

# **Industrial interface for YUMI robot - Industrial software (ABB Robotic Studio)**

---

Leonardo Testolin, VR436823

October 6, 2020

Università degli Studi di Verona

# **Panoramica Robot YUMI bi-manuale**

---

## Robot YUMI bi-maneuale

Il robot IRB 14000 è un robot industriale a due bracci con controller integrato (IRC5). Ciascun braccio ha sette assi che consentono un grado di libertà superiore a quello dei tradizionali robot a 6 assi



**Figure 1:** Robot Yumi

## **Controller integrato**

---

## Controller Integrato

- L'IRC5 è il punto di riferimento dell'industria robotica nella tecnologia dei controller di robot. Oltre all'esclusivo controllo di movimento di ABB, offre flessibilità, sicurezza, modularità, interfacce applicative, controllo multi-robot e supporto degli strumenti per PC

# Controller Integrato

- L'IRC5 è il punto di riferimento dell'industria robotica nella tecnologia dei controller di robot. Oltre all'esclusivo controllo di movimento di ABB, offre flessibilità, sicurezza, modularità, interfacce applicative, controllo multi-robot e supporto degli strumenti per PC
- Il Controller permette che il movimento di un robot sia prevedibile e le sue prestazioni elevate, senza necessità di regolazione da parte del programmatore



**Figure 2:** IRC5 Single Cabinet

# IRC5 Caratteristiche Principali

Tra le caratteristiche che disponiamo definiamo essere:

- **Programmabile:** tutti i sistemi robot ABB sono programmati con RAPID, il linguaggio di programmazione flessibile e di alto livello di ABB

# IRC5 Caratteristiche Principali

Tra le caratteristiche che disponiamo definiamo essere:

- **Programmabile:** tutti i sistemi robot ABB sono programmati con RAPID, il linguaggio di programmazione flessibile e di alto livello di ABB
- **Relabile:** le funzioni di diