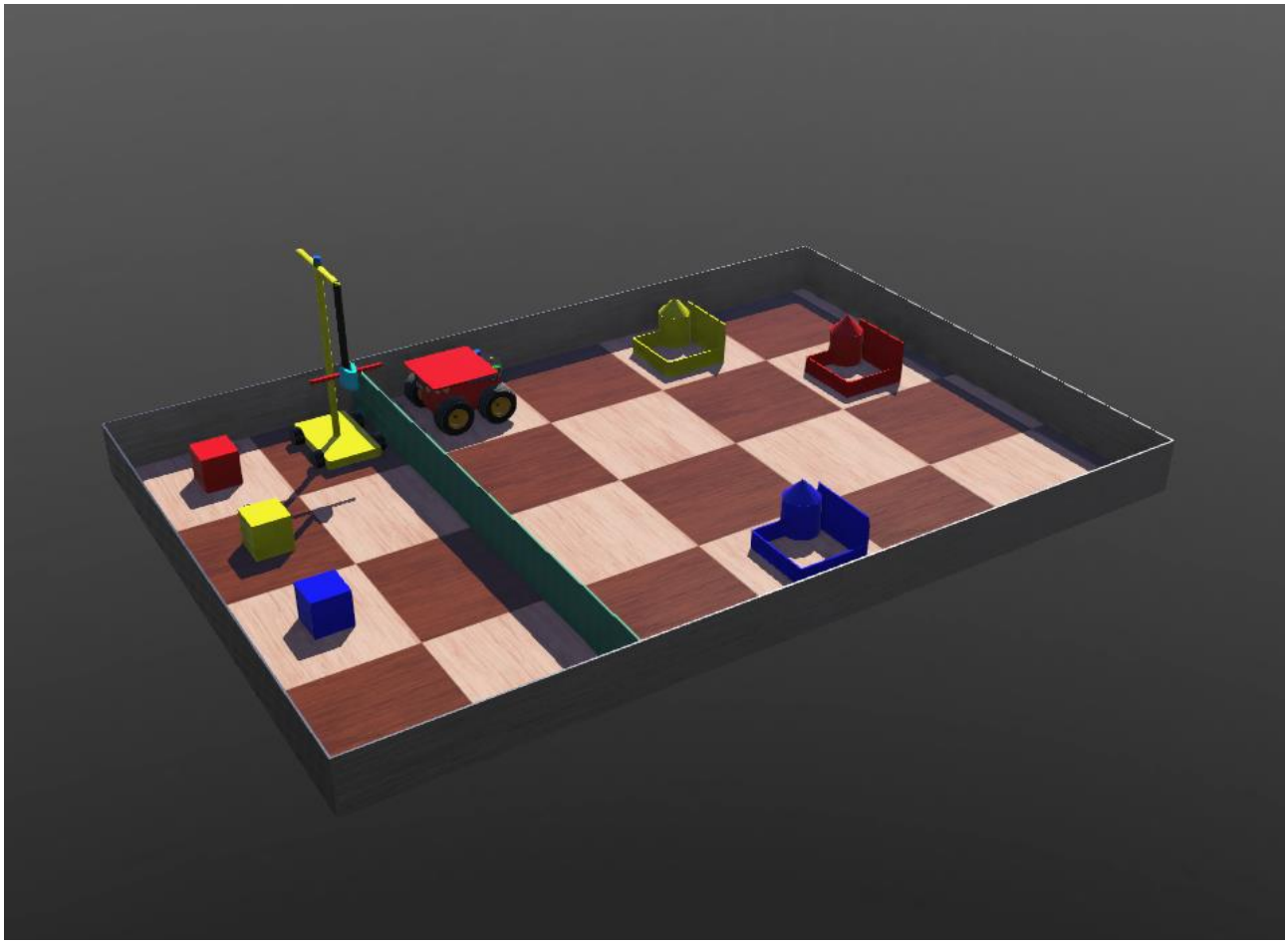
# Progetto Robotica e Intelligenza Artificiale 2
# Robot Corriere & Gru



Università degli Studi di Palermo

Webots
robot simulation

Antonio Pio Sciacchitano

Daniele Franco

# Sommario


1. Introduzione ..... 2
   1.1 Descrizione generale ..... 2
   1.2 Scopo del progetto ..... 2
   1.3 End Users ..... 2
2. Analisi dei requisiti ..... 3
   2.1 Requisiti funzionali ..... 3
   2.2 Requisiti non funzionali ..... 3
3. Componenti del sistema ..... 4
   3.1 Mappa ..... 4
   3.2 Robot Corriere ..... 5
   3.3 Robot Gru ..... 6
   3.4 Diagrammi ..... 7
      3.4.1 Mappatura ..... 7
      3.4.2 Consegne ..... 8
      3.4.3 Smistamento ..... 9
4. Tecnologie utilizzate ..... 9
   4.1 Movimento ..... 9
   4.2 Comunicazione ..... 10
   4.3 Algoritmo A* ..... 10
   4.4 Emozioni ..... 11
   4.5 Filtro di Kalman ..... 11
5. Documentazione codice ..... 12
   5.1 Graph ..... 12
   5.2 KalmanFilter ..... 12
   5.3 kf.py ..... 14
   5.4 corriere.py ..... 15
   5.5 presa.py ..... 16

# 1. Introduzione

## 1.1 Descrizione generale

Il Progetto consiste nella realizzazione di due robot, uno capace di **consegnare** pacchi come un corriere, l'altro, invece, con il compito **caricare** il pacco sul primo robot e di dare informazioni riguardanti la consegna, in modo tale che il primo robot sappia associare il pacco alla casa corretta.

Per semplificazione, da qui in poi, il primo robot sarà chiamato 'robot-corriere' e il secondo 'robot gru'.

Il robot-corriere sa: **mappare** l'area utilizzando la logica del 'wall-following', **individuare** le case presenti e il loro colore, e, una volta ricevuto il pacco dal 'robot-gru', **consegnare** i pacchi alle case del colore corrispondente.

È stato inoltre realizzato un sistema che simula delle **emozioni** che può provare il 'robot corriere', le quali si manifestano durante la consegna: nel momento in cui al 'robot corriere' viene assegnato un pacco, esso è triste e di conseguenza andrà più lento; diversamente, quando ha effettuato la consegna, essendo felice, andrà più veloce.

Il robot-gru sa **prendere** i pacchi e **trasportarli** grazie a dei bracci estensibili, e comunicare la destinazione a seconda del colore.

