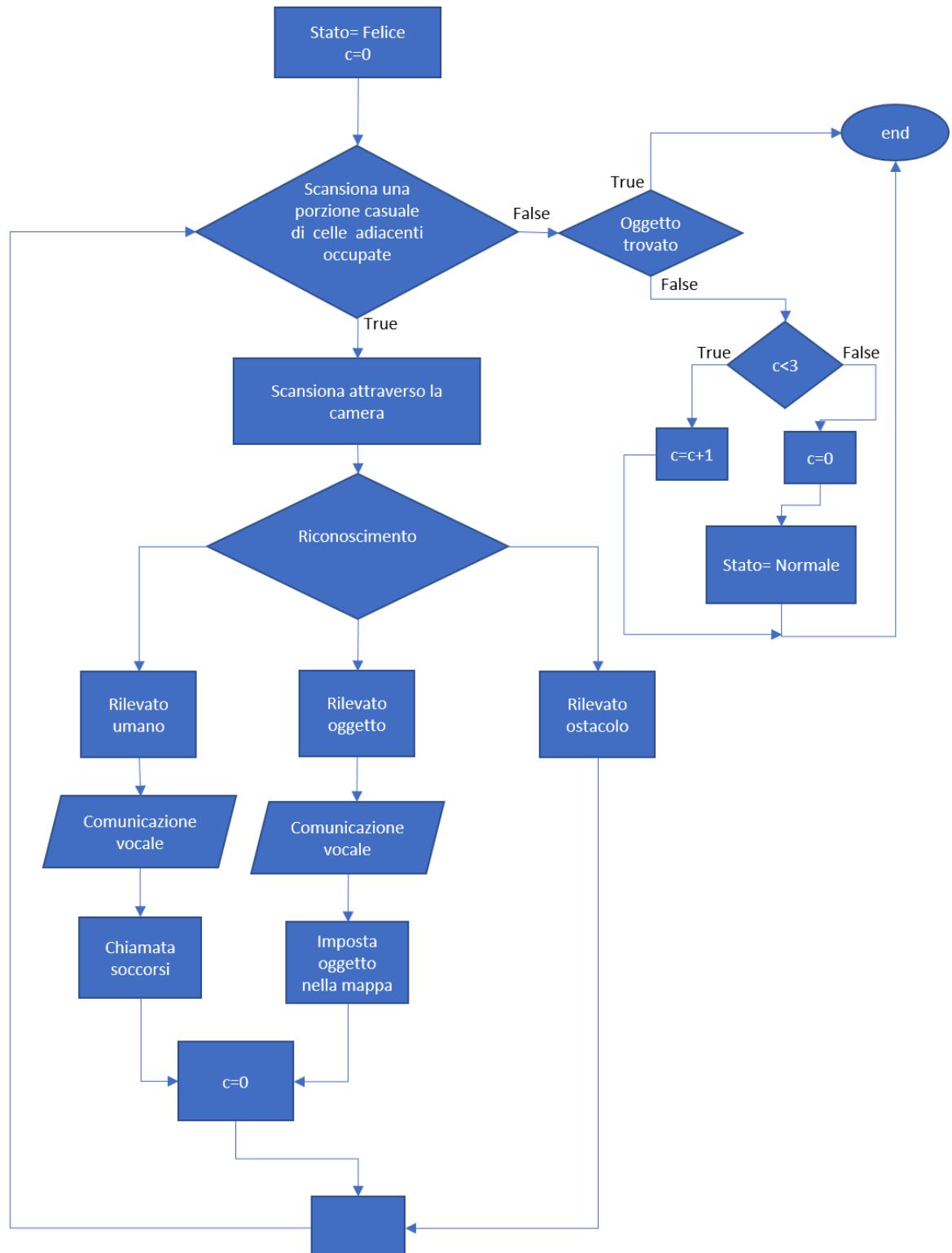
Come detto precedentemente una particolarità del robot allo stato Triste è che se esso, effettuando la scansione, intravede un umano ad una distanza maggiore rispetto alla cella adiacente, una volta ultimata la scansione della cella si dirige immediatamente verso l'umano per poterlo portare in salvo, altrimenti si muove verso la prima cella adiacente libera non visitata in senso orario, partendo da nord.

Se anche tutte queste celle fossero già state visitate calcola il percorso per raggiungere la più vicina cella libera non visitata e continuare la mappatura da lì.

