Webots

Raul Luizaga

October 2022

1 Webots

1.1 Installazione su Ubuntu

- Andare sul sito ufficiale e scaricare .deb
- Aprire il terminale dalla cartella in cui si è scaricato il file .deb
- digitare i seguenti comandi :
  sudo dpkg -i webots_2022a_amd64.deb

Potresti incontrare i seguenti errori :

- dpkg: problemi con le dipendenze impediscono la configurazione di webots:
- webots dipende da ffmpeg

Possibili risoluzioni :

- Il pacchetto ffmpeg non è installato:
  - *sudo apt update*
  - *sudo apt install ffmpeg*
- se non vanno i comandi di sopra sopra
  - *sudo apt –fix-broken install*
- Webots dipende da libssh-devm, tuttavia Il pacchetto libssh-dev non è installato:
  - *sudo apt-get install libssh-dev*
- webots dipende da libfox-1.6-dev, tuttavia Il pacchetto libfox-1.6-dev non è installato
  - *sudo apt-get install libfox-1.6-dev*

1.2 Creazione di un Robot P-Rob3

Nella sezione Simulation View fare:

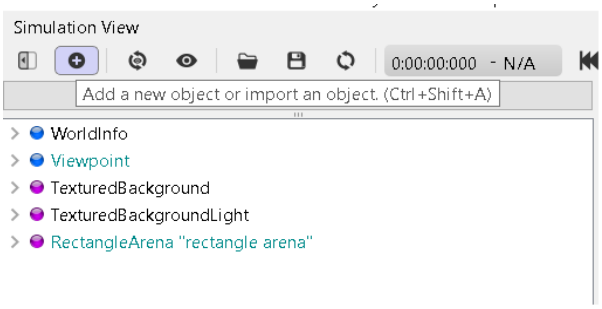

Figure 1: aggiungi

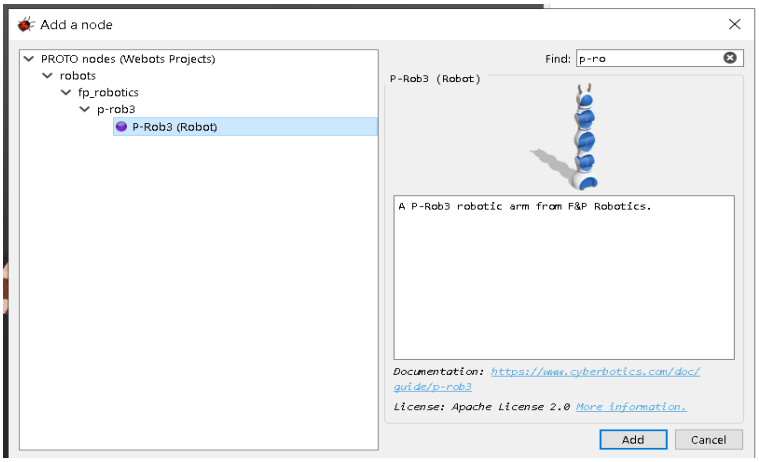

Figure 2: Cercare P-ROB3 nella sezione find e confermare

Dovrebbe essersi generato il nuovo oggetto nella Scene tree sinistra.

Un Proto node è definito in un Proto file il quale è un file di testo con l'estensione proto. La definizione del Proto node elenca i campi e definisce come questi campi influiscono sull'oggetto sotto-stante che è definito come utilizzando nodi di base e/o proto. Per accedere al file con estensione proto :

- selezionare P-rob3
- tasto destro
- selezionare View PROTO source

Per questa simulazione iniziale non ci serve modificare il Proto file. Il Proto node rappresenta solo l'oggetto nell'ambiente non posso comunicare con esso, per questo motivo deve essere convertito in un Base node.