L'implementazione è stata effettuata mediante il software di simulazione **Webots** che ha permesso di realizzare un ambiente nel quale sono presenti delle case di diversi colori e una zona in cui si trovano i pacchi dei colori corrispondenti.

## 1.2 Scopo del progetto

Lo scopo principale del progetto è quello di realizzare un ambiente in cui interagiscano più robot in **collaborazione**, che sfruttino i dati ricavati dai **sensori** di cui sono dotati e agiscano sul mondo tramite gli **attuatori**.

## 1.3 End-Users

Gli utilizzatori finali del robot saranno tutti quei soggetti che hanno necessità di una consegna rapida e mirata, la quale potrà essere effettuata sia all'interno di un locale chiuso, come un complesso industriale, nel quale è necessario spostare un determinato oggetto da una parte all'altra con una logica, sia al di fuori di esso, come in un paese o città per consegnare pacchi postali.

# 2. Analisi dei requisiti

## 2.1 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 di distanza che gli permettono di misurare la distanza dagli oggetti nelle 4 direzioni, e di seguire la logica del Wall Following
- **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 il loro colore.
- **Consegna**: il robot è in grado di consegnare gli oggetti nell'ambiente, questo avviene mediante una pinza di cui è dotato.
- **Localizzazione**: deve essere dotato di un sistema basato sul filtro di Kalman per il tracking della posizione del robot
- **Emozioni**: il robot è in grado di simulare degli stati emotivi che gli permettono di essere più veloce di essere più veloce nelle consegne.

## 2.2 Requisiti non funzionali

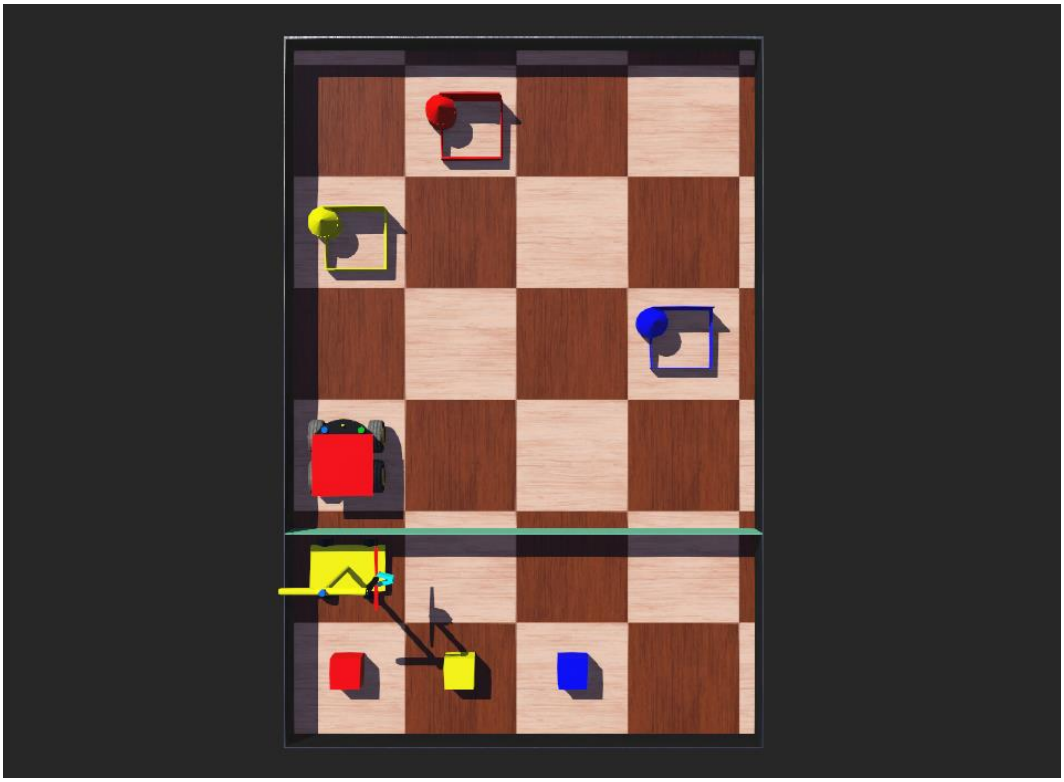
- **Utilizzo del sistema di simulazione Webots**: Il sistema di controllo del robot deve essere compatibile con Webots per validare le strategie di controllo in un ambiente virtuale realistico e configurabile.
- **Prestazioni**: Il codice del controller del robot deve essere scritto in Python per garantire una maggiore compatibilità con i controller, semplificando così la manutenzione e l'estensione del sistema nel tempo.
- **Precisione dei sensori di distanza**: I sensori di distanza devono avere una varianza massima di 0.1 per garantire letture precise e affidabili, fondamentali per la navigazione e il rilevamento degli ostacoli del robot.

# 3. Componenti del sistema

Il sistema è diviso in diversi componenti che **interagiscono** tra loro: la mappa che ospita l'ambiente e i due robot che collaborano.

## 3.1 Mappa

Per la realizzazione dello spazio si è scelto di simulare uno scenario, suddiviso in **due parti**, di cui una rappresenterebbe il centro di smistamento dei pacchi, mentre l'altra un quartiere residenziale, con delle case, a cui devono essere consegnati dei pacchi.



Essa è alta 4.6 metri e larga 3.1 metri. Le dimensioni sono state aumentate di 1 centimetro per lasciare più spazio di rotazione al robot-corriere. E' suddivisa in celle quadrate aventi lato di 75 centimetri l'una.

La mappa è **codificata** in modo diverso dai due robot:

- Per il robot corriere è codificata come una **matrice** in un mondo 2D, con una lista di 5 elementi per ogni cella, dove il primo elemento contiene la codifica della cella (se è occupata vi è l'iniziale della lettera del colore, se è libera c'è lo '0'); mentre gli altri elementi si riferiscono allo stato occupato o meno delle celle adiacenti, nord-est-sud-ovest e sono codificate con '0' e '1', se sono occupate con '1' se sono libere con '0'.

```
mappa=[[0,1,1,0,1],     ['R',1,0,0,0],  [0,1,0,0,1],    [0,1,1,0,0],
       ['G',0,0,0,1],   [0,1,0,0,1],    [0,0,0,0,0],    [0,0,1,1,0],
       [0,1,0,0,1],     [0,0,0,0,0],    [0,0,1,0,0],    ['B',0,1,0,0],
       [0,0,0,1,1],     [0,0,0,1,0],    [0,0,0,1,0],    [0,1,1,1,0]]
```