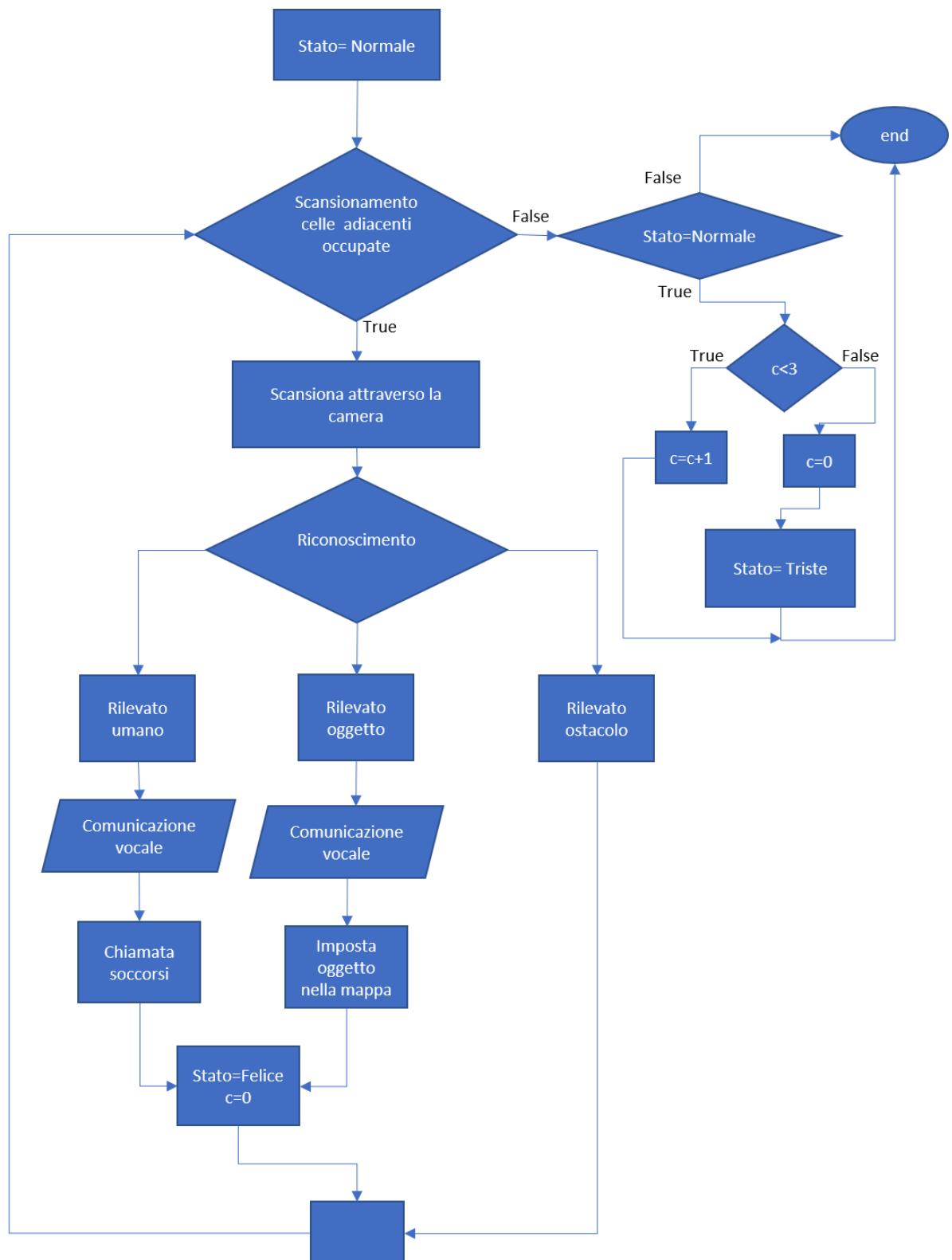
Successivamente dopo aver visitato tutte le celle il robot controlla se sono rimaste celle non visitate perché irraggiungibili, cioè tutte quelle posizioni alle quali il robot non può accedere a causa di altri oggetti che ne bloccano il passaggio, e quindi le imposta come celle occupate.

Infine il robot controlla che non siano rimaste celle che contengono oggetti ancora da scansionare che potrebbe aver saltato durante lo stato Felice, quindi le raggiunge e le scansiona.

