

Alma Mater Studiorum Università di Bologna
Corso di laurea in Ingegneria Informatica Magistrale

Realizzazione di un Differential Drive Robot a Controllo Remoto

Attività progettuale in Ingegneria dei Sistemi Software

Alfredo Grande
Prof. Antonio Natali

Sessione d'esami Settembre 2025

Indice

Indice	2
Obiettivi	3
Requisiti	3
Requisiti Hardware	3
Requisiti Software	3
Requisiti Funzionali	3
Vincoli di progetto	4
Analisi dei requisiti	4
Macro-componenti software	4
Realizzazione hardware	4
Sistema di controllo locale	8
Criticità riscontrate	9
Sistema di controllo centrale	10
Monitoraggio della posizione del robot	10
Possibili approcci al problema	11
SLAM: Simultaneous Localization and Mapping	11
Fiducial markers + overhead camera	11
Sistema di coordinate e tracciamento del robot	12
Controllo del movimento del robot	12
Controllo vettoriale diretto	13
Controllo PID	13
Pianificazione del movimento	15
Attore QAK	18
Espansioni future	20
Miglioramenti hardware	20
Espansioni del software	20

Obiettivi

L'obiettivo di questa attività progettuale è quello di integrare un sistema fisico al software prodotto con il progetto *TemaFinale25*, realizzato per l'esame di Ingegneria dei Sistemi Software.

Al termine del progetto d'esame, è stato prodotto cargoservice, un software per l'automazione delle operazioni di carico della stiva di una nave per mezzo di un Differential Drive Robot (DDR) autonomo.

Per semplificare le operazioni di testing, è stato utilizzato durante lo sviluppo un Virtual Robot, una simulazione di un DDR in uno spazio virtuale (Wenv).

Al termine di questa attività progettuale, il Virtual Robot verrà sostituito con un robot fisico, integrato all'interno del sistema e in grado di interfacciarsi al software esistente.

Requisiti

Requisiti Hardware

- **RH1:** Realizzare un differential drive robot (DDR) fisico dotato di motori e sensori necessari al movimento nello spazio.
- **RH2:** Garantire la connettività del DDR con il sistema esterno

Requisiti Software

- **RS1:** Realizzare un sistema di controllo per gestire gli spostamenti del DDR
- **RS2:** Implementare un sistema di localizzazione del DDR all'interno di una mappa limitata.
- **RS3:** Implementare algoritmi di *pathfinding* per consentire il movimento autonomo con evitamento ostacoli.
- **RS4:** Realizzare un'interfaccia grafica per il monitoraggio e la gestione remota del DDR.

Requisiti Funzionali

- **RF1:** Il DDR deve essere integrato e compatibile con il sistema *cargoservice*.
- **RF2:** Il DDR deve poter operare sia in modalità autonoma (navigazione con *pathfinding*) sia tramite controllo manuale dell'utente.

Vincoli di progetto

Il vincolo principale è rappresentato dal budget limitato: essendo un progetto autofinanziato, la spesa complessiva per l'acquisto dell'hardware non deve superare **60 Euro**.

Analisi dei requisiti

L'obiettivo di questa attività progettuale è quello di sostituire il Virtual Robot impiegato nel sistema cargosystem mantenendo lo stesso modello di comunicazione e interazione con il sistema.

Macro-componenti software

I macro-componenti software da realizzare durante l'attività progettuale sono:

1. Sistema di controllo locale (a bordo del robot)

- Responsabile dell'esecuzione dei comandi di movimento
- Riceve istruzioni dall'esterno
- Gestisce l'attuazione dei motori e il feedback dei sensori

2. Sistema di controllo centrale

- Coordina e supervisiona il comportamento del robot
- Monitora la posizione e lo stato del robot all'interno dell'ambiente
- Pianifica i movimenti (con algoritmi di pathfinding) e invia i comandi al sistema di controllo locale

3. Attore QAK

- Interfaccia tra il software *cargoservice* e il sistema di controllo centrale
- Riceve comandi dal sistema *cargoservice* e li traduce in messaggi comprensibili al sistema di controllo

Realizzazione hardware

La fase successiva ha riguardato la **realizzazione del robot fisico**.

Questa attività è stata affrontata per prima poiché l'hardware rappresentava il principale vincolo del progetto: disporre rapidamente di un prototipo funzionante è stato considerato un passo necessario per validare i requisiti, comprendere meglio i limiti del sistema e orientare le scelte di progettazione successive.

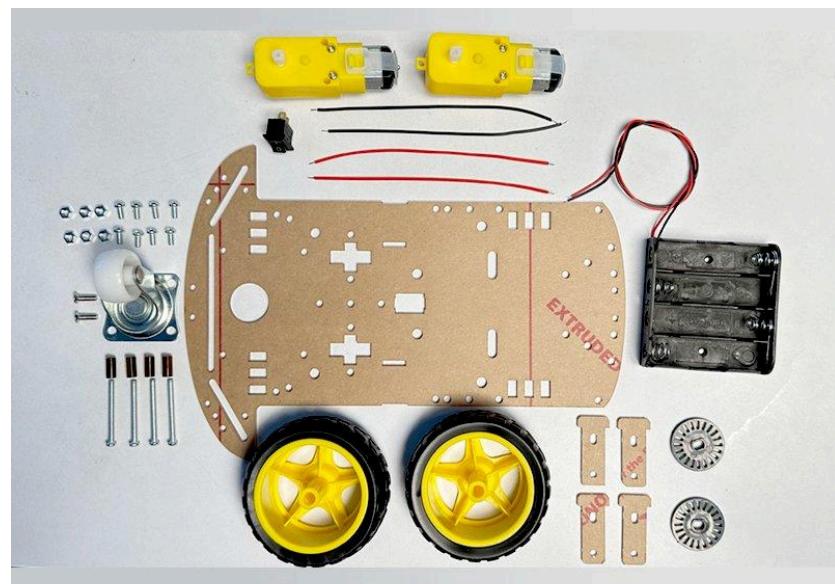
Per il controllo del robot, la scelta dell'unità di elaborazione si è ridotta a due opzioni principali: **Raspberry Pi 3B** o **Arduino Uno**.