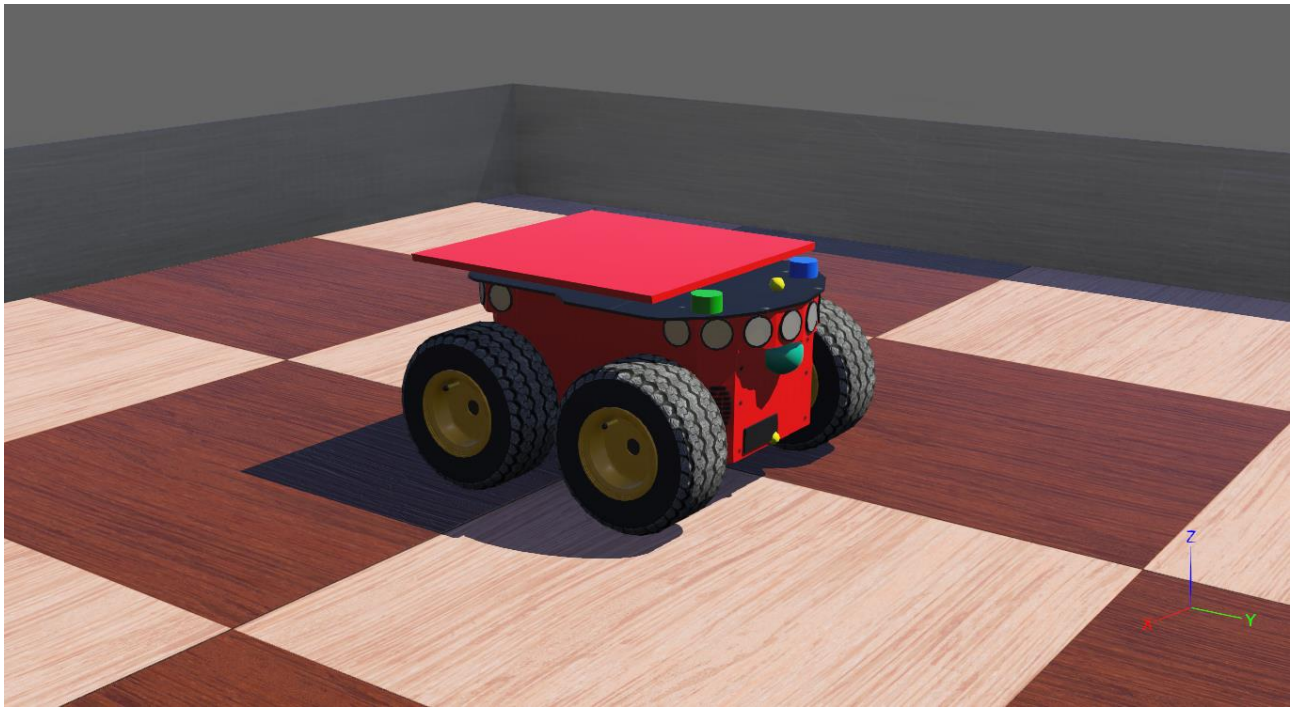
- Per il robot gru la codifica è composta da una **lista** che indica l'ordine da sinistra a destra dell'ordine del colore dei pacchi.

```
consegne=["R", "G", "B"]
```

## 3.2 Robot Corriere

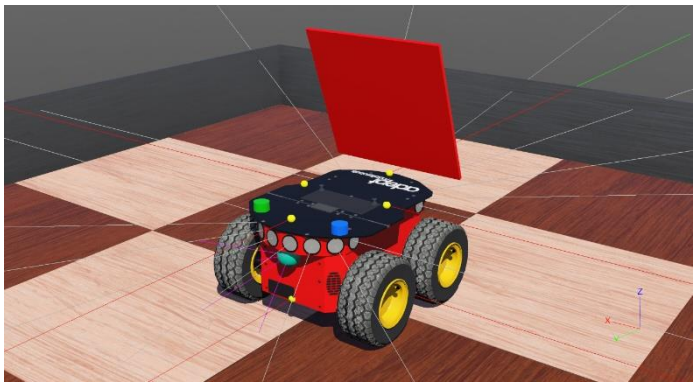
Il robot che è stato scelto per l'implementazione è il **Pioneer 3-AT** di Adept; esso è un robot multiuso, utilizzato per applicazioni di ricerca e prototipazione che coinvolgono mappatura, navigazione, monitoraggio, ricognizione e altri comportamenti. La piattaforma base Pioneer 3-AT è assemblato con motori dotati di encoder da 500 tick, ruote da 22 cm, robusto corpo in alluminio, 8 sensori a ultrasuoni (sonar) rivolti in avanti, 8 sonar posteriori opzionali.

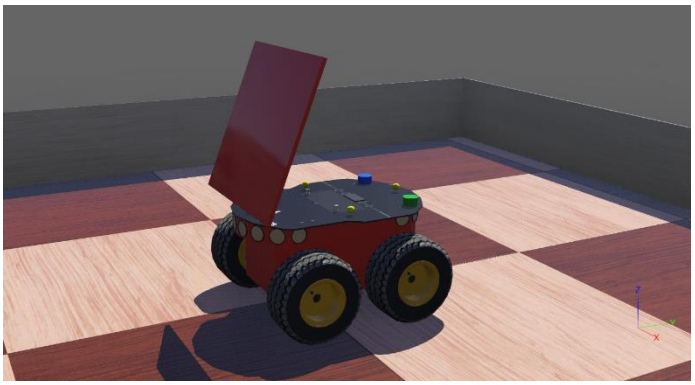
Il robot può raggiungere velocità di 1,6 metri al secondo e trasportare un carico utile fino a 35 Kg.




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

Sono stati aggiunti inoltre dei sensori **emettitore** (verde) e **ricevitore** (blu) che permettono di comunicare con il robot gru, una **camera** posta sotto i sensori del robot (sfera verde) che ci permette di riconoscere gli oggetti, e due ulteriori sensori di distanza posti in basso utilizzati per il **wall-following**.