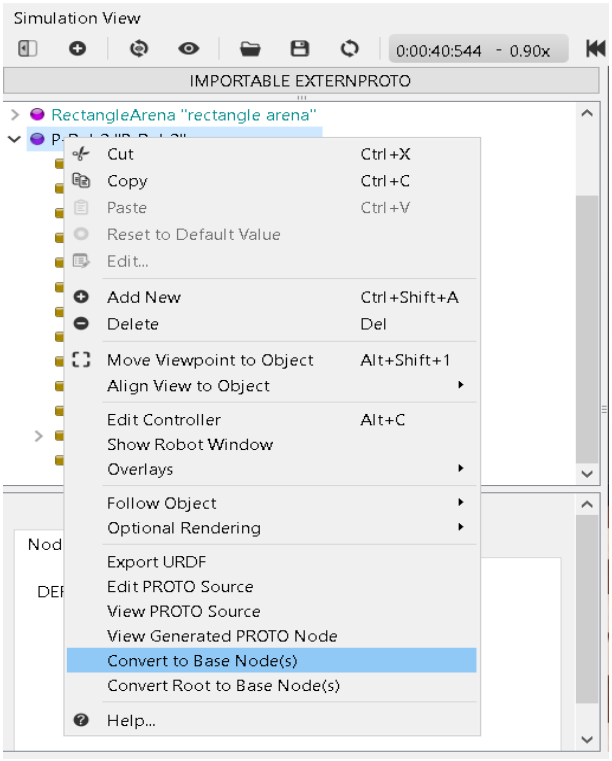

Figure 3: Conversione da Proto node a Base node

1.3 Posizione dei componenti all'interno dei nodi di configurazione

La struttura del P-rob3 è formato da vari Hinge Joint ovvero giunti che permettono il movimento(in questo caso movimento di tipo rotatorio) tra elementi solidi collegati all'estremità del giunto. Hinge Join ha al suo interno:

1. HingeJointParameters: posizione, assi x,y,z, ecc...
2. Device: contente i parametri del motore ed il sensore di posizione
3. endPoint Solid : definisce un altro giunto e la posizione di riferimento rispetto al giunto precedente o la opportuna base di appoggio del robot.

All'interno di webots sono presenti alcuni nodi che racchiudono specifiche strutturali, mentre altri racchiudono altri nodi come il nodo Children che è

utilizzato per includere altri HingeJoint. La costruzione dei vari componenti del Robot avviene dal basso fino ad arrivare alle pinze.

Per poter accedere alle varie parti mobili in Webots devi accedere tramite nodi che si estendono a cascata come l'esempio qui sotto :

- **children** (primo motore)
- HingeJoint
- EndPointSolid
- **children** (secondo motore)
- HingeJoint
- EndPointSolid
- **children** (terzo motore)
- HingeJoint
- EndPointSolid
- **children** (quarto motore)
- HingeJoint
- EndPointSolid
- **children** (quinto motore)
- HingeJoint
- EndPointSolid
- **children** (sesto motore)
- Group
- **children**
- DEF GRIPPET Solid
- **children** (motodi pinza destra e sinistra)
- DEF RIGHT_FINGER HingeJoint and DEF LEFT_FINGER HingeJoint con all'interno i solo endPointSolid

1.4 Creazione di un Controllore

I robot in Webots non sono creati con un controllore integrato, di conseguenza è nostro lavoro inserirne uno. C'è la possibilità di utilizzare un gran numero di linguaggi di programmazioni differenti.

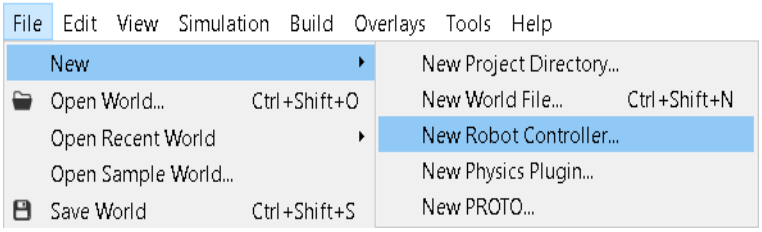

Figure 4: Creazione controllo in Webots

Eseguire la sequenza di istruzioni eseguite nell'immagine e settare il controllo in base ai propri gusti. Nel mio caso ho utilizzato Visual Studio Code. Una volta definito un controllo per il robot devo dire al robot di usare quello specifico controllo. Entrare nel nodo P-ROB3 e cercare un nodo chiamato "controller", inizialmente sarà settato come *generic*. La cosa che dobbiamo fare è accedere al nodo e fare Select, di conseguenza appariranno una serie di controlli e tra questi ci sarà quello che abbiamo creato al passo precedente. Dopo averlo selezionato schiacciare l'opzione edit, questo servità per aprire il codice in Webots e poter eseguire operazioni di Build e Build Clean.