Fra le due opzioni, Arduino sarebbe stato in grado di gestire il controllo dei motori e dei sensori, tuttavia con capacità limitate di calcolo e senza supporto nativo per networking wireless.

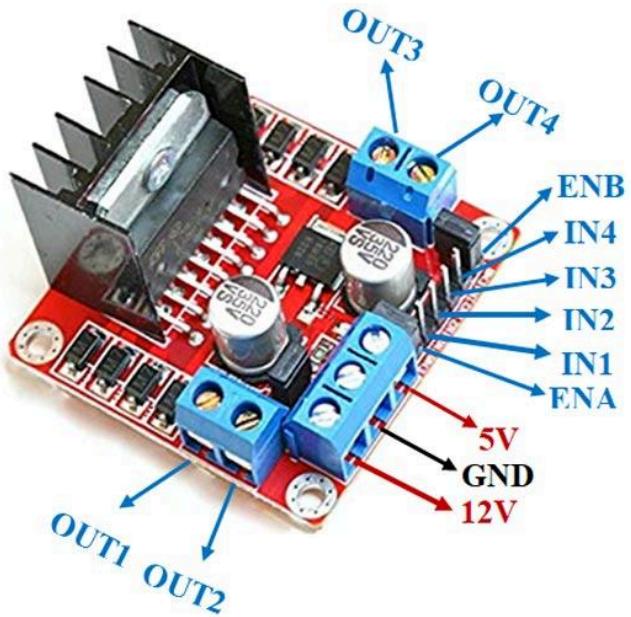
Raspberry Pi, invece, rappresenta la scelta migliore in quanto possiede nativamente il supporto alla connettività wireless; inoltre possiede una maggiore capacità di calcolo, rendendolo più adatto al controllo con istruzioni in tempo reale.

Per la realizzazione del robot è stato utilizzato un kit commerciale, comprensivo di **Motori DC con riduttori, ruote, telaio** e un **portabatterie AA** per l'alimentazione dei motori.

Sono inoltre incluse le viti e i cavi necessari all'assemblaggio, e la ruota girevole che consente al robot di ruotare sul posto.

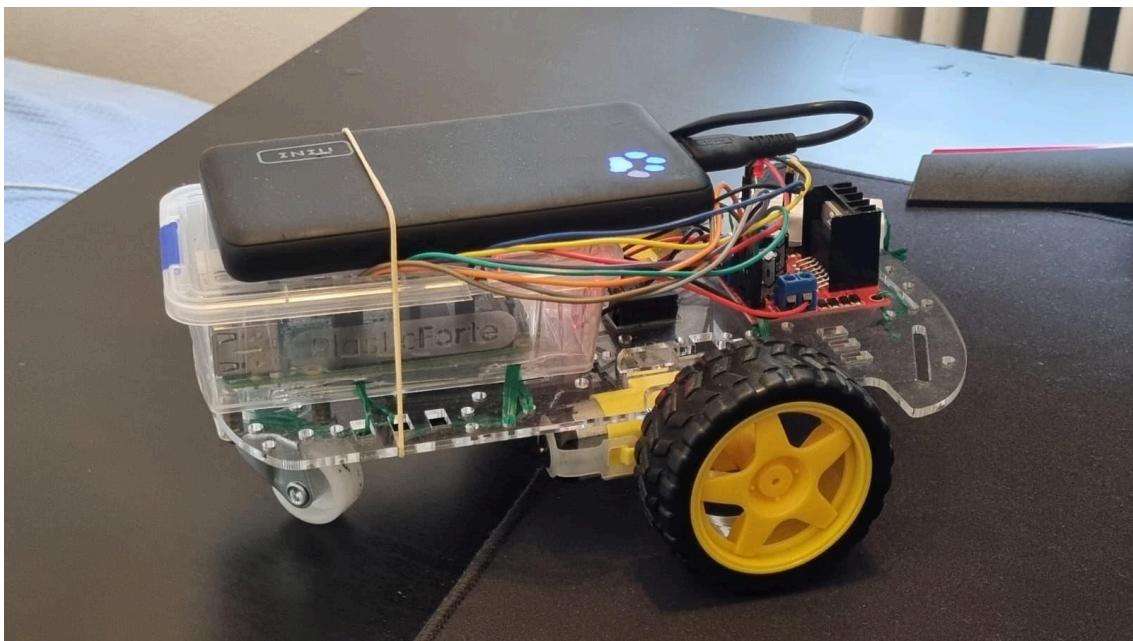


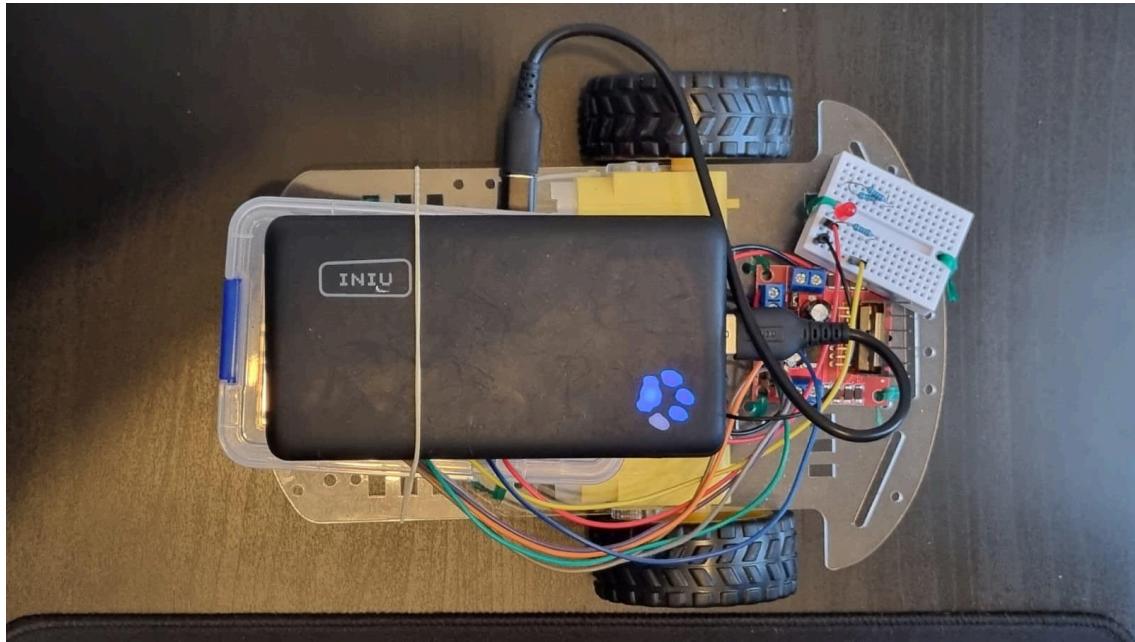
Il driver scelto è la **scheda L298**, un circuito integrato in grado di pilotare i motori DC del robot, controllandone direzione e velocità. La scheda permette il **controllo bidirezionale** dei motori e protegge il sistema da sovraccorrenti, risultando compatibile con il microcontrollore scelto e con i requisiti di potenza dei motori del kit.



Per l'alimentazione del raspberry, è stato utilizzato un powerbank USB. Questo consente di separare l'alimentazione dei motori dal raspberry, evitando così picchi o cali di tensione, che potrebbero danneggiare la board.

Dopo l'assemblaggio e il cablaggio dei componenti, il robot si presenta come mostrato in figura:





Nota: il sistema non è dotato di sensori per il controllo della velocità delle ruote.

Sistema di controllo locale

Per il sistema di controllo locale è stato implementato un **server Python** in grado di ricevere comandi da remoto tramite **WebSocket** e di gestire l'attuazione dei motori, garantendo così il controllo del robot in tempo reale.

Il robot può ricevere i seguenti comandi di base:

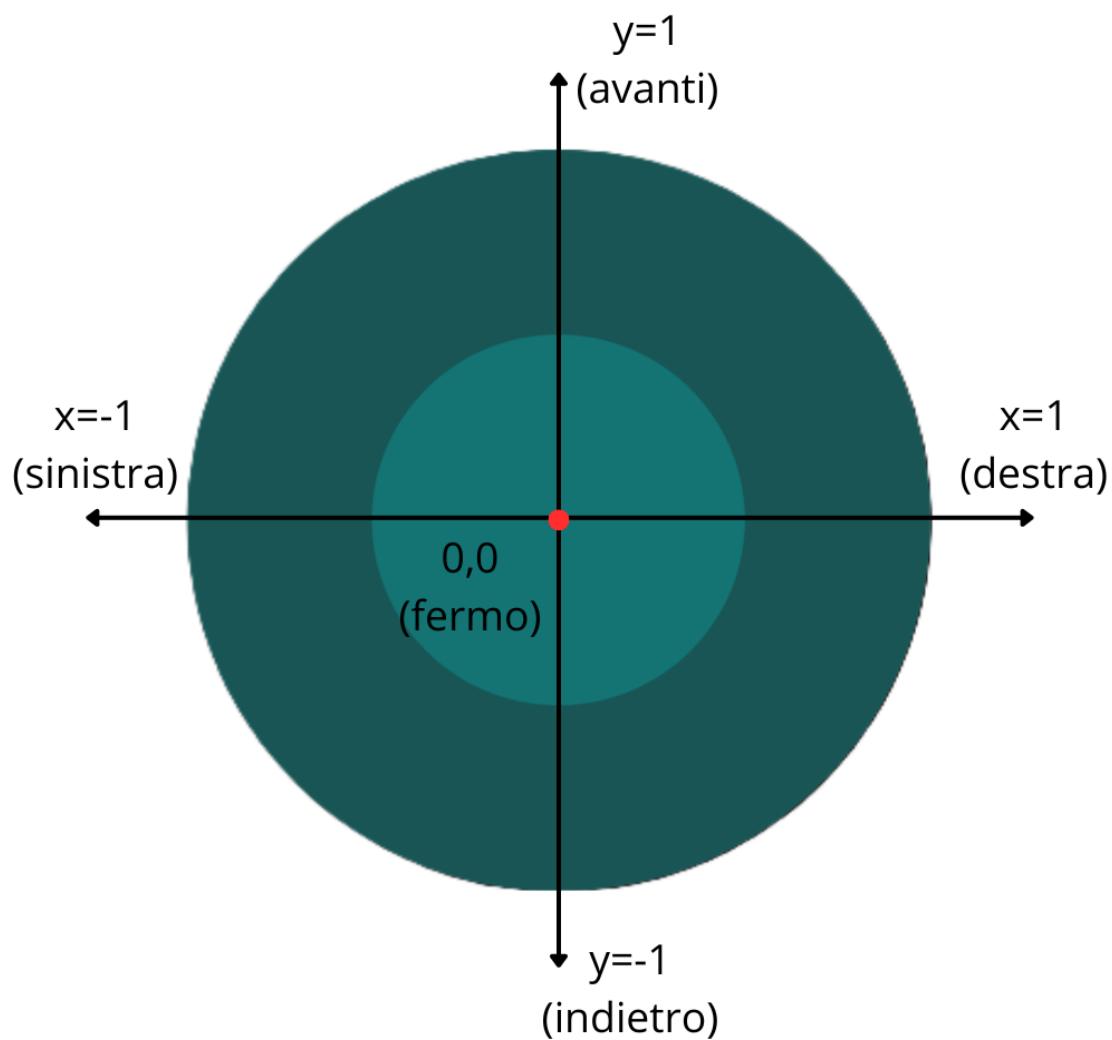
```
COMMANDS = {  
    "forward": cmd_forward,  
    "backward": cmd_backward,  
    "left": cmd_left,  
    "right": cmd_right,  
    "stop": stop_all,  
}
```