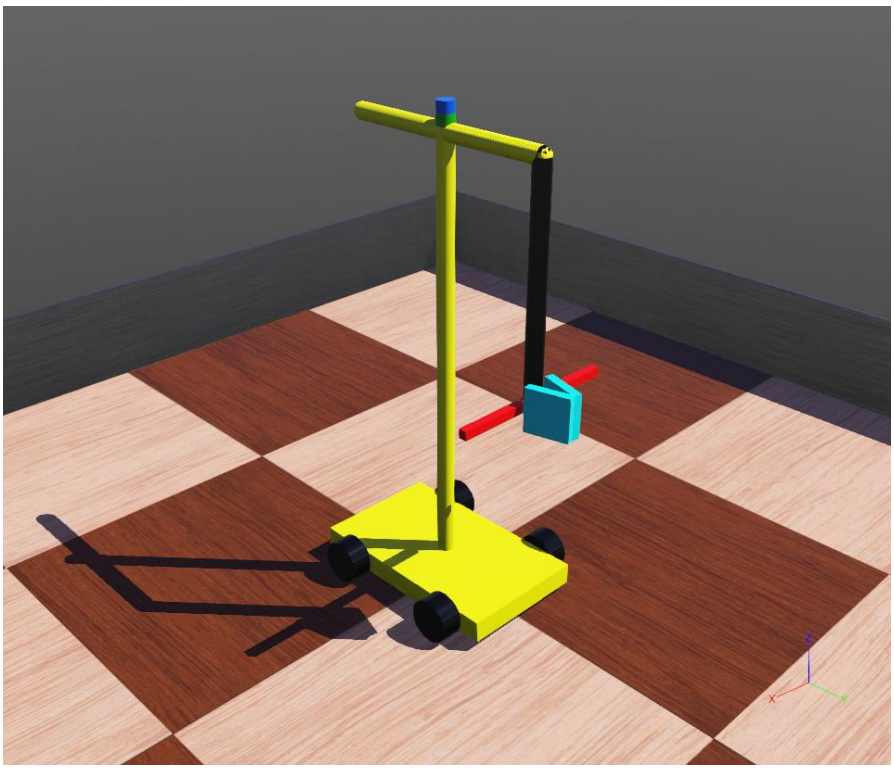




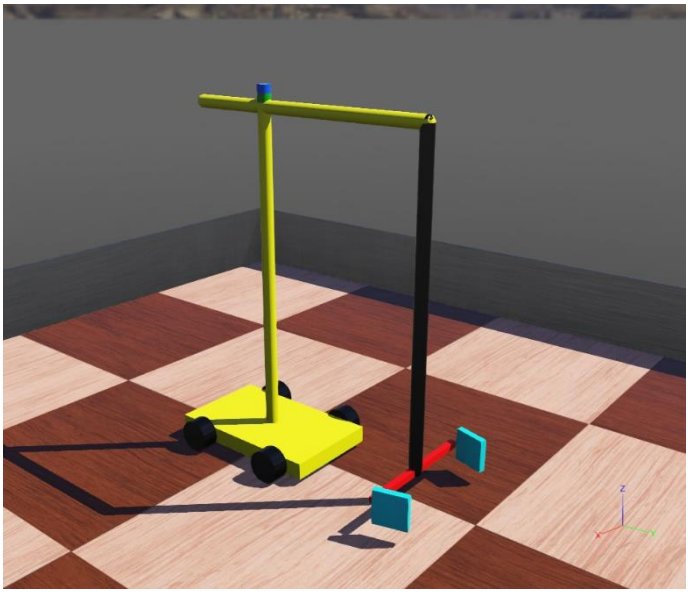
Sulla parte superiore del robot è stata aggiunta una **piastra inclinabile** che permette di trasportare gli oggetti con più stabilità e di scaricarli.

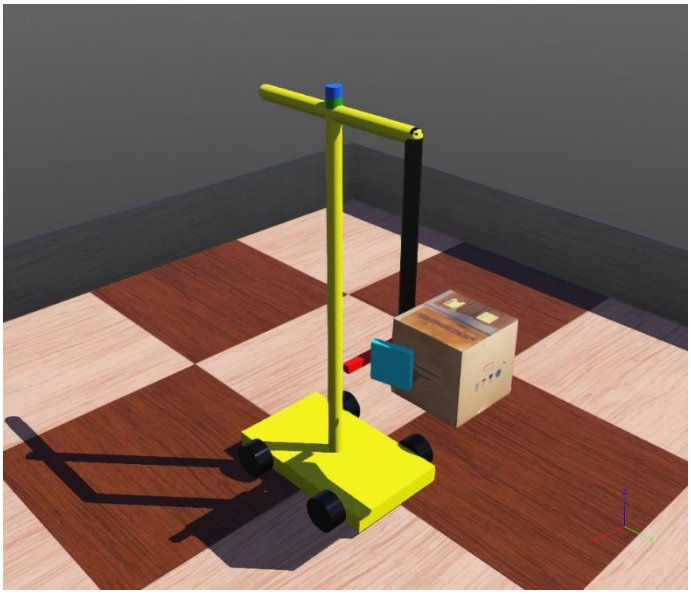
## 3.3 Robot Gru

Il robot è stato costruito ex-novo per adattarlo alle necessità del progetto. È ispirato al concetto della **gru**, in quanto deve poter prendere e trasportare oggetti, eventualmente ruotando su sé stesso. Inoltre, è munito di quattro ruote per potersi spostare avanti e indietro.



È dotato di due **bracci estensibili**, una orizzontale e una verticale, all'estremità del quale vi sono due **pinze** che possono allargarsi o restringersi, in modo per adattarsi alla forma dell'oggetto da afferrare. Può **ruotare** su sé stesso di 360° e possiede anche un **emettitore** (verde) e un **ricevitore** (blu) che gli permettono di comunicare con il robot corriere.