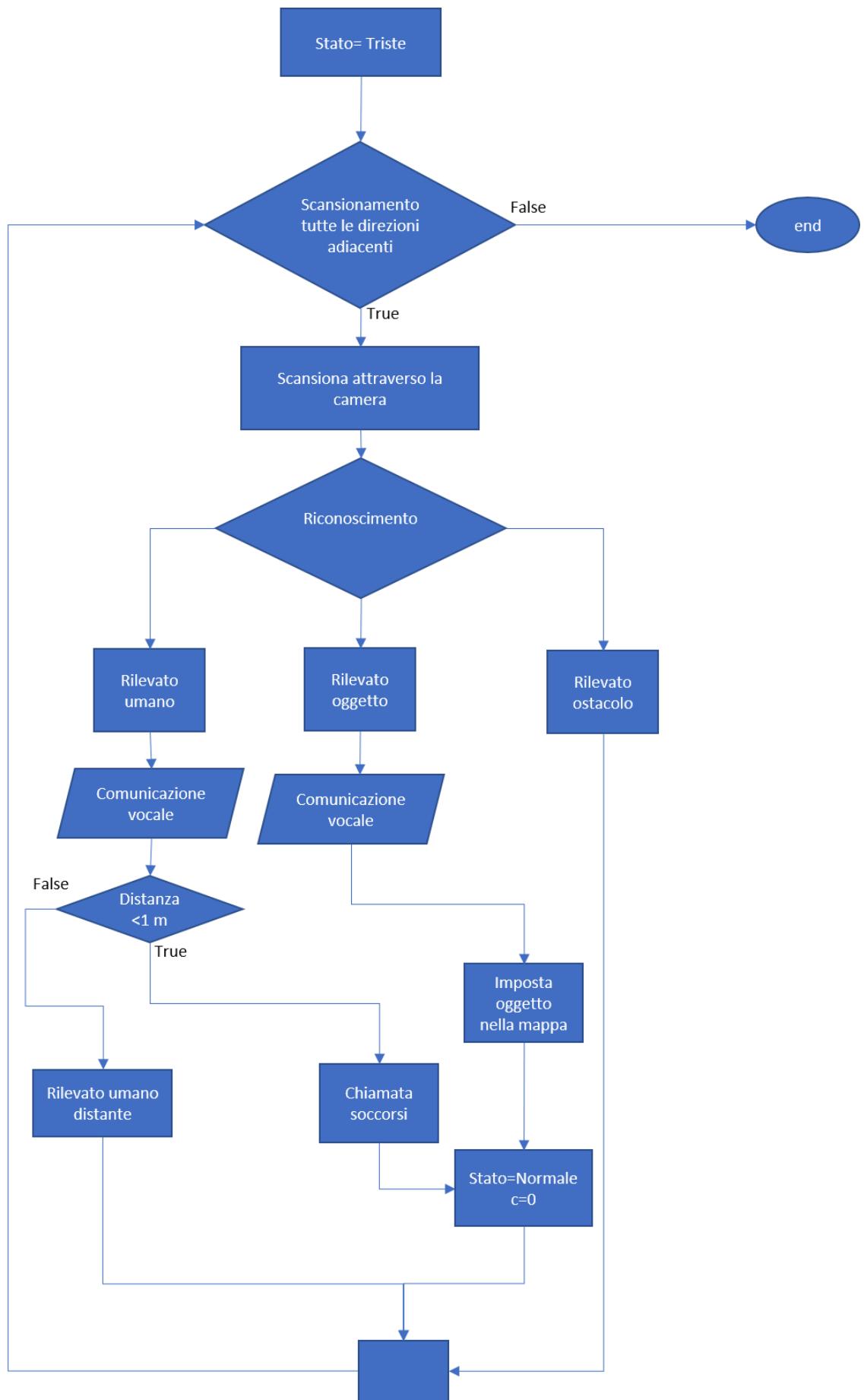
### 3.1.1. Ricerca Oggetti stato Felice



### 3.1.2. Ricerca Oggetti stato Normale

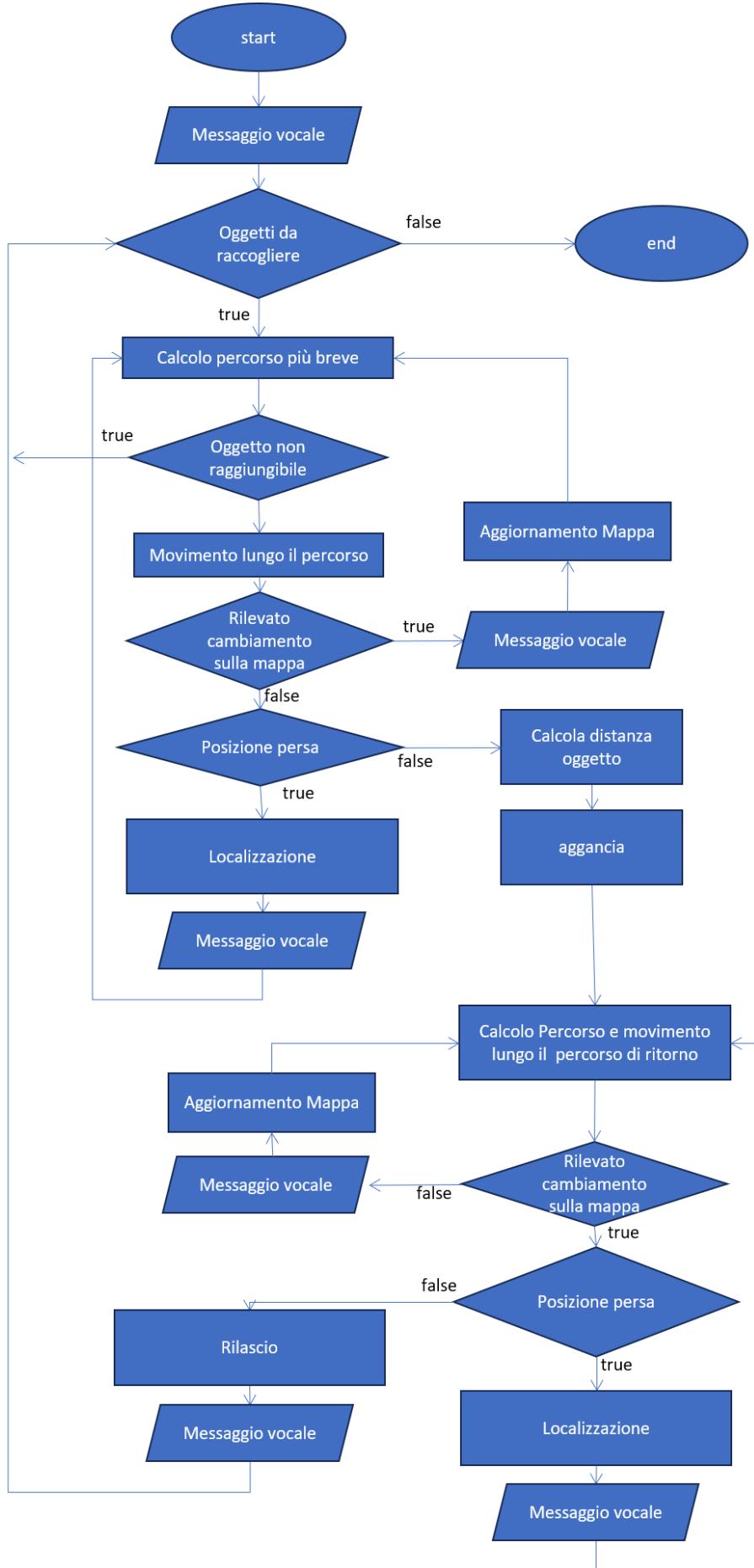


### 3.1.3. Ricerca Oggetti stato Triste



### 3.2. Collect

Di seguito il diagramma di flusso dell'algoritmo collect



Per effettuare il recupero degli oggetti iniziamo comunicando vocalmente l'inizio del processo di raccolta, per ogni oggetto da raccogliere il robot calcola il percorso e si muove lungo di esso.