Questa struttura consente movimenti semplici: avanzamento, retromarcia e rotazione sul posto. Tuttavia, i movimenti risultano improvvisi e poco fluidi, poco adatti a un controllo fine in tempo reale.

Per migliorare la precisione, sfruttando il supporto della scheda **L298** al controllo tramite **PWM**, è stata implementata una modulazione della velocità dei motori, permettendo movimenti più graduali e controllati.

Per il controllo del robot si è adottato un **modello di input “joystick”** invece di inviare direttamente le velocità dei due motori. In questo modello l'input per il movimento è un **vettore di controllo**, che rappresenta la posizione di un joystick lungo gli assi **X** e **Y**, determinando l'orientamento e la velocità dello spostamento desiderato:

- **Asse Y**: valori compresi tra -1 e 1, dove **-1** corrisponde al movimento all'indietro e **1** all'avanti.
- **Asse X**: valori compresi tra -1 e 1, dove **-1** corrisponde a rotazione a sinistra e **1** a rotazione a destra.



Tutti i valori intermedi rappresentano modulazioni di potenza dei motori.
Ad esempio: X=0,Y=0.5 significa “movimento in avanti con 50% di potenza”.

Il software determina automaticamente quali motori azionare e quanta potenza erogare in base al vettore di controllo joystick. Questo approccio semplifica il controllo da remoto, rendendo i comandi più intuitivi e indipendenti dalle caratteristiche specifiche dei motori.

Criticità riscontrate

La principale criticità riscontrata durante questa fase riguarda il controllo del robot. Infatti, si sperimenta che anche applicando lo stesso segnale di controllo a entrambi i motori, il robot **non segue una traiettoria perfettamente rettilinea**.

Questo comportamento è da imputarsi a piccole differenze tra i motori e le componenti meccaniche, che fanno sì che due motori nominalmente identici possano fornire risposte differenti. Tale fenomeno rappresenta un **vincolo fisico inevitabile** nei sistemi reali.

Sistema di controllo centrale

Il **sistema di controllo centrale** coordina il comportamento del robot, ricevendo informazioni sul suo stato e sulla posizione all'interno dell'ambiente e inviando comandi al sistema locale per guidarne i movimenti.

Funzioni principali:

1. Monitoraggio dello stato del robot
 - Posizione e orientamento all'interno della mappa
 - Stato dei motori e eventuali errori o interruzioni
2. Controllo del movimento
 - Calcolo in tempo reale del vettore di controllo del robot
3. Pianificazione del movimento
 - Calcolo dei percorsi ottimali verso un target
 - Gestione dell'evitamento ostacoli in tempo reale
4. Interfaccia con i sistemi esterni
 - Ricezione di comandi dall'esterno
 - Trasmissione all'esterno informazioni sullo stato e sul completamento dei movimenti

Anche per il server centrale è stato scelto il linguaggio **Python**, in quanto dotato di librerie che semplificano le operazioni di computer vision e consentono di velocizzare la fase di sviluppo.

Monitoraggio della posizione del robot

Uno dei principali problemi di ogni sistema fisico è la coerenza tra il modello interno e lo stato reale del mondo.

Nel sistema VirtualRobot, la posizione del robot viene rappresentata all'interno di una griglia, e il movimento avviene a *step* all'interno di essa. Per mantenere la coerenza tra modello e realtà, questo approccio impone i seguenti requisiti:

- Determinismo dei comandi: a uno stesso input corrisponde sempre la stessa risposta del robot.
- Adesione al percorso: il robot deve essere in grado di muoversi in linea retta e non deviare dalla traiettoria assegnata
- Dimensionamento: la mappa deve essere divisibile in celle della stessa dimensione del robot

Poiché questi requisiti sono **troppe stringenti** o impossibili allo stato attuale del robot fisico, è necessario esplorare approcci alternativi.

Possibili approcci al problema

SLAM: Simultaneous Localization and Mapping

Permette al robot di costruire una mappa dell'ambiente mentre ne stima la propria posizione.

Si tratta di una delle tecniche più utilizzate nell'industria, tuttavia richiede sensori avanzati come LiDAR, che non rientrano nel budget attuale.

Fiducial markers + overhead camera

Una tecnica alternativa e più adeguata ai limiti di budget imposti, è quella di posizionare dei fiducial markers ai bordi della mappa e sul robot, per rilevarne la posizione grazie ad un feedback video trasmesso da una fotocamera overhead (webcam o smartphone).

Per i marker è stata scelta la tecnologia open source ArUco, che fornisce una libreria Python basata su OpenCV per consentire il rilevamento dei tag.



ArUco 42



ArUco 18



ArUco 12



ArUco 27



ArUco 43



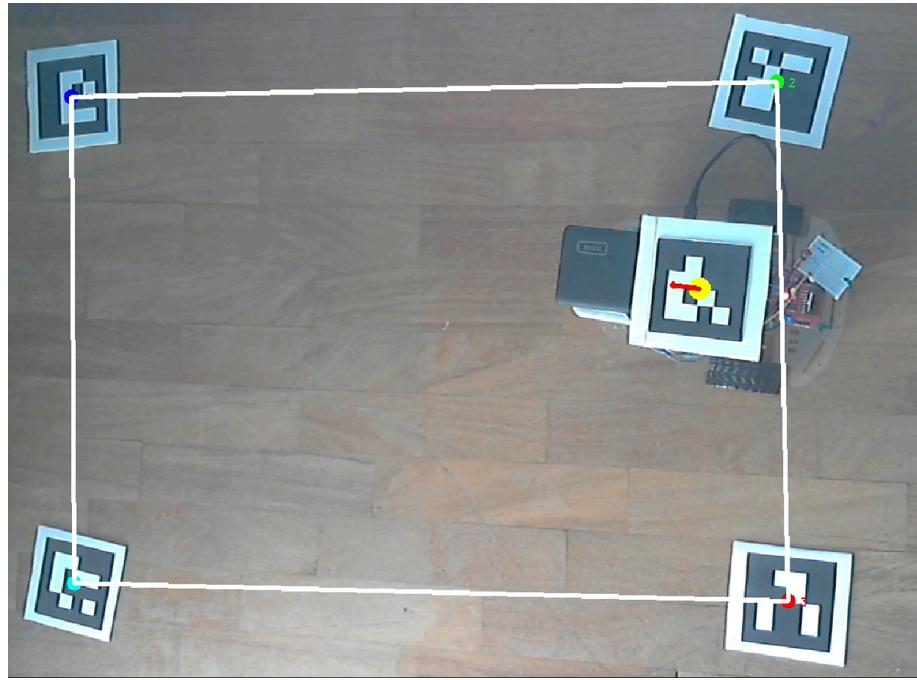
ArUco 5

Grazie a questi strumenti, è stato possibile realizzare il primo sottocomponente software: *VisionSystem*.

Questo componente si occupa di:

- Interfacciarsi con la webcam (inizializzazione e ricezione dello stream video)
- rilevare le posizioni dei marker sullo schermo
- calcolare posizione e orientamento del robot

La seguente schermata è stata ottenuta durante il primo test del sistema:



Nota: in questa schermata la freccia che indica la direzione del robot è invertita.

Sistema di coordinate e tracciamento del robot

Per rappresentare la posizione di oggetti all'interno della mappa viene utilizzato un sistema di coordinate cartesiane (X, Y).

I 4 angoli della mappa vengono individuati grazie a 4 marker ArUco (ID 1,2,3,4). L'ordine con cui vengono collocati i marker non è rilevante: ciò che conta è la loro posizione relativa.

La mappatura marker-coordinate è la seguente:

- marker in alto a sinistra: (0,0)
- marker in alto a destra: (100,0)
- marker in basso a destra: (100,100)
- marker in basso a sinistra: (0,100)

Il software esegue quindi una **trasformazione prospettica** che converte le coordinate acquisite dalla telecamera (sistema basato sui pixel) nel sistema di coordinate normalizzato della mappa, che varia da 0 a 100 sia sull'asse X che sull'asse Y.

La trasformazione effettuata si basa sulla **matrice di trasformazione omografica** che viene calcolata sfruttando la libreria openCV.

La posizione del robot è dunque data dalle coordinate X, Y che corrispondono al centro del marker Aruco con ID=0 all'interno della mappa.

Tuttavia questa informazione da sola non è sufficiente a modellare correttamente il sistema fisico: è necessario introdurre anche l'angolo θ , che indica l'orientamento del robot rispetto all'asse di riferimento della mappa.

In questo modo, la posizione del robot viene espressa come **terna** (X, Y, θ) .

Controllo del movimento del robot

Il punto successivo affrontato durante lo sviluppo è stato quello del controllo del movimento del robot.

Il primo obiettivo è stato quello di permettere al robot di raggiungere un target all'interno della mappa, partendo dalla propria posizione attuale. Per raggiungere questo obiettivo, sono state esplorate due metodologie: **controllo vettoriale diretto** e **controllo PID**.

Controllo vettoriale diretto

Viene tracciato un vettore dalla posizione attuale del robot al target. Il vettore di controllo corrisponde direttamente a questo vettore, che definisce direzione e velocità del movimento attraverso le coordinate *joystick* definite in precedenza.

Si tratta di un approccio semplice ed efficace, tuttavia il robot controllato in questo modo tende ad oscillare lateralmente durante il movimento, a meno che non abbia il target direttamente di fronte a sé.

Controllo PID

Per ottenere un controllo più preciso e stabile, è stato scelto un approccio PID (Proporzionale, Integrale, Derivativo) in grado di regolare dinamicamente l'input di controllo in funzione dell'errore tra la posizione desiderata e quella effettiva.

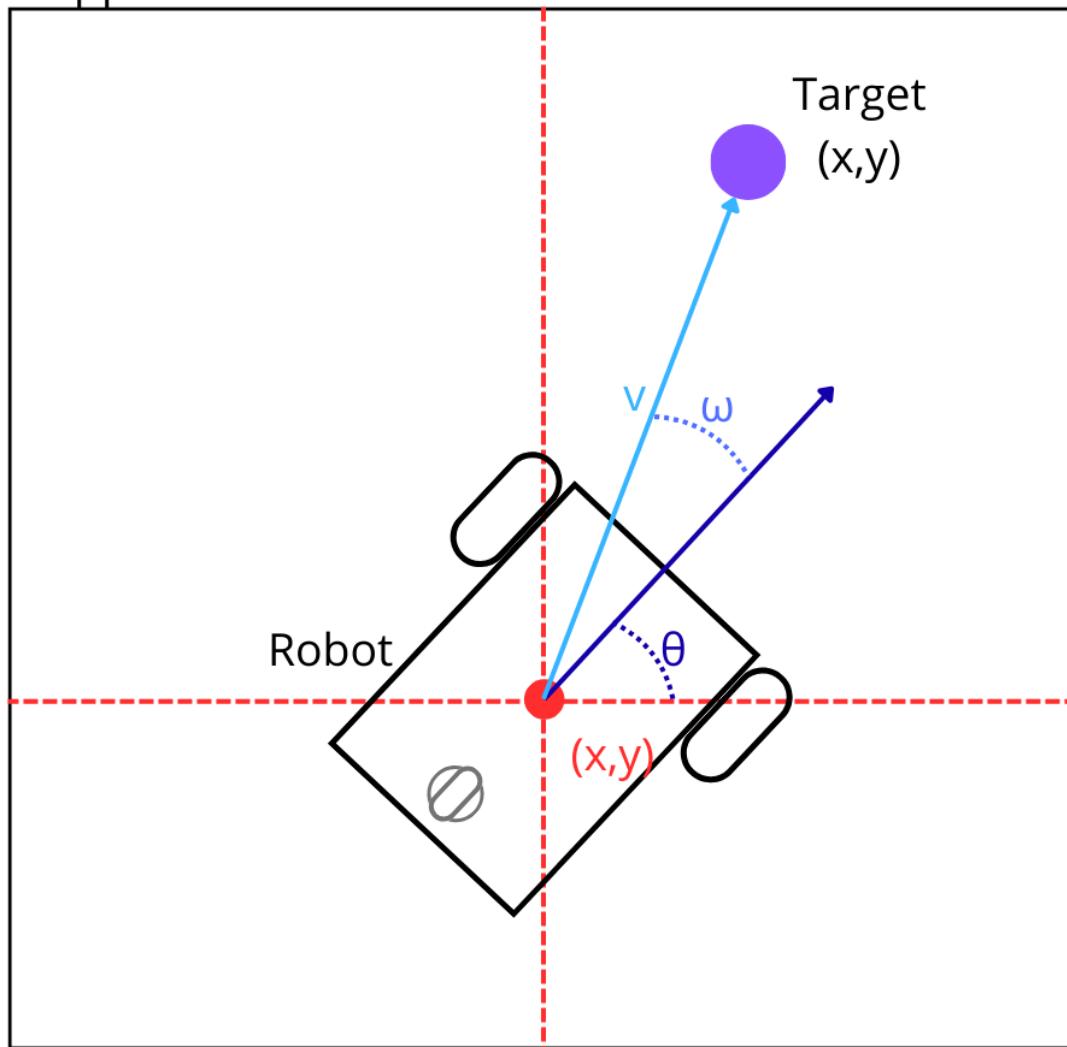
Il PID è stato applicato a due grandezze fondamentali:

- **Distanza v dal target**: calcolata come distanza euclidea fra la posizione corrente del robot e il punto obiettivo
- **Errore angolare ω** : differenza tra l'orientamento del robot e la direzione del target

L'idea alla base di questo approccio è calcolare l'errore in forma polare (v, ω) , per poi calcolare il vettore di controllo (x, y) .

- Dalla grandezza v deriva la lunghezza del vettore di controllo (compresa fra 0 e +1), cioè l'intensità del movimento.
- Dalla grandezza ω deriva l'angolo del vettore di controllo, quindi direzione e verso del movimento.

Mappa



Viene inoltre aggiunto un *bias*: quando l'errore angolare assoluto è maggiore di 100° , v viene azzerato e ω viene saturato a -1 o +1 (in base al segno dell'errore angolare).