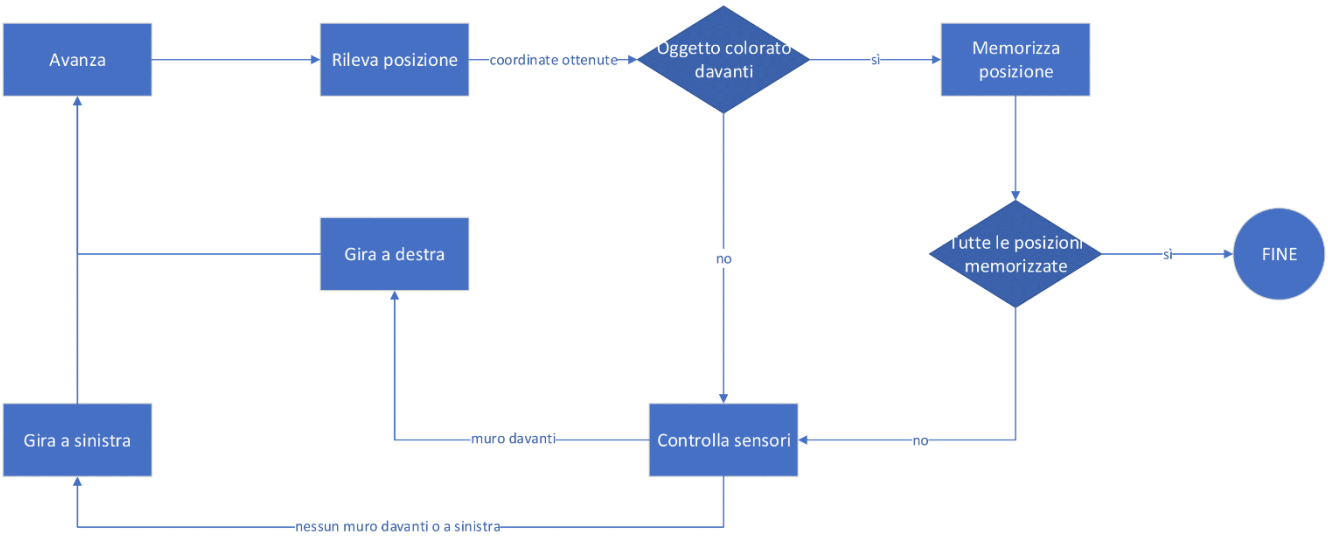




## 3.4 Diagrammi

Per potersi integrare, le varie componenti del sistema seguono uno schema ben preciso, ovvero una **sequenza di stati**, evidenziati nei seguenti diagrammi.

### 3.4.1 Mappatura



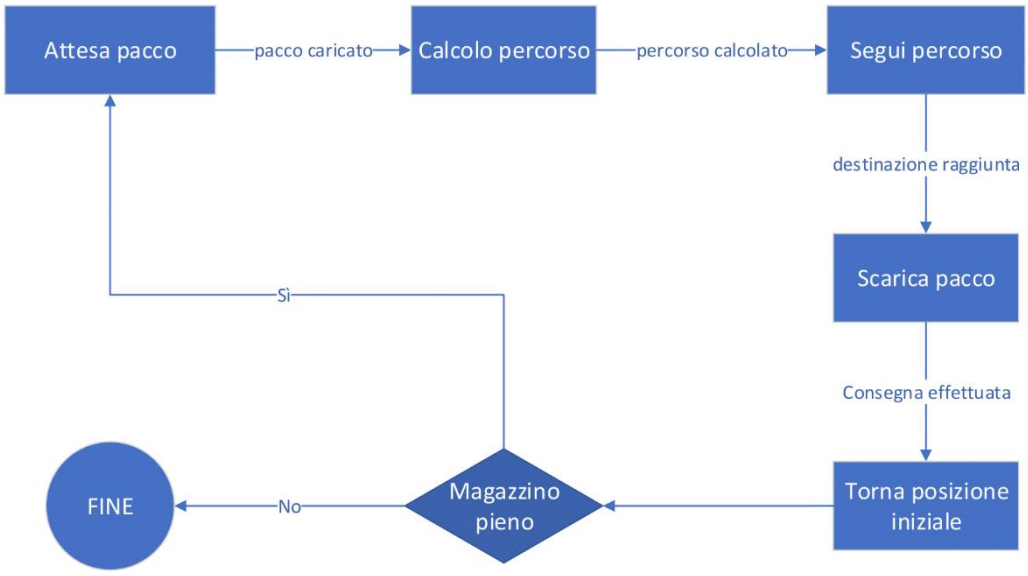

La prima fase del progetto è la **mappatura**, cioè la fase in cui il robot corriere esplora la mappa seguendo la logica del wall-following e memorizza la posizione, ovvero il numero della cella, in cui si trovano le case, che vengono riconosciute tramite il loro colore dalla camera.

La logica del wall-following consiste nel **costeggiare** i muri della mappa, lasciando sempre una parete alla sinistra del robot, esplorando così in **senso orario**. Qualora il robot si trovasse un muro davanti, allora esso ruoterà a destra, mentre se non trovasse né un muro a sinistra né davanti, allora ruoterà verso sinistra.

Nel frattempo che avanza, viene stimata la posizione del robot per poter di conseguenza conoscere quella delle case.

Quando incontrerà un oggetto abbastanza **vicino**, ne esaminerà il colore e, a seconda della direzione cardinale, ne memorizzerà la **posizione**.

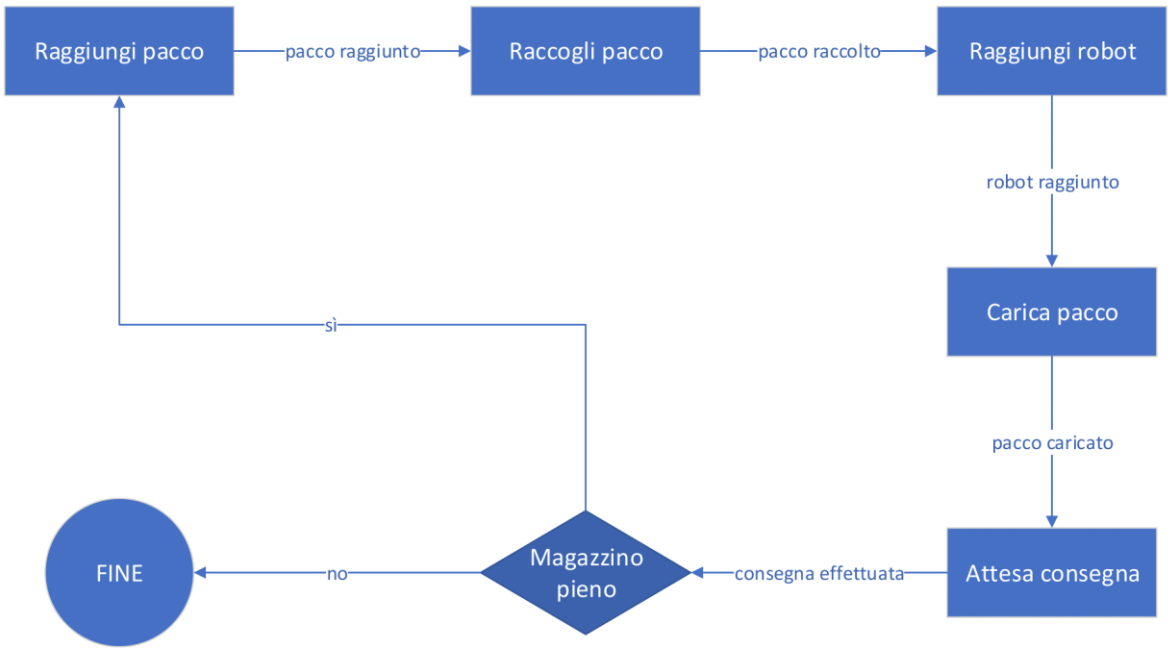
### 3.4.2 Consegne




La seconda fase prevede che i due robot **collaborino** per portare a termine l'**obiettivo** comune, cioè che vengano effettuate tutte le consegne.