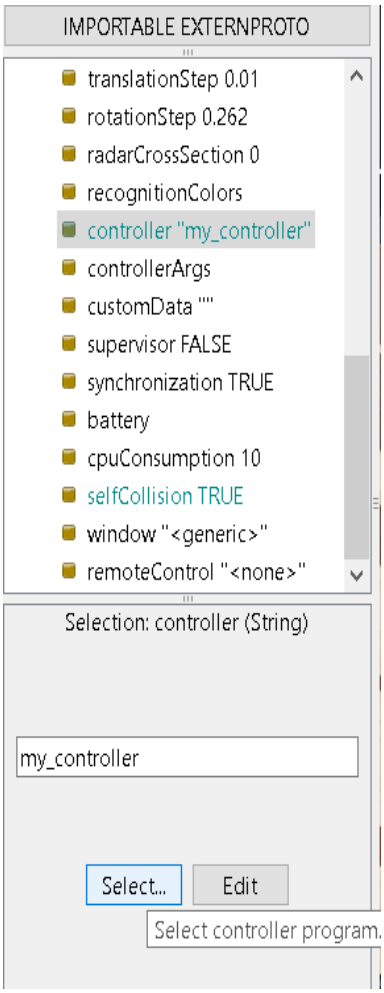

Figure 5: Selezione controllo per un robot

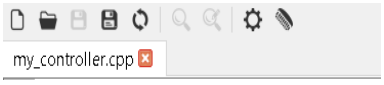

Figure 6: Opzioni per il controllore

1.5 Codice controllore

Per il controllore del P-Rob3 ho deciso di creare un controllore in *c++*. I principali obiettivi iniziali sono stati :

1. Come accedere ai motori, costruirli e manipolarli

2. Come accedere ai sensori posizionati sui motori ed accedere ai loro dati

3. Quali librerie utilizzare

Nell'immagine sottostante sono presenti le line di codice delle librerie e delle variabili globali utilizzate per implementare il codice.

```
#include <webots/Robot.hpp>
#include <webots/Motor.hpp>
#include<webots/PositionSensor.hpp>
#include <unistd.h>
#define TIME_STEP 64 // tempo di clok
#define MAX_SPEED 1.74 // velocità massima per tutti i motori
#define MAX_SPEED1 1 // velocità massima pinze  1.0472

using namespace webots;


const double PI = 3.14159265358979323846264338327950288419716939937510582O9;
```

Figure 7: Librerie e variabili globali utilizzate

Nell'immagine sottostante possiamo vedere come creare l'istanza dell'intero Robot.

```
Robot *robot = new Robot();
```

Figure 8: istanza P-Rob3

Nell'immagine sottostante possiamo vedere come creare le istanze dei motori.In questo caso il costruttore necessità di una stringa come input che deve coincidere con il nome dell'oggetto motore all'interno del nodo *device*

```
Motor* motore_1 = new Motor("motor 1"); // tutt
// Nome children "link_1_2 , dentro endPoint"
Motor* motore_2 = new Motor("motor 2");
// nome children "link_2_3 , dentro endPoint"
Motor* motore_3 = new Motor("motor 3");
// nome children "link_3_4 , dentro endPoint"
Motor* motore_4 = new Motor("motor 4");
// nome children "link_4_5 , dentro endPoint"
Motor* motore_5 = new Motor("motor 5");
// nome children "link_5_6 , dentro endPoint"
Motor* motore_6 = new Motor("motor 6");

  /*in questo ultimo children "link_6_top"
 abbiamo 8 TouchSensor con id "button i", i = 0

Oltre a questo ha un nodo Group che contiene un
dentro il quase sono presenti "DEF RIGHT_FINGER
device -> motore e sensore posizione 7) e "DEF
contiene device -> motore e sensori posizione 8
c'è anche un sensore alla base  chiamato delle
sensor"
MOTORI DELLE PINZE
*/
Motor* motore_pdx = new Motor("motor 7");
Motor* motore_psx = new Motor("motor 7 left");
```

Figure 9: Istanza motori

Nell'immagine sottostante possiamo vedere come creare le istanze dei sensori di posizione posizionato presso i motori.In questo caso il costruttore necessità di una stringa come input che deve coincidere con il nome dell'oggetto sensore motore all'interno del nodo *device*. Per i sensori si necessita di abilitare la sincronizzazione con il tempo di simulazione di Webots.

```
// SENSORI DI POSIZIONE ASSOCIATI AI MOTORI
PositionSensor* pos_1 = new PositionSensor("motor 1 sensor");
pos_1->enable(TIME_STEP);

PositionSensor* pos_2 = new PositionSensor("motor 2 sensor");
pos_2->enable(TIME_STEP);

PositionSensor* pos_3 = new PositionSensor("motor 3 sensor");
pos_3->enable(TIME_STEP);

PositionSensor* pos_4 = new PositionSensor("motor 4 sensor");
pos_4->enable(TIME_STEP);

PositionSensor* pos_5 = new PositionSensor("motor 5 sensor");
pos_5->enable(TIME_STEP);

PositionSensor* pos_6 = new PositionSensor("motor 6 sensor");
pos_6->enable(TIME_STEP);

PositionSensor* pos_pdx = new PositionSensor("motor 7 sensor");
pos_pdx->enable(TIME_STEP);

PositionSensor* pos_psx = new PositionSensor("motor 7 left sensor");
pos_psx->enable(TIME_STEP);
```

Figure 10: Istanza sensori di posizione

Nell'immagine sottostante è presente un parte di codice dove ho creato variabili che rappresentano i limiti di velocità per ogni motore di cui è munito il P-Rob3.

```
double motore_1_vmax = MAX_SPEED;
double motore_2_vmax = MAX_SPEED;
double motore_3_vmax = MAX_SPEED;
double motore_4_vmax = MAX_SPEED;
double motore_5_vmax = MAX_SPEED;
double motore_6_vmax = MAX_SPEED;
double motore_pdx_vmax = MAX_SPEED1;
double motore_psx_vmax = MAX_SPEED1;
```

Figure 11: Velocità massime

Nell'immagine sottostante è presente un parte di codice dove ho creato variabili che rappresentano la posizione che desidero per ogni Robot nel suo ambiente di lavoro. Ho preferito inserire i gradi che poi andrò a convertire in radianti per metterlo in ingresso alla funzione setPosition() dei motori.

```
double pos_1_goal = 100 * (PI/180); // massimo 170 gradi
std::cout<<"posizione desiderata : "<< pos_1_goal<<std::endl;
double pos_2_goal = 10 * (PI/180); // massimo 109.5 gradi
double pos_3_goal = 60 * (PI/180); // massimo 114.5 gradi
double pos_4_goal = 90 * (PI/180); // massimo 170 gradi
double pos_5_goal = 60 * (PI/180); // massimo 114.5 gradi
double pos_6_goal = 90 * (PI/180); // massimo 169.5 gradi
double pos_pdx_goal = 60 * (PI/180); // massimo 60 gradi
double pos_psx_goal = 60 * (PI/180); // massimo 60 gradi
```

Figure 12: Posizioni Obiettivo

Nell'immagine sottostante è presente un parte di codice dove ho settato le condizioni iniziali di posizione e di velocità .

```
motore_1->setPosition(INFINITY);
motore_1->setVelocity(0.0);

motore_2->setPosition(INFINITY);
motore_2->setVelocity(0.0);

motore_3->setPosition(INFINITY);
motore_3->setVelocity(0.0);

motore_4->setPosition(INFINITY);
motore_4->setVelocity(0.0);

motore_5->setPosition(INFINITY);
motore_5->setVelocity(0.0);

motore_6->setPosition(INFINITY);
motore_6->setVelocity(0.0);

motore_pdx->setPosition(INFINITY);
motore_pdx->setVelocity(0.0);

motore_psx->setPosition(INFINITY);
motore_psx->setVelocity(0.0);
```

Figure 13: Velocità iniziali e Finali

Nell'immagine sottostante è presente un parte di codice dove ho settato delle variabili booleani di controllo settate a false per determinare la fine e l'inizio delle azioni eseguite dai motore del P-Rob3.

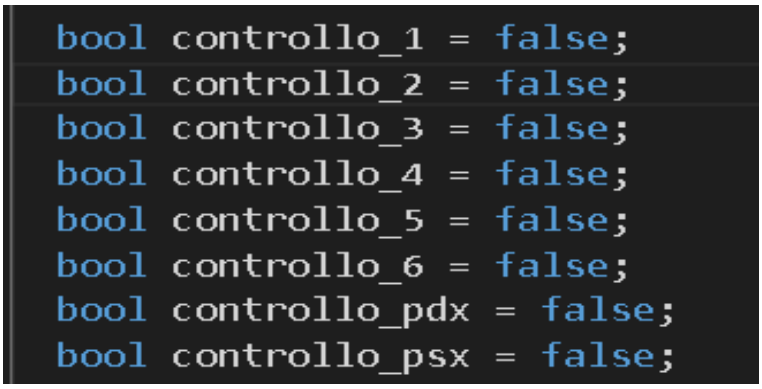

Figure 14: Velocità iniziali e Finali

Nell'immagine sottostante è presente un parte di codice che viene eseguito per ogni motore all'interno di un ciclo while con condizione di uscita la fine della simulazione. Alla fine di ogni azione di un motore si aspetta per un secondo e poi viene eseguita l'operazione sul motore successivo.

```
// Set position credo dia coordinate assolute e non relative, il discorso vale anche per valori negativi
if(controllo_1 == false){
  motore_1->setVelocity(motore_1_vmax);
  motore_1->setPosition(pos_1_goal); // motore 1 può spostarsi a massimo 2.95903 (170 gradi) (rotazione intorno a se stesso )
  std::cout<<"posizione del motore 1 gradi : "<< pos_1->getValue()*(180/PI)<<std::endl;
  std::cout<<"posizione del motore 1 radianti  : "<< pos_1->getValue()<<std::endl;

  if( ( abs(pos_1->getValue() - pos_1_goal )< 0.01)){
    sleep(1);
    controllo_1 = true;

    }
}
```

Figure 15: controllo raggiungimento posizione

1.6 Risultati della simulazione

Nelle seguenti immagini saranno presentate le prime simulazioni che ho eseguito per mostrare il funzionamento del P-Rob3. I motori del robot hanno eseguito i seguenti movimenti:

1. Motore_1 nella posizione di100°
2. Motore_2 nella posizione di 10°
3. Motore_3 nella posizione di60°

4. Motore_4 nella posizione di 90°
5. Motore_5 nella posizione di60°
6. Motore_6 nella posizione di 90°
7. Motore_pdx nella posizione di60°
8. Motore_psx nella posizione di 60°

