Università degli Studi di Palermo

DOCUMENTAZIONE

PROGETTO RAINBOW

Michele Grimaldi – Vito Di Grigoli

Prof. Antonio Chella

Ing. Francesco Lanza

Sommario

Introduzione

Il nostro progetto mira alla realizzazione di un robot che abbia come obiettivo la raccolta di alcune palline sparse intorno l'ambiente simulato seguendo un ordine specifico. In particolare, abbiamo scelto di basare la nostra priorità sui colori dell'arcobaleno utilizzando i cinque colori di Newton anziché i sette dell'arcobaleno completo per semplificare il mondo, già abbastanza complesso da renderizzare e da eseguire in fase di simulazione.

Si è scelto di utilizzare la piattaforma Webots perché open source e quella utilizzata durante le esercitazioni svolte dall'ingegnere Lanza.

Il compito del robot è quello di andare a recuperare le palline intorno alla mappa ed evitare ostacoli qualora si presentino. Una volta recuperata ciascuna pallina, sarà obiettivo del robot portarla in un punto specifico della mappa in cui sarà presente uno scivolo a forma di imbuto. All'interno di questo scivolo saranno raggruppate le palline in ordine generando come effetto finale un arcobaleno.

Anatomia del robot

Si è scelto di non utilizzare i robot predefiniti di Webots, ma costruirne uno manualmente utilizzando gli assets che la piattaforma mette a disposizione.

Per quanto riguarda i sensori utilizzati, abbiamo scelto di sfruttare:

- Unità inerziale. Simula un sensore che misura gli angoli relativi di rollio, beccheggio e imbardata. Lo abbiamo utilizzato soprattutto per calcolare l'angolo di imbardata del robot rispetto agli assi di riferimento di Webots.
- GPS. Simula un sensore di posizionamento che misura la posizione assoluta nel sistema di coordinate Webots.
- Camera. Simula una tipica telecamera RGB in cui è stata attivata la funzionalità di riconoscimento dei colori.
- Sensori di distanza, di cui due utili per individuare gli ostacoli e uno per dare l'input al braccio di chiudersi nel momento in cui è in possesso della pallina.

Per quanto riguarda la struttura, abbiamo scelto di creare:

- Tre bracci, di cui due fissi per non far scappare la pallina e uno che ruota grazie all'utilizzo di una cerniera che consente solo un movimento rotatorio attorno a un dato asse. Nello specifico abbiamo scelto come asse su cui far ruotare il motore rotazionale l'asse Z.
- Un corpo, per dare un aspetto visivo al robot.
- Quattro ruote motrici con un motore rotazionale in ognuna di esse per permettere al robot di muoversi.

Struttura dell'ambiente

Si è scelto di utilizzare le impostazioni di default che mette a disposizione Webots e di cambiare soltanto il parametro basicTimeStep, settandolo a un valore pari a 16. In questo modo verrà aumentata l'accuratezza e la stabilità della simulazione, specialmente per i calcoli fisici e il rilevamento delle collisioni.

L'ambiente è così composto:

- Cinque palline di colore differente al fine di ottenere un effetto arcobaleno al termine della raccolta di esse. Abbiamo inoltre deciso di settare il parametro recognitionColors sul colore corrispondente per consentirne il riconoscimento alla camera.
- Quattro muri, per delimitare l'ambiente di azione del robot.
- Una piattaforma, utile per portare il robot davanti la rampa da qualsiasi direzione esso provenga.
- Una rampa, in cui è stato posizionato uno scivolo a forma di imbuto adibito alla raccolta delle palline.
- Un ostacolo da far evitare al robot durante i suoi spostamenti.

Controller

Si è scelto di impostare un TimeStep uguale al basicTimeStep dell'ambiente per sincronizzare le azioni del robot all'esecuzione della simulazione. In particolare, abbiamo richiamato la funzione wb_robot_step(TimeStep) ogni qual volta venisse eseguito un movimento del robot.

Per far svolgere le varie mansioni al robot, abbiamo creato cinque diversi stati:

1. **Detect_ball**. Stato iniziale del robot in cui inizia a ruotare su sé stesso, in senso antiorario, fintanto che non individua la pallina dello stesso colore passato alla funzione. Una volta individuata, viene centrata rispetto alla visuale della camera con un grado di precisione di ±4px.

2. **Grabs_ball**. Il robot si presta a raccogliere la pallina davanti a sé evitando gli ostacoli qualora fossero presenti lungo il tragitto. Una volta che il robot percepisce che è in possesso della pallina, si ferma e chiude il braccio catturandola.
3. **Move_to_platform**. Acquisisce le coordinate del punto di partenza del robot tramite GPS e calcola la distanza e la direzione in cui il robot deve ruotare per raggiungere la piattaforma tramite odometria, evitando ostacoli qualora fossero presenti.
4. **Move_to_ramp**. Simile allo stato precedente, ma avente come obiettivo finale il raggiungimento della rampa.
5. **Release_ball**. Una volta raggiunta la rampa, la pallina viene rilasciata e fatta scivolare sul condotto a forma di imbuto. Infine, il robot torna indietro fin quando non raggiunge di nuovo la piattaforma.

Per aumentare la leggibilità e il riutilizzo del codice, sono state implementate delle funzioni in grado di eseguire task elementari come:

- move_robot;
- move_arm;
- enable_sensors;
- recognition_color;
- is_that_color;
- odometry;
- goal_oriented;
- distance_to_point;
- avoid_obstacle.

void move_robot (const char *command)

Questa funzione permette di settare la velocità di ogni singola ruota del robot inizializzandola a velocità iniziale di 0.5 rad/s.

In base alla stringa che viene passata alla funzione, il robot può eseguire i seguenti movimenti:

- **LEFT**: per far ruotare il robot a sinistra, viene settata la velocità delle ruote di sinistra a -0.5 rad/s, lasciando invariate le velocità delle ruote di destra.
- **RIGHT**: per far ruotare il robot a destra, viene settata la velocità delle ruote di destra a -0.5 rad/s, lasciando invariate le velocità delle ruote di sinistra.

- **GO STRAIGHT**: per muovere il robot in avanti viene settata la velocità di tutte e quattro le ruote a 3.0 rad/s.
- **GO BACK**: per muovere il robot indietro viene settata la velocità di tutte e quattro le ruote a -3.0 rad/s.
- **STOP**: per fermare il robot viene settata la velocità di tutte e quattro le ruote a 0 rad/s.

void move_arm (const char *command)

Questa funzione permette di settare il movimento del braccio meccanico.

In base alla stringa che viene passata alla funzione, il robot può eseguire i seguenti movimenti:

- **CLOSE**: viene fatto ruotare il braccio di 90 gradi tramite il motore rotazionale, impostando la variabile di rotazione a -1.57 radianti, quindi equivalente a -π/2.
- **OPEN**: viene riportato il braccio alla posizione iniziale tramite il motore rotazionale, impostando la variabile di rotazione a 0 radianti.

void enable_sensors (void)

Questa funzione permette di attivare tutti i sensori presenti nel robot quali:

- GPS;
- unità inerziale;
- sensori di distanza;
- motore rotazionale del braccio e delle ruote;
- camera.

Viene utilizzata all'interno della funzione main permettendo l'utilizzo di tutti i sensori fin da subito.

char* recognition_color(const double *color)

Questa funzione permette di riconoscere il colore passato in formato RGB e ritornare la stringa del colore in base al valore dei tre canali passati alla funzione. I colori che riesce a riconoscere sono:

- rosso;
- giallo;
- verde;
- blu;

- viola.

Abbiamo effettuato le dovute proporzioni per convertire i parametri dei tre canali visto che i colori andavano in un intervallo [0,1].

const WbCameraRecognitionObject* is_that_color (const double *color)

Questa funzione permette di cercare, all'interno della lista degli oggetti riconosciuti dalla camera, la palla del colore specificato e restituisce la sua posizione all'interno della lista. Qualora il colore della pallina specificata non fosse presente, la funzione restituirebbe un puntatore a NULL.

const double odomentry (const double Xf, const double Zf)

Questa funzione acquisisce le coordinate del robot e calcola la distanza dalla meta da raggiungere. Infine, ritorna l'angolo di rotazione che il robot deve effettuare per mettersi in direzione dell'obiettivo.

void goal_oriented (const double theta_primo)

Questa funzione permette di prendere l'angolo di imbardata del robot, calcolarne la differenza con theta_primo, passato come parametro alla funzione e risultato della funzione odometry, e, infine, ruotare il robot in senso orario o antiorario fintanto che la differenza tra i due angoli sia trascurabile.

Per ottimizzare il verso di rotazione, verrà ruotato il robot in verso antiorario se la differenza fra i due angoli è positiva o nel senso opposto qualora la differenza sia negativa, diminuendo così il tempo che il robot impiega a orientarsi verso l'obiettivo.

double distance_two_points (const double X0, const double Z0, const double Xf, const double Zf)

Questa funzione permette di calcolare la distanza tra due punti tramite la distanza di Euclide. Vengono passati alla funzione le coordinate del punto iniziale e finale e ritorna la distanza fra di essi.

void avoid_obstacle (const double *color)

Questa funzione permette di percepire ed evitare gli ostacoli durante il tragitto tramite l'utilizzo di sensori di distanza. Il valore di ritorno di questi sensori permette di individuare se l'ostacolo si trova a destra o sinistra del robot tramite variabile booleana.

In base alla posizione dell'ostacolo, il robot effettua dei movimenti che permettono di aggirarlo e continuare l'azione che stava effettuando precedentemente.

Qualora il robot, tramite sensore, percepisca la pallina all'interno delle braccia, verrà diretto verso la piattaforma. In caso contrario ritornerà allo stato iniziale, ovvero la ricerca della pallina stessa.

Conclusioni

Possiamo concludere che, nonostante le svariate problematica legate all'utilizzo della piattaforma poiché abbastanza pesante durante l'esecuzione della simulazione, siamo riusciti a raggiungere i vari obiettivi che ci eravamo prefissati all'inizio del progetto. Il robot riesce a concludere tutti i task che gli vengono sottoposti con un alto grado di precisione.

Abbiamo pensato che questo robot potrà essere utilizzato a scopi educativi, affiancandolo ad esempio a un bimbo durante le sue ore di gioco. Esso potrà insegnargli a riconoscere i vari colori e ordinarli secondo un ordine specifico.

Si potrebbe pensare di continuarne lo sviluppo cercando di implementare un'empatia al robot. Ciò getterebbe le basi per una comunicazione più efficiente mentre cerca di interagire col bambino.

Sono stati effettuati di recente degli esperimenti da parte della Columbia University di New York in cui un gruppo di ricercatori è riuscito a far prevedere a un robot i comportamenti di un secondo robot soltanto osservandolo. Si tratta comunque di una forma primitiva di empatia, ma la capacità di predire il comportamento di un altro robot (o persona) in assenza di comunicazione verbale potrebbe aprire nuove strade allo sviluppo in questo settore, considerando anche i problemi etici importanti legati ad esso.

Anticipare il pensiero dell'uomo, manipolare l'uomo stesso ed essere in grado di prendere decisioni autonome basate sulla predizione meritano un'attenta riflessione di tipo etico e filosofico.