In ogni cella controlla se vi sono incongruenze tra ciò che rilevano i sensori e la mappa, in questo caso determina se vi è stato un cambiamento nella mappa, ad esempio la caduta di una roccia il robot aggiorna la mappa e ricalcola il percorso altrimenti se vi è stato uno slittamento o una scossa derivata dal terremoto il robot grazie al filtro particolare ed una serie di movimenti si riesce a rilocalizzare e quindi calcola nuovamente il percorso al fine di raggiungere l'oggetto.

Arrivato a destinazione, il robot calcola la distanza dall'oggetto da recuperare, si assicura di essere nella posizione ottimale per raccogliere l'oggetto, altrimenti si raddrizza, e procede al recupero.

Successivamente ricalcola il percorso di ritorno e consegna l'oggetto presso il punto di raccolta valutando eventuali cambiamenti nella mappa come precedentemente descritto.

## 4. Architettura del Programma

Per l'implementazione del robot ci siamo serviti del controller `Rescue_controller.py` che gestisce le operazioni del robot e di un secondo controller `Salvataggio.py` che funge da supervisor.

Per le operazioni di Mappatura, Movimento, Raccolta e Localizzazione abbiamo deciso di implementare la creazione delle seguenti classi:

- `Movement.py`
- `Mapping.py`
- `Collect.py`
- `Cam.py`
- `Particle_filter.py`
- `TTS.py`

### 4.1. `Rescue_controller.py`

Questo script rappresenta il main della nostra simulazione. Vengono infatti inizializzate tutte le variabili essenziali, quali:

- `TIME_STEP`: passo temporale per la simulazione
- `robot`: istanza della classe `Robot`
- `pos_start`: posizione iniziale del robot
- `dim_map`: dimensioni della mappa

All'interno del ciclo di controllo principale per la simulazione, viene effettuata la creazione delle istanze di varie classi: `TTS`, `Cam`, `Particle_filter`, `Movement`, `Mapping` e `Collect` con i parametri appropriati. Successivamente è presente un ciclo di controllo principale che viene eseguito finché non viene eseguita tutta la simulazione, ed all'interno di esso viene eseguita la funzione `mapping()` per aggiornare la mappa e la funzione `start_collect()` per avviare la raccolta di oggetti.

## 4.2. Salvataggio.py

Questo file python ha lo scopo di implementare il Supervisor dentro un oggetto robot di webots, il cui compito è gestire la simulazione per quanto riguarda l'aggiunta degli umani da salvare ,l'aggiunta degli oggetti da recuperare e conseguentemente il soccorso dell'umano ed il recupero degli oggetti.

Descrizione Supervisor:

Ogni nodo robot in webots può essere definito come Supervisor, questo comporta il poter gestire i nodi della simulazione avendo accesso ai vari campi di ogni nodo al fine di aggiungere/modificare i campi, eliminare i nodi etc..

In particolare consta dei seguenti metodi :

- **aggiungi\_umano(traslation,rotation,robot,name):** metodo che permette l'aggiunta di un nodo Pedestrian ,il cui proto abbiamo modificato al fine del riconoscimento della camera avendo aggiunto un recognition colors, e la traslazione e rotazione verso un punto della mappa desiderato
- **aggiungi\_box(traslation,color\_rec,robot,name,nameproto):** metodo che permette l'aggiunta di vari nodi, i cui proto abbiamo modificato al fine del riconoscimento della camera avendo aggiunto per ogni tipo di oggetto da recuperare un recognition colors univoco, e la traslazione verso un punto della mappa desiderato
- **aggiungi\_rock(traslation, robot, name):** metodo che permette l'aggiunta di un nodo Rock, aumentando i fattori di 'translation' e 'scale'; se l'aggiunta di tale roccia si troverà nel percorso del robot, causerà al Rescuerbot il ricalcolamento del percorso per raggiungere un oggetto di raccolta
- **recupera\_oggetti(objs):** metodo che permette il recupero degli oggetti valutando se la posizione attuale di ogni oggetto sia il punto di recupero designato ( nel nostro caso l'angolo in basso a sinistra della prima stanza)
- **salvataggio\_umano(rescuer):** metodo che simula il salvataggio di un umano ,in particolare una volta che il robot ha individuato l'umano e lo trova ad una distanza minore di 1m verrà impostato il campo customData del Rescuerbot (dentro il metodo rimuovi\_umano della classe Mapping), che conterrà la posizione dell'umano da salvare con la conseguente eliminazione del nodo umano corrispondente

### 4.3. Movement.py

La classe Movement gestisce tutto ciò che riguarda i movimenti del robot, la ricerca dei percorsi più brevi per raggiungere gli oggetti, la gestione di nuovi ostacoli rilevati durante il percorso e i sensori che permettono tutto ciò.

Per la ricerca dei percorsi è stato utilizzato l'algoritmo A\*, un particolare tipo di ricerca best-first, implementato dalla libreria pathfinding; esso utilizza una stima euristica che classifica ogni nodo attraverso una stima del percorso migliore che passa attraverso tale nodo, nella nostra implementazione,i nodi sono rappresentati dalle celle e il costo del percorso tra ogni nodo è unitario.