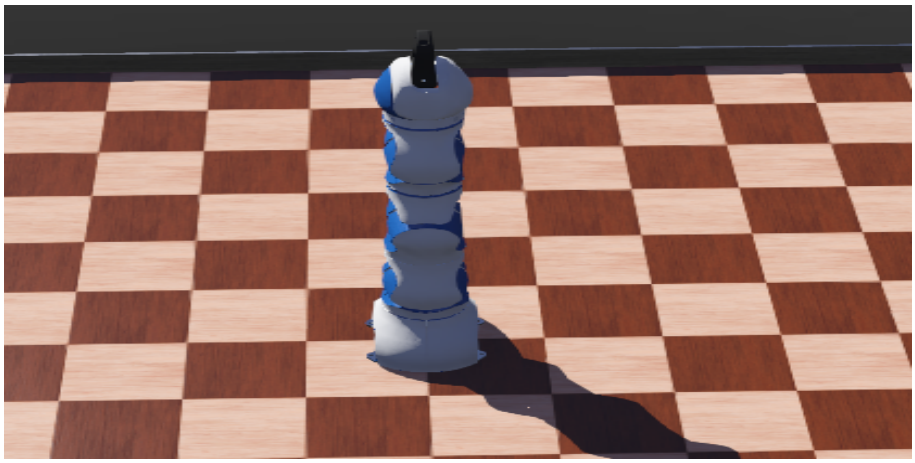

Figure 16: Posizione iniziale del P-Rob3

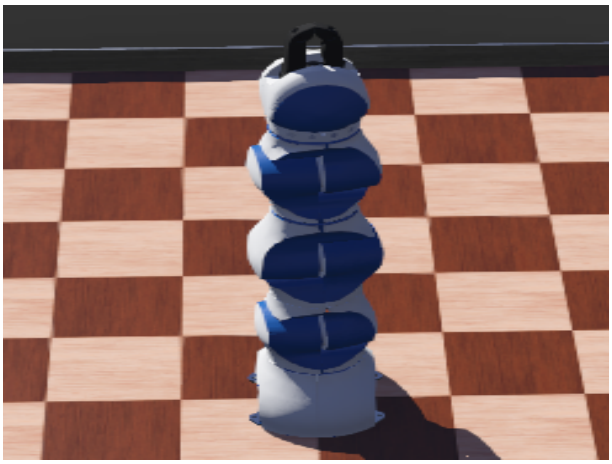

Figure 17: Fine azione motore1 del P-Rob3

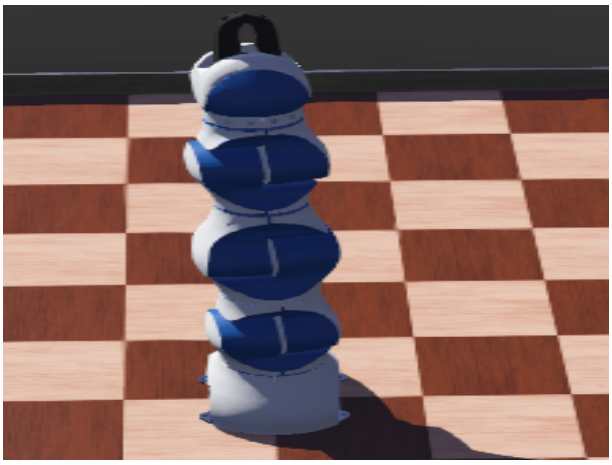

Figure 18: Fine azione motore2 del P-Rob3

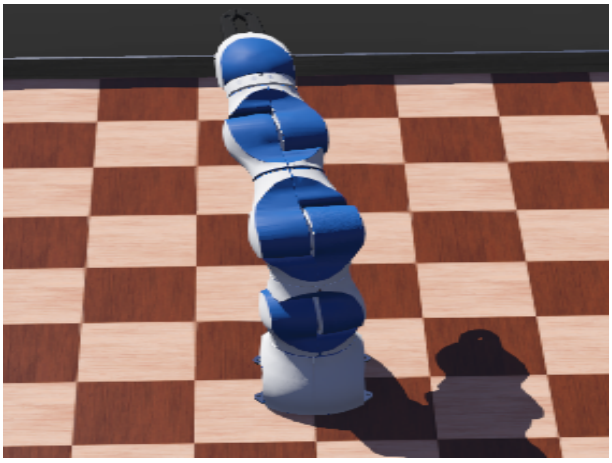

Figure 19: Fine azione motore3 del P-Rob3

Figure 20: Fine azione motore4 del P-Rob3