Il robot corriere si occupa della porzione della mappa che simula la zona residenziale; infatti, attende che venga caricato il pacco sopra di esso dal robot gru, per poi calcolare il percorso verso la destinazione corretta e seguirlo. Una volta consegnato tornerà alla posizione iniziale e, nel caso ci fossero altre consegne da effettuare, attenderà il pacco successivo.

### 3.4.3 Smistamento




Il robot gru si occupa della porzione di mappa che simula il magazzino, dove sono contenuti tutti i pacchi da consegnare. Esso prima raggiunge il pacco, lo afferra e lo trasporta fino al robot corriere per poi caricarlo. In seguito, attende che la consegna sia avvenuta per potersi recare verso il pacco successivo, se il magazzino dovesse ancora contenerne.

# 4. Tecnologie utilizzate

Per la realizzazione del progetto sono state implementate e utilizzate cinque tecnologie e metodologie diverse, in modo da permettere ai robot di svolgere tutte le funzioni richieste.

## 4.1 Movimento

I robot seguono un modello di movimento a **linee spezzate**; infatti, hanno soltanto due movimenti a disposizione: **lineare** e **rotazionale**.

Il movimento lineare è realizzato facendo muovere i motori tutti alla **stessa velocità**:

$$v_{dx} = v_{sx}$$

Quello rotazionale invece è realizzato facendo muovere i motori della parte **destra in modo opposto** a quelli di sinistra, a seconda del verso di rotazione:

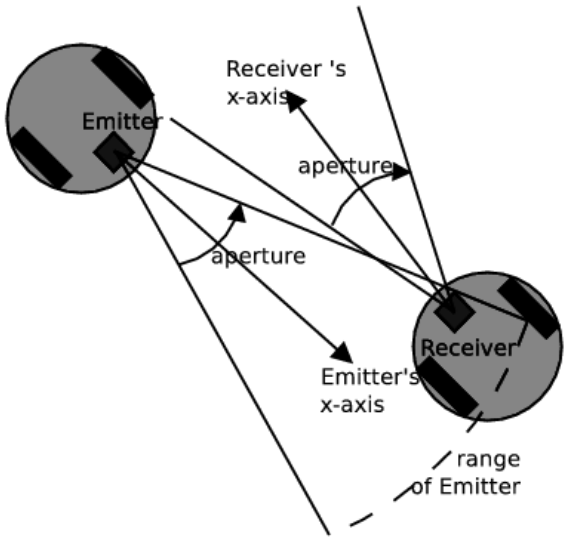
$$v_{dx} = -v_{sx}$$

Questo tipo di movimento, oltre ad essere semplice da implementare, si adatta molto bene con il mondo a griglia, poiché i robot possono spostarsi di una cella alla volta.

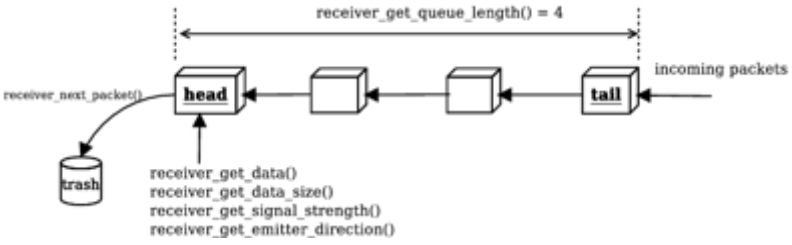
## 4.2 Comunicazione

La tecnologia che permette ai robot di comunicare in modo bidirezionale è implementata attraverso un **emettitore** e un **ricevitore**.

L'emettitore modula segnali radio, seriali o infrarossi, a seconda del tipo scelto, e vengono inviati tramite un canale di comunicazione. In questo caso i segnali sono di tipo **radio** e sono inviati in canale **broadcast** con un range illimitato.




Il ricevitore riceve e può processare segnali radio seriali e infrarossi, entro un determinato range e trasmessi in un determinato canale. I dati vengono ricevuti in **pacchetti**, trasmessi ad una certa frequenza, e organizzati in una **coda**, che si svuota via via che i segnali vengono elaborati.




## 4.3 Algoritmo A*

L'algoritmo di ricerca A* ai basa sui **grafi** e individua un percorso da un nodo iniziale ad uno goal. Si basa su stime **euristiche**, quindi garantisce di trovare il percorso più **breve** al **minor costo** possibile.

In questo caso il costo coincide con la lunghezza del percorso, infatti, l'obiettivo è ottenere la sequenza di celle più corta che porta alla destinazione desiderata.

## 4.4 Emozioni

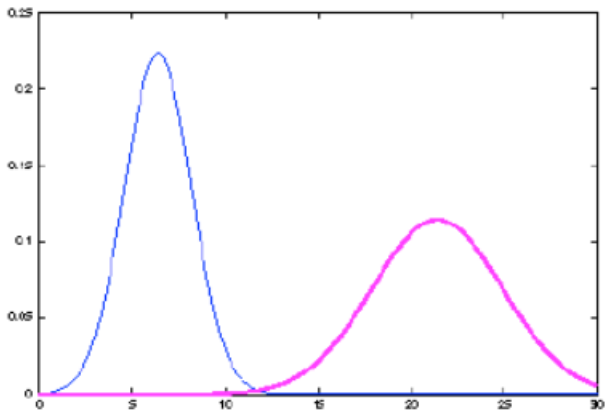

Il robot Corriere è dotato di due **stati emozionali**: “normale” e “felice”.

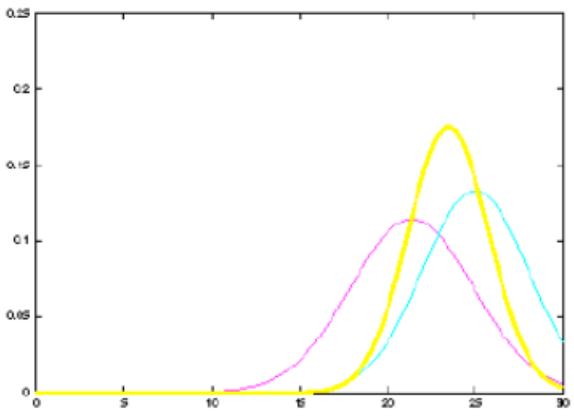
Lo stato “normale” viene assunto quando il robot è in **attesa** che il pacco venga caricato sopra di esso, mentre quello “felice” viene raggiunto quando è appena stata **effettuata** una consegna.

Essi vengono simulati attraverso un **cambio di velocità**, raddoppiandola quando il robot si trova nello stato “felice”.

## 4.5 Filtro di Kalman

Per realizzare la prima fase del progetto, ovvero quella di **mappare** la cella in cui si trova ogni casa, viene implementato il filtro di Kalman, che per mette al robot corriere di **stimare la sua posizione**. In questo modo, grazie al riconoscimento dei colori della camera, riesce a stimare la posizione delle case nella mappa, convertendo le coordinate nel numero della cella corrispondente.





Il filtro di Kalman viene applicato iterando le misurazioni e il movimento. Viene effettuata una **predizione** sulla posizione futura del robot in base al movimento che sta compiendo, e in seguito vengono effettuate delle **misurazioni**, tramite i sensori, che vengono sfruttate per **aggiornare** e **correggere** la predizione fatta in precedenza. Questo ciclo corrisponde alla moltiplicazione tra due distribuzioni di probabilità Gaussiane, che secondo le proprietà, il risultato è ancora una Gaussiana.

# 5. Documentazione codice

Il progetto è composto da tre controller principali e da due classi: Graph e KalmanFilter

## 5.1 Graph

La classe graph è progettata per costruire un grafo a partire da una mappa in modo da eseguire l'algoritmo A* a partire da esso.

Presenta la variabile **graph** in cui è memorizzato il grafo sotto forma di dizionario, in cui per ogni chiave, le celle, sono associati i loro vicini.

Possiede diversi metodi:

- **Costruttore __init__(mappa)**: prende in input la mappa, rappresentata come matrice, la converte nel grafo e istanzia la variabile graph assegnandole il dizionario appena creato.
- **get_neighbors(node)**: restituisce i vicini di un dato nodo nel grafo
- **a_star_algorithm(start, stop)**: restituisce la sequenza di nodi individuando il percorso ottimale tra il nodo di partenza e quello destinazione, utilizzando appunto l'algoritmo A*. Esso si basa su due liste: una tiene traccia dei nodi già visitati, mentre l'altra dei nodi da visitare. Determina il costo del percorso, e quindi quale nodo scegliere rispetto ad un altro, tramite la lunghezza, quindi prediligerà i percorsi più brevi.

## 5.2 KalmanFilter

La classe KalmanFilter è un'implementazione del filtro di Kalman, ovvero dell'algoritmo di stima della posizione per tracciare oggetti in sistemi dinamici.

Presenta le seguenti variabili:

- **dt**: frequenza di campionamento
- **vettore x**: definisce lo stato. $x_1$ rappresenta la coordinata $x$, $x_2$ la coordinata $y$, $x_3$ la velocità lungo $x$ e $x_4$ la velocità lungo $y$.

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \dot{x}_{k-1}\Delta t + 1/2\ddot{x}_{k-1}\Delta t^2 \\ y_{k-1} + \dot{y}_{k-1}\Delta t + 1/2\ddot{y}_{k-1}\Delta t^2 \\ \dot{x}_{k-1} + \ddot{x}_{k-1}\Delta t \\ \dot{y}_{k-1} + \ddot{y}_{k-1}\Delta t \end{bmatrix}$$

- **vettore u**: definisce le variabili in input di controllo. $u_1$ rappresenta l'accelerazione nella direzione $x$, mentre $u_2$ l'accelerazione nella direzione $y$.
- **Matrice A**: rappresenta la relazione tra lo stato attuale e quello successivo dell'oggetto nel tempo.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Matrice B**: definisce l'effetto dell'ingresso di controllo (come l'accelerazione) sullo stato dell'oggetto.

$$\mathbf{B} = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 & 0 \\ 0 & \frac{1}{2}(\Delta t)^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}$$

- **Matrice H**: specifica come lo stato dell'oggetto viene misurato per ottenere le misurazioni disponibili.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- **Matrice di covarianza Q**: rappresenta l'incertezza associata al modello del processo dinamico dell'oggetto.

$$\mathbf{Q} = \begin{bmatrix} \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} & 0 \\ 0 & \frac{\Delta t^4}{4} & 0 & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & 0 & \Delta t^2 & 0 \\ 0 & \frac{\Delta t^3}{2} & 0 & \Delta t^2 \end{bmatrix} \sigma_a^2$$

- **Matrice di covarianza R**: indica l'incertezza associata alle misurazioni effettuate dall'oggetto.

$$\mathbf{R} = \begin{array}{c} \\ x \\ y \end{array} \begin{array}{c} \begin{array}{cc} x & y \end{array} \\ \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \end{array}$$

- **std_acc**: rappresenta la magnitudo del rumore del processo.
- **x_std_meas**: indica la deviazione standard delle misurazioni nella direzione $x$
- **y_std_meas**: indica la deviazione standard delle misurazioni nella direzione $y$

Possiede i metodi:

- **Costruttore __init__(dt, u_x, u_y std_acc, x_std_meas, y_std_meas)**: inizializza tutte le variabili della classe. Le matrici hanno già una impostazione di default, mentre le altre dipendono dal valore passato in input.
- **predict()**: esegue la predizione dello stato futuro dell'oggetto. Utilizzando il modello dinamico del sistema e il controllo fornito (accelerazione), viene effettuata la stima insieme alla sua incertezza attraverso l'aggiornamento della covarianza dell'errore.

$$\hat{\mathbf{x}}_k^- = A\hat{\mathbf{x}}_{k-1} + B\mathbf{u}_{k-1}$$

$$\mathbf{P}_k^- = A\mathbf{P}_{k-1}A^T + \mathbf{Q}$$

- **update(z)**: esegue l'aggiornamento dello stato stimato dell'oggetto utilizzando una nuova misura proveniente dai sensori (z). Utilizzando il confronto tra la misura attuale e la previsione dello stato, insieme alla loro incertezza, viene calcolato il guadagno di Kalman. Questo guadagno viene quindi utilizzato per correggere lo stato predetto, riducendo così l'errore stimato.

$$\mathbf{P}_k = (I - \mathbf{K}_k H)\mathbf{P}_k^-$$

$$\mathbf{K}_k = \mathbf{P}_k^- H^T (\mathrm{H}\mathbf{P}_k^- H^T + \mathbf{R})^{-1}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - H\hat{\mathbf{x}}_k^-)$$

## 5.3 kf.py

Questo controllore si occupa della fase di mapping ed è assegnato al robot corriere.