Al costruttore della classe passiamo in input il Timestep, l'oggetto robot, le coordinate di partenza e il Particle Filter; essi verranno salvati come attributi della classe. Inoltre vengono attivati i motori e i sensori del robot. Come altri attributi troviamo:

- leftMotor: rappresenta il motore sinistro del robot
- rightMotor: rappresenta il motore destro del robot
- left\_ps: rappresenta il sensore di posizione della ruota sinistra
- right\_ps: rappresenta il sensore di posizione della ruota destra
- lidar\_front: rappresenta il sensore lidar posto davanti al robot
- lidar\_back: rappresenta il sensore lidar posto dietro al robot
- lidar\_dx: rappresenta il sensore lidar posto a destra del robot
- lidar\_sx: rappresenta il sensore lidar posto a sinistra del robot
- imu: rappresenta il sensore inerziale del robot
- MAX\_SPEED: rappresenta la velocità massima a cui si muoverà il robot
- ps\_values: lista a cui vengono assegnati i valori del sensore di odometria
- dist\_values: lista a cui vengono assegnati i valori del sensore di odometria convertiti in metri
- raggio\_ruota: rappresenta il raggio delle ruote del robot espresso in metri
- circonferenza\_ruota: rappresenta il valore della circonferenza della ruota espressa in metri
- enc\_unit: rappresenta il valore moltiplicativo per convertire i radiantini dati dal sensore di odometria in metri
- robot\_pose: lista a cui vengono assegnati i valori delle coordinate a cui si trova il robot e la direzione a cui punta
- lidar\_value: lista a cui vengono assegnati i valori del sensore lidar di distanza

Di seguito verranno analizzati i vari metodi:

- **lidarsensor()**: metodo che permette l'aggiornamento dei sensori di distanza
- **odo()**: metodo che permette l'aggiornamento dei sensori di posizione
- **direction()**: metodo che tramite il sensore imu valuta la direzione del robot
- **obj\_dir(x,y)**: metodo che permette al robot di ruotare in direzione della casella di coordinate x,y
- **obj\_front\_pose()**: metodo che restituisce le coordinate della casella che si trova davanti a lui, in base alla propria rotazione
- **get\_opposite\_dir(dir)**: metodo che ritorna la direzione opposta a quella passata in input
- **robot\_update(dist)**: metodo che permette l'aggiornamento della variabile robot\_pose e che viene invocato ad ogni movimento del robot, se dist=0 significa che il robot ha effettuato una rotazione quindi aggiorniamo la direzione, altrimenti il robot si è mosso in avanti quindi aggiorniamo la posizione
- **lidar\_permutation()**: metodo che ritorna una lista contenente i valori dei sensori di distanza sempre nell'ordine [Nord,Sud,East,Ovest] quindi in base alla direzione in cui punta il robot effettua una permutazione della lista lidar\_value
- **layer\_reattivo(delta)**: metodo che ritorna False se il sensore di distanza posto davanti al robot rivela un oggetto alla distanza delta passata come input e True altrimenti
- **stop\_motors()**: metodo che imposta la velocità dei motori a 0 quindi effettua lo stop del robot
- **move(dist)**: metodo che permette al robot di muoversi in avanti. Ad ogni step del robot viene valutata la distanza percorsa mediante il sensore di odometria e quando risulta minore o uguale al valore di

distanza passato in input viene fermato il robot, inoltre ogni step viene invocata anche la funzione layer reattivo con un delta a 0.3m che ci permette di fermarci alla giusta distanza dagli oggetti e quindi di ovviare a piccoli errori dati dal sistema di odometria.

- **move\_back(dist):** metodo che permette al robot di muoversi indietro di un valore pari a dist, funziona allo stesso modo di move() ma la velocità viene impostata negativa
- **rotate(dir):** metodo che permette la rotazione del robot verso la direzione indicata in input e utilizza un sensore IMU per controllare l'angolo di rotazione. In base alla direzione corrente del robot e alla direzione desiderata, imposta le velocità dei motori per eseguire una rotazione corretta. Il ciclo di controllo continua finché l'angolo desiderato non viene raggiunto. Dopo la rotazione, la direzione del robot viene aggiornata. La precisione dell'angolo di rotazione è controllata attraverso la variazione dell'angolo ottenuto dal sensore IMU.
- **follow\_path(path):** metodo che permette la guida del robot lungo un percorso specificato, rappresentato da una lista di coordinate (x, y). Per ogni coppia di coordinate successiva nel percorso, il robot si orienta nella direzione corretta (Est, Ovest, Nord o Sud) e si sposta avanti di un metro.
- **find\_path\_obj(map,x,y):** metodo che cerca un percorso sulla mappa tra la posizione corrente del robot e una posizione obiettivo passata come input. Viene utilizzato il metodo AStarFinder della libreria pathfinding che considera True le posizioni libere e False quelle occupate, quindi viene invertita la mappa e convertita in booleani. Infine viene restituito il percorso calcolato escludendo la posizione finale cioè quella in cui è posizionato l'oggetto da raggiungere.
- **find\_path\_min(free\_cells,x,y,map):** metodo simile a find\_path\_obj ma cerca il percorso più breve tra una posizione di partenza specificata e un insieme di celle libere 'free\_cells'
- **position\_recalculation():** metodo che permette di ricalcolare la posizione del robot mediante il filtro particellare, viene guidato il robot in un movimento casuale attraverso l'ambiente. Utilizza il lidar per evitare collisioni e mantiene una lista delle posizioni visitate. Il ciclo continua finché il robot visita almeno 5 celle diverse, per permettere al filtro particellare di effettuare misurazioni diverse al fine di stimare la posizione corretta.
- **agg\_filtro():** metodo che gestisce l'aggiornamento della posizione stimata del robot quando viene rilevata una posizione non corretta a seguito di uno slittamento o di una scossa di terremoto. Effettua una

ridistribuzione delle particelle nel filtro particellare tramite il metodo `redistribution()` della classe Particle filter, esegue movimenti casuali per ricalcolare la posizione tramite il metodo `position_recalculation()`, e infine aggiorna la posizione e la direzione del robot.