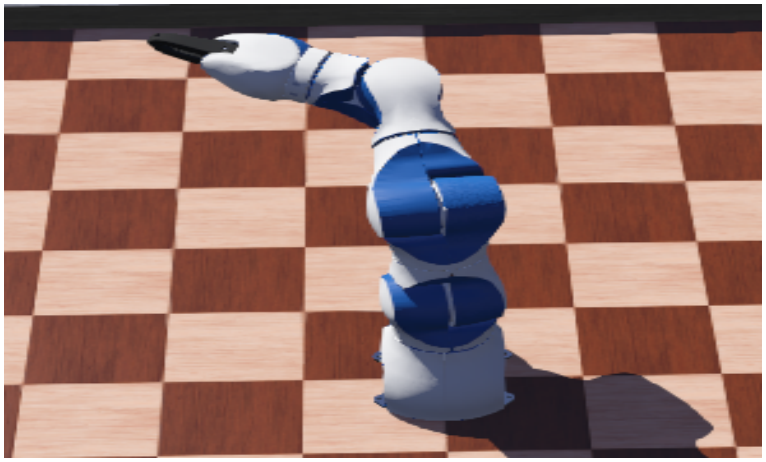

Figure 21: Fine azione motore5 del P-Rob3

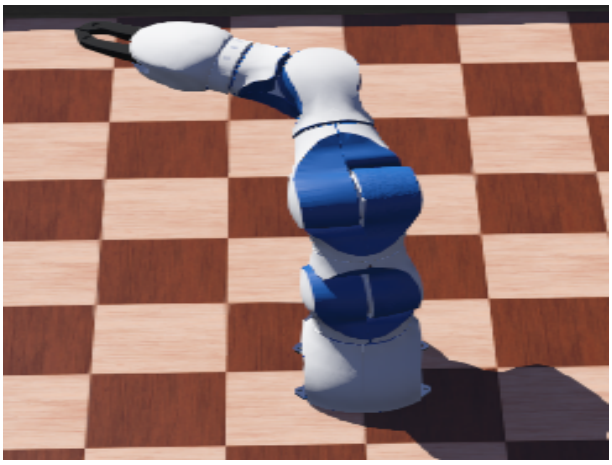

Figure 22: Fine azione motore6 del P-Rob3

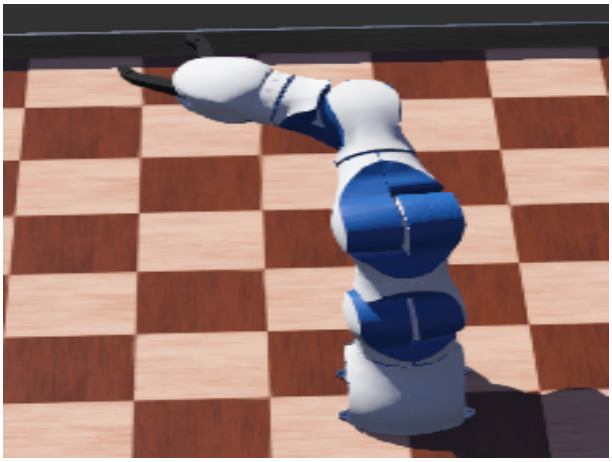

Figure 23: Fine azione motorepdx del P-Rob3

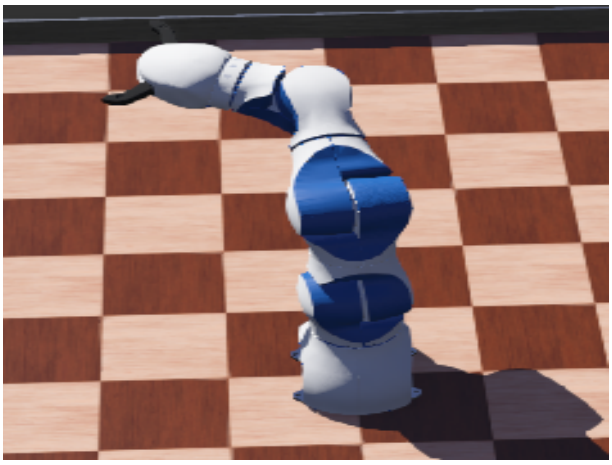

Figure 24: Fine azione motorepsx del P-Rob3