In questo modo il robot effettua una rotazione sul posto fino ad allinearsi al target, per poi fare dei microaggiustamenti durante il movimento.

I parametri PID sono stati in seguito calibrati per ottenere una risposta ottimale:

```
# Intensity
PID_INTENSITY_KP = 0.9
PID_INTENSITY_KI = 0.1
PID_INTENSITY_KD = 0.3

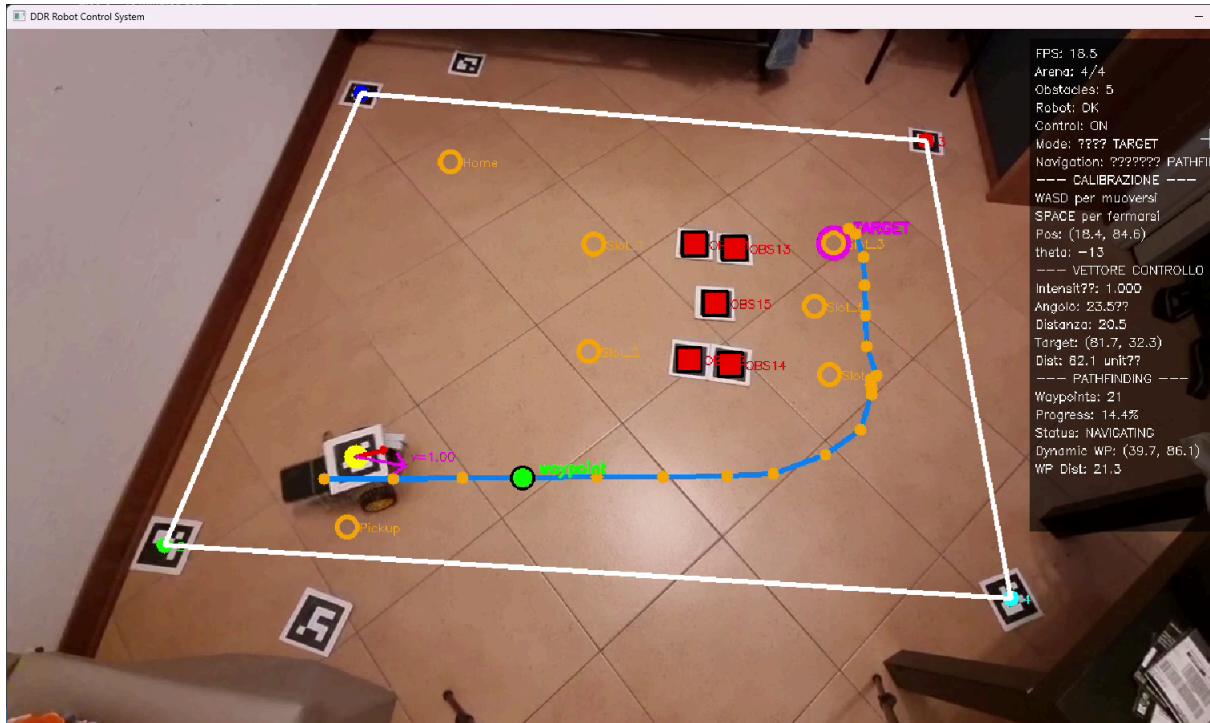
# Angle
PID_ANGLE_KP = 0.8
PID_ANGLE_KI = 0.07
PID_ANGLE_KD = 0.5
```


Il pathfinding viene effettuato con un algoritmo A*, con euristica octile distance, che tiene conto del costo del movimento diagonale. Abbiamo infatti:

```
# Algoritmo A*
STRAIGHT_COST = 1.0          # Costo movimenti cardinali
DIAGONAL_COST = 1.41421356    # sqrt(2) per movimenti diagonali
```

Il movimento del robot viene controllato spostando il target del robot in modo continuo lungo il percorso trovato dall'algoritmo A*. In questo modo il controller PID e il Pathfinding System cooperano per spostare il robot all'interno della mappa, seguendo il percorso migliore fino al raggiungimento del target finale.

Nella schermata seguente è possibile vedere l'algoritmo di pathfinding in azione.



Notiamo nella schermata superiore che i marker hanno ruoli diversi:

- Marker Robot (ID=0)
- Marker Arena (ID=1-4)
- Marker Target (ID=5,6) consentono di indirizzare il robot ad un punto specifico sulla mappa
- Marker Slot (ID=11-15) identificano sia un target che un ostacolo.

Nel file di configurazione è possibile definire i target e gli slot, assegnandogli dei nomi specifici:

```
ROBOT_MARKER_ID = 0
ARENA_MARKER_IDS = [1, 2, 3, 4]
TARGETS ={
    5 : "Home",
    6 : "Pickup",
}
TARGETS_MARKER_IDS = list(TARGETS.keys())
SLOTS = {
    11 : "Slot_1",
    12 : "Slot_2",
    13 : "Slot_3",
    14 : "Slot_4",
    15 : "Slot_5",
}
SLOTS_MARKER_IDS = list(SLOTS.keys())
```

Integrazione QAK

Per integrare il sistema appena creato a *cargoservice*, è necessario costruire un *middleware* in grado di effettuare il **binding** tra due livelli diversi di astrazione: il livello degli attori QAK, ad astrazione maggiore, e quello del software di controllo, più vicino alla macchina.