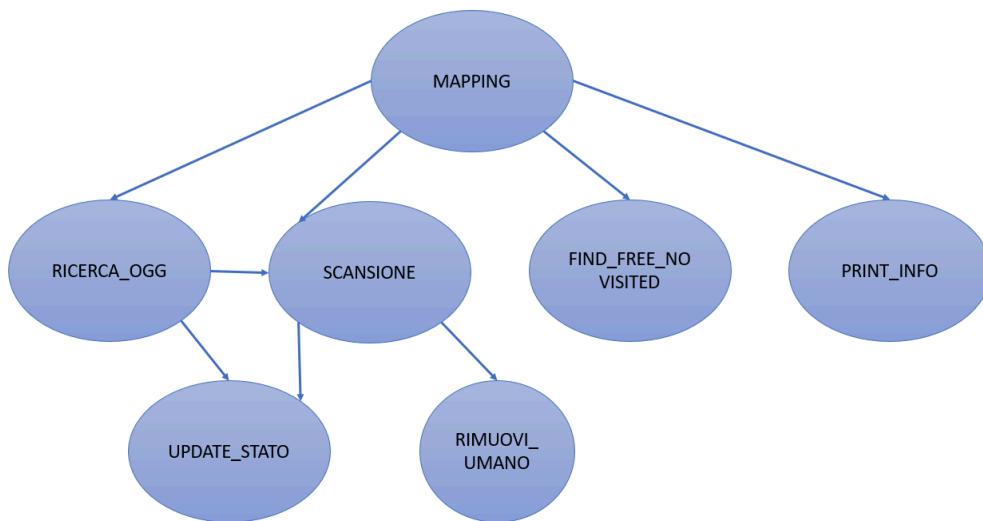
- **rock\_falling():** metodo che analizza le celle circostanti al Rescuerbot e permette di comprendere se il robot ha individuato una nuova roccia caduta oppure se ha effettuato una traslazione involontaria; in particolare qualora non dovesse trovare corrispondenza tra valori sensoristici dei lidar rispetto le celle circostanti e rispetto la mappa memorizzata, il Rescuerbot si muoverà di 3 celle non visitate al fine di accertarsi se l'oggetto individuato non era presente precedentemente e memorizzarlo nella mappa.
- **follow\_path\_filtered(path,map):** metodo che permette la guida del robot lungo un percorso specificato, rappresentato da una lista di coordinate (x, y). Per ogni coppia di coordinate successiva nel percorso, il robot si orienta nella direzione corretta (Est, Ovest, Nord o Sud) e si sposta avanti di un metro. Durante il movimento, al fine di stimare se qualcosa è andato storto e quindi se in una data cella viene rilevata una roccia non identificata precedentemente invoca il metodo `rock_falling` per accertarsi dell'accaduto e si rilocalizza utilizzando il filtro particellare qualora non avesse riscontrato una nuova roccia ma semplicemente una traslazione involontaria.

## 4.4. Mapping.py

La classe Mapping corrisponde alla classe relativa alla mappatura, che consente quindi al robot di riconoscere l'ambiente che lo circonda andando a controllare in particolar modo se (e dove) siano presenti spazi liberi, ostacoli , umani da salvare ed oggetti da recuperare . Tali operazioni sono inoltre condizionate da uno stato emotivo del robot che risulterà essere “Triste”, “Normale” o “Felice”.

Le variabili della classe Mapping sono le seguenti:

- movement: un oggetto della classe Movement;
- cam: un oggetto della classe Cam;
- map: matrice intera numpy che rappresenta l'occupazione delle celle ;
- visited: matrice booleana numpy che rappresenta la visita delle celle;
- width, length: dimensioni di larghezza e lunghezza della mappa;
- x\_start , y\_start : posizioni di partenza in x e y del robot;
- stato: stringa che rappresenta lo stato del robot ,default.”Triste”;
- counter\_state: intero che rappresenta il numero di celle senza aver individuato oggetti da recuperare/umani;
- far\_human: lista che rappresenta distanza e direzione relativa all'umano quando lo ha individuato in una cella più lontana di 1m;
- tts: un oggetto della classe TTS;



Legenda valori nelle relative mappe:

- Map (matrice per la mappa principale): -1 cella non ancora visitata, 0 cella libera, 1 cella occupata da oggetto sconosciuto, 2 cella occupata da umano, 3 cella occupata da un oggetto da recuperare, 4 cella occupata da muro o oggetto da non recuperare;
- Visited (matrice per il controllo di celle visitate): FALSE cella non visitata, TRUE cella visitata

Altri metodi utilizzati nella classe sono i seguenti:

- **rimuovi\_umano(x, y)**: metodo che serve per settare il campo CustomData a determinati valori, che a loro volta indurranno il Supervisor a effettuare il “salvataggio” dell’umano;
- **find\_free\_novisited()**: metodo che restituisce una lista di coordinate delle celle libere e non visitate;
- **print\_info()**: metodo che serve per stampare sia la posizione e la direzione del robot sia la mappa trovata sino a quel punto.
- **scansione(x,y)**: metodo il cui scopo è individuare oggetti, attraverso la camera, nelle celle adiacenti o umani più o meno distanti, in particolare se lo stato è Triste permettiamo al Rescuerbot di cercare umani a distanza maggiore, altrimenti sarà limitato a cercarli solamente nelle celle adiacenti; infine in base a cosa troverà verrà riprodotta una frase associata alla scoperta, qualora non dovesse trovare un oggetto di interesse(roccia/mobili/pareti..) porrà l’elemento nella mappa a 4.
- **update\_stato(f)**: metodo che viene invocato per cambiare lo stato emotivo sulla base di oggetti/umani trovati oppure al fine di aumentare il contatore di celle qualora non trovasse alcun elemento d’interesse.
- **ricerca\_ogg(x,y)**: metodo che permette attraverso la lettura dei sensori di invocare la scansione sulla base dello stato emotivo .
- **mapping()**: metodo che implementa l’algoritmo di mapping; attraverso l’uso dei sensori Lidar inizialmente verranno posti ad 1 gli elementi individuati che necessiteranno di una scansione per il riconoscimento , inoltre durante la mappatura qualora il Rescuerbot dovesse individuare un umano ( durante lo stato emotivo triste) esso immediatamente ci si dirigerà per chiamare i soccorsi, riprendendo la mappatura all’avvenuto soccorso; infine prima di terminare verranno controllate le celle non ancora visitate e libere , se non ce ne fossero allora verranno impostate le celle irraggiungibili a 4(ostacolo) e qualora il Rescuerbot dovesse aver saltato qualche scansione di elementi durante la mappatura (stato emotivo Felice) ci si dirigerà per scansionarli e riconoscerli.

## 4.5. Collect.py

La classe collect ha come scopo quello di eseguire la fase di recupero degli oggetti trovati durante la mappatura dell'ambiente, ottenuta in fase esplorativa, cioè localizzazione degli oggetti, creazione percorso più breve per la raccolta di ogni oggetto, movimento lungo il percorso tracciato, riconoscimento e riposizionamento frontale rispetto l'oggetto da raccogliere e tracciamento percorso più breve rispetto punto di rilascio degli oggetti trovati il quale sarà scelto come strada da seguire per il ritorno dell'oggetto smarrito ;

In particolare essa è composta dai seguenti attributi :

- movement: istanza della classe Movement, che permette di muovere il robot verso l'oggetto localizzato nella mappa;
- cam: istanza della classe Cam che permette il riconoscimento dell'oggetto una volta che il robot riesce a visualizzarlo;
- map: una lista python che memorizzerà la mappa ottenuta nella fase di Mapping;
- robot: istanza del nostro robot che permette di accedere ai parametri del mondo creato in webots(principalmente per il timestep);
- pos\_start: variabile che indica la posizione di partenza nella fase di raccolta degli oggetti;
- lift\_motor: dispositivo webots che permette movimenti verticali del gripper necessario in fase di raccolta e rilascio degli oggetti;
- finger\_motor: dispositivo webots che permette l'apertura e la chiusura del gripper necessario per raccogliere e lasciare gli oggetti;
- gripperMaxSpeed: variabile che indica la velocità di movimenti del gripper;
- tts: istanza della classe TTS che permette di trasformare il testo in voce;

e dai seguenti metodi :

- **lift(position)**: permette il movimento verticale del gripper necessario in fase di raccolta dell'oggetto;
- **move\_fingers(position)**: permette la chiusura/apertura delle pinze durante la raccolta ed il rilascio dell'oggetto;

- **get\_well(objs)**: permette attraverso l'istanza della camera la misura della distanza lungo l'asse orizzontale locale della camera rispetto l'oggetto puntato , con conseguente aggiustamento e raddrizzamento del robot rispetto l'oggetto individuato;
- **aggancia()**: permette il riconoscimento dell'oggetto ,ottenimento delle dimensioni dell'oggetto attraverso l'istanza di Cam ,necessarie per i movimenti in fase di raccolta del robot che dovrà abbassare il gripper ,aprire le pinze ,muoversi verso l'oggetto avendo calcolato prima la distanza con il sensore lidar frontale ed essendosi raddrizzato precedentemente ,chiudere le pinze e sollevare il carrello del gripper ed infine tornare nella cella precedente;
- **rilascia(ogg)**: permette il rilascio dell'oggetto agganciato con conseguentemente un esplicitazione vocale da parte del robot dell'oggetto rilasciato;
- **start\_collect()**: metodo principale della classe che permette la raccolta degli oggetti sparsi nella mappa ,trovando il percorso più breve sia verso l'oggetto che verso il punto di rilascio ,e gestendo l'aggancio e rilascio nelle due fasi; in particolare una volta trovato il percorso più breve per il recupero di un oggetto, il robot dovrà seguire il percorso tracciato attraverso il metodo della classe movement (follow\_path\_filtered) il quale prevede oltre al semplice seguire il percorso anche un possibile slittamento di ruote/scosse di terremoto che potrebbero portare il robot a doversi rilocalizzare e tracciare dalla nuova posizione il percorso verso l'oggetto.

## 4.6. Cam.py

La classe Cam corrisponde alla classe relativa all'uso della Camera integrata al robot.

Il costruttore richiede principalmente due input, robot e timestep, ed all'interno attiva sia la Camera sia la funzionalità di riconoscimento oggetti; Webots infatti permette di simulare tale funzione mediante il modulo CameraRecognitionObject che consente di associare ad ogni oggetto un colore di recognition.

L'unica funzione definita è **recognition()**: in essa, tramite il metodo getRecognitionObjects(), troveremo una lista di tutti gli oggetti che la Camera è riuscita a trovare in quella particolare inquadratura. Ovviamente tale metodo ci serve per riconoscere l'oggetto che la Camera ha frontalmente, quindi, dato che la Camera può trovare anche più di un oggetto in una sola inquadratura, eseguiamo delle ulteriori operazioni per far sì che ci torni solo l'oggetto che ha frontalmente e quindi quello più vicino ad essa.