Presenta le variabili:

- **robot**: istanza del robot
- **positions**: dizionario con le posizioni individuate delle tre case
- **speed**: velocità del robot in $rad/s$
- **raggio**: raggio delle ruote
- **speed_ms**: velocità del robot in $m/s$
- **asse**: distanza tra i centri delle ruote
- **dir**: punto cardinale verso cui è direzionato il robot
- **pos**: numero della cella in cui si trova il robot
- **timestep**: numero di timestep del mondo corrente Webots
- **KF**: istanza della classe KalmanFilter
- **ds0 - ds3**: istanza dei sensori di distanza utilizzati per il tracciamento della posizione
- **dw0, dw1**: istanza dei sensori di disanza utilizzati per il wall-following
- **camera**: istanza della camera
- **b_sx/dx, f_sx/dx**: istanze dei motori delle quattro ruote

Possiede i metodi:

- **rotate(verso)**: ruota il robot di 90° in senso orario o antiorario, a seconda del verso dato in input.
- **get_sensor_ds()**: restituisce i valori di due sensori, uno per la cooridnata x e l'altro per la y. I due sensori da leggere vengono scelti a seconda della direzione.
- **get_position()**: restituisce le coordinate della posizione del robot utilizzando il ciclo predict-update del filtro di Kalman.
- **get_cell(x,y)**: restituisce il numero della cella in cui si trova il robot mappando

le coordinate x e y.

- **check_colore()**: restituisce il colore rilevato dalla telecamera, False se non ne viene rilevato nessuno
- **save_pos(colore)**: aggiorna il dizionario positions inserendo la posizione rilevata della casa del colore dato in input, a seconda della direzione cardinale del robot
- **check_stop()**: controlla se il robot debba fermarsi, cioè se tutte le posizioni sono state rilevate
- **check_turn()**: controlla se il robot debba effettuare una curva leggendo i dati dei sensori dw0 e dw1 e ritorna il verso di rotazione.
- **change_dir(turn)**: aggiorna la direzione cardinale del robot dopo che è stata effettuata una rotazione a partire da quella attuale, a seconda del verso di rotazione dato in input.

Infine, vi sono tre print in cui vengono stampate le posizioni delle case rilevate.

```
Casa Rossa rilevata nella cella: 1
Casa Blu rilevata nella cella: 10
Casa Gialla rilevata nella cella: 4
```

## 5.4 corriere.py

Il controller "corriere.py" è assegnato al robot corriere e gestisce la movimentazione del robot, il calcolo dei percorsi, la gestione delle consegne e la comunicazione con il sistema di gestione del magazzino. Utilizza un algoritmo di ricerca A* per determinare i percorsi ottimali verso le destinazioni specificate e per effettuare le consegne dei pacchi

Presenta le variabili:

- **robot**: istanza del robot
- **emitter**: istanza dell'emettitore
- **receiver**: istanza del ricevitore
- **mappa**: codifica della mappa
- **grafo**: istanza della classe grafo
- **raggio**: raggio delle ruote
- **asse**: distanza tra i centri delle ruote
- **dir**: punto cardinale verso cui è direzionato il robot
- **start**: posizione iniziale
- **pos**: numero della cella in cui si trova il robot
- **timestep**: numero di timestep del mondo corrente Webots
- **b_sx/dx, f_sx/dx**: istanze dei motori delle quattro ruote
- **m**: istanza del motore della piastra inclinabile
- **emote**: rappresentazione stato emozionale

Possiede i metodi:

- **move_forward**(): Avanza il robot di una cella nella direzione attuale.
- **rotate(verso)**: ruota il robot di 90° in senso orario o antiorario, a seconda del verso dato in input.
- **face_nord(), face_sud(), face_est(), face_ovest():** Orienta il robot verso una direzione cardinale tramite una serie di rotazioni.
- **wait():** Mette il robot in attesa per mezzo secondo.
- **follow_path(path):** Fa seguire al robot il percorso specificato attraverso una successione di avanzamenti e rotazioni.
- **scarica():** Aziona il motore della piastra inclinabile per scaricare un oggetto.
- **get_destinazione():** Riceve la destinazione dal robot gru tramite un messaggio.
- **get_pos_casa(dest):** Trova la posizione nella mappa di una casa data la sua etichetta.
- **send_OK():** Manda un messaggio di conferma al sistema.

## 5.5 presa.py

Il controller “presa.py” è assegnato al robot gru ed è responsabile del recupero dei pacchi dal magazzino utilizzando un braccio robotico. Esso gestisce la movimentazione del braccio, il riconoscimento visivo dei pacchi e la comunicazione con il sistema di gestione del magazzino per coordinare le operazioni di prelievo.

Presenta le variabili:

- **robot**: istanza del robot
- **emitter**: istanza dell’emettitore
- **receiver**: istanza del ricevitore
- **consegne**: codifica della mappa
- **speed**: velocità del robot in $rad/s$
- **raggio**: raggio delle ruote
- **speed_ms**: velocità del robot in $m/s$
- **pos**: numero della cella in cui si trova il robot
- **h**: istanza del braccio orizzontale
- **v**: istanza del braccio verticale
- **l**: istanza della pinza sinistra
- **r**: istanza della pinza destra
- **b**: istanza del motore rotazionale della base
- **r_sx/dx, f_sx/dx**: istanze dei motori delle quattro ruote
- **timestep**: numero di timestep del mondo corrente Webots

Possiede i metodi:

- **wait()**: Mette il robot in attesa per un breve periodo di tempo.
- **prendi()**: Aziona i bracci del robot per afferrare un oggetto.
- **gira(verso)**: Ruota il robot in senso orario o antiorario, a seconda del verso dato in input.
- **rilascia()**: Rilascia un oggetto afferrato.
- **sposta(dir)**: Sposta il robot in avanti o indietro.
- **check()**: Controlla se è stato ricevuto un messaggio di conferma dal corriere.
- **send(messaggio)**: Invia un messaggio contenente la destinazione al corriere.
- **pos_iniziale()**: Riporta il robot alla posizione iniziale.