Il modulo *basicrobot*, fornito dalla software house, è stato originariamente pensato per interfacciarsi con robot differenziali semplici e viene utilizzato in *cargoservice* per governare il movimento del robot virtuale. Tuttavia, tale software assume un modello di movimento lineare vincolato a una mappa a griglia, risultando quindi incompatibile con il funzionamento del robot fisico sviluppato in questo progetto.

Di conseguenza, si rende necessario sostituire *basicrobot* con una nuova versione più adatta alle caratteristiche del robot reale. Come punto di partenza, è utile analizzare i messaggi gestiti dall'attore QAK *basicrobot*, al fine di individuare il protocollo di comunicazione utilizzato per la comunicazione machine-to-machine.

```
Request engage : engage(OWNER, STEPTIME)           // sent by Cargorobot
Reply engagedone : engagedone(ARG) for engage      // sent by basicrobot
Reply engagerefused : engagerefused(ARG) for engage // sent by basicrobot
Dispatch disengage : disengage(ARG)                // sent by Cargorobot

Request pickUpCargo : pickUpCargo(SLOT)            // sent by Cargoservice
Reply pickUpDone:pickUpDone(SLOT) for pickUpCargo // sent by Carborobot
Request moverobot : moverobot(TARGETX, TARGETY)    // sent by Cargorobot
Reply moverobotdone : moverobotok(ARG) for moverobot // sent by basicrobot
Reply moverobotfailed: moverobotfailed(PLANDONE, PLANTODO) for moverobot
// sent by basicrobot

Event alarm : alarm(X)                           // sent by Cargorobot
Dispatch dropDone : dropDone(SLOT)              // sent by Cargorobot
Dispatch setdirection : dir( D ) "D =up|down!left|right" // sent by Cargorobot
Request getenvmap : getenvmap(X)                 // sent by Cargorobot
Reply envmap : envmap(MAP) for getenvmap         // sent by Cargorobot

Dispatch cmd : cmd( MOVE ) "MOVE = a|d|l|r|h"
```

Notiamo innanzitutto che questi messaggi sono di **alto livello** e altamente **technology independent**, questo ci consente di mantenere lo stesso “contratto” comunicativo fra gli attori e semplifica il processo di adattamento del software.

L'unico messaggio che appare incompatibile con il modello di controllo impiegato è *cmd*, tuttavia si tratta di un messaggio utilizzato solo a fini di debug e non necessario alla normale operatività del software.

Viene creata una classe Java helper, *RobotClient*, in grado di comunicare con il controller centrale.

Il mezzo scelto per il trasporto è **websocket**, con messaggi in formato **json**.

I seguenti sono i messaggi inviati dal client Java al server, e seguono il modello **Request**:

```
{ "command": "engage" }
```

```
{ "command": "disengage" }

{ "command": "movetosquare", "x": <int>, "y": <int> }
```

Il server risponde con una fra le seguenti **response**:

```
{ "status": "success", "message": "string", ...optional fields, e.g.
"movement_id": "id" }

{ "status": "error", "message": "descrizione" }
```

Abbiamo inoltre un **evento** asincrono inviato dal server al client, al termine di un movimento:

```
{ "command": "moverobotdone", "movement_id": "id", "status": "success",
"message": "Movimento completato con successo" }
```

Il Client, dopo aver inviato il comando *movetosquare* riceve un *movement id*, che identifica univocamente il comando inviato. Il Client dunque si pone in attesa del messaggio *moverobotdone*, ottenendo così un comportamento sincrono con dei messaggi di natura asincrona.

Per consentire la compatibilità tra il sistema qak, che opera su una griglia di coordinate, e il sistema di controllo del robot, è stata implementata una semplice conversione:

```
MAP_TO_TARGET = {
    (0, 0): "Home",
    (0, 4): "Pickup",
    (1, 1): "Slot_1",
    (4, 1): "Slot_2",
    (1, 3): "Slot_3",
    (4, 3): "Slot_4",
    (5, 2): "Slot_5",
}
```

Espansioni future

Miglioramenti hardware

Per migliorare le prestazioni del robot si potrebbero effettuare le seguenti migliorie al comparto hardware:

- Sensori encoder per il movimento delle ruote: potrebbero migliorare la precisione del controller PID aggiungendo un ulteriore feedback sul comportamento del robot.
- Migliore distribuzione del peso: durante la fase di costruzione il peso dei componenti si è concentrato sulla parte posteriore del robot, riducendo la trazione delle motrici; sistemare questo difetto potrebbe migliorare l'affidabilità e la precisione nei movimenti.
- Ruote con maggiore aderenza al terreno: potrebbero evitare slittamenti.
- Motori e driver con prestazioni più elevate.
- Sensori Lidar o fotocamere stereoscopiche: consentirebbero di mappare l'ambiente circostante al robot in tre dimensioni, fornendo un modello più aderente alla realtà.

Espansioni del software

Sul fronte software, sono possibili sviluppi che renderebbero il robot più autonomo e versatile:

- Implementazione più sofisticata del controller PID.
- Integrazione con ROS (Robot Operating System).
- Algoritmi di SLAM per una mappatura più accurata dell'ambiente.
- Calcolo degli ingombri del robot durante le manovre per consentire un migliore evitamento degli ostacoli.
- Integrazione di Machine Learning per il riconoscimento di oggetti e ostacoli.
- Algoritmi di line following per una maggiore aderenza al percorso prefissato.