Per il riconoscimento degli oggetti, abbiamo deciso di avere, oltre agli umani che verranno poi "salvati", dei Box (decisione presa soprattutto a causa della successiva presa del gripper integrato al robot) differenziandoli tramite i colori; quindi, trovato l'oggetto più vicino alla Camera, usiamo il metodo getColors() che ci ritnerà i colori RGB relativi a quell'oggetto, ovvero:

- R=1.0, G=1.0, B=0.0 □ oggetto riconosciuto: Box gioielli
- R=1.0, G=0.0, B=0.0 □ oggetto riconosciuto: Umano
- R=0.0, G=1.0, B=0.5 □ oggetto riconosciuto: Box soldi
- R=0.0, G=1.0, B=0.0 □ oggetto riconosciuto: Box foto

La funzione infine ci ritnerà il nome dell'oggetto riconosciuto e la distanza alla quale è stato trovato.

## 4.7. Particle\_filter.py

La classe Particle\_filter ha come scopo quello di implementare il filtro particellare al fine di aiutare la localizzazione del robot nella mappa in casi di emergenza (scosse di terremoto) oppure a causa di slittamento delle ruote ;

Descrizione filtro particellare:

Il filtro particellare è un metodo utilizzato nella robotica probabilistica per la stima di processi non lineari .

Questo filtro è spesso utilizzato per problemi di localizzazione.

Il filtro particellare usa un insieme di campioni pesati(posizioni della mappa), chiamati particelle che rappresentano una possibile ipotesi sullo stato del sistema(posizione del robot nella mappa).

Nel contesto della localizzazione, le particelle vengono propagate secondo il modello di moto del robot(ovvero seguiranno ogni comando di traslazione dato al robot).

Quindi le particelle vengono pesate in base alla verosimiglianza delle osservazioni( nel nostro caso alle celle aventi nei 4 punti cardinali la stessa configurazione cioè la presenza o meno di oggetti attorno alla cella ).

Infine, viene eseguito un ri-campionamento (Campionamento per importanza), in cui le particelle con peso basso vengono eliminate e quelle con peso alto vengono replicate al fine di scartare le particelle meno probabili che non rispettano la configurazione della cella attuale del robot ed incrementare le particelle probabili maggiormente aventi una configurazione compatibile con quella attuale del robot

la classe è composta dai seguenti attributi :

- width: larghezza della mappa
- length: lunghezza della mappa
- map: una lista python che memorizzerà la mappa ottenuta nella fase di Mapping
- particles: array numpy che conterrà tutte le particelle generate uniformemente rispetto le dimensioni della mappa
- n\_particles: variabile che indica il numero totale di particelle

e dai seguenti metodi :

- **redistribution()**: permette la ridistribuzione delle particelle uniformemente rispetto la mappa utile nel caso di perdita dello stato del sistema (posa del robot)
- **print\_particles(particles)**: permette attraverso la libreria matplotlib di esplicitare per via grafica la posizione ed il peso delle particelle nella mappa durante i movimenti del robot
- **real\_state(cell)**: permette di valutare la configurazione dello stato di una cella ovvero se sono presenti oggetti nei 4 punti cardinali
- **evaluate\_mis(misurations)**: valuta la misura ottenuta dai sensori lidar presenti nel robot,e pone 1 nel caso di presenza di oggetti a distanza minore di 1 m altrimenti 0
- **position\_estimate(sd,dir)**: permette di stimare le celle più probabili nella mappa che potrebbero rappresentare la posizione attuale del robot, valutando ogni cella della mappa attraverso il metodo real\_state
- **isLocalized(raggio)**: metodo che valuta la precisione della localizzazione effettuata, analizzando la percentuale di particelle condensate entro il raggio definito e restituendo True qualora la percentuale fosse minore di 0.99 oppure False se la localizzazione effettuata sia ottimale
- **particle\_filter(azione,sd,dir)**: metodo principale della classe che fa uso dei metodi precedenti per applicare il filtro particellare e stimare la posa del robot;  
Inizializziamo i pesi delle particelle come equiprobabili.  
Successivamente per ogni posizione stimata probabile associa i pesi alle particelle misurandoli rispetto la distanza da ogni posizione stimata nella mappa e normalizzandoli , favorendo particelle più vicine alle posizioni stimate ;  
Infine effettua un campionamento per importanza sulle particelle ,ricampionando più frequentemente particelle aventi pesi maggiori e scartando quelle con pesi minori essendo meno probabile il loro campionamento, restituendo il punto medio rispetto la particelle considerate più probabili come posizione stimata.

## 4.8. TTS.py

La classe TTS implementa il sistema che permette al robot di parlare, inizializza lo speaker, imposta l'Engine che permette di tradurre i testi in voce e imposta la lingua.

Essa dispone del metodo:

- **text\_to\_speech(testo)**: metodo che dato in input un testo permette al robot di convertirlo in parole e parlare mediante lo speaker.

## 5. Simulazione

Abbiamo deciso di dividere la simulazione in due parti in base ai due algoritmi sviluppati: Mapping e Collect.

### 5.1. Mapping

Al seguente link si può visualizzare il video di simulazione dell'algoritmo Mapping:

<https://youtu.be/NmGWv1Lw3kg>

### 5.2. Collect

Al seguente link si può visualizzare il video di simulazione dell'algoritmo Collect:

<https://youtu.be/TpUQ5vXL52k>

Al seguente link si può visualizzare il video di simulazione dell'algoritmo Collect con traslazioni:

<https://youtu.be/As6rJ4zv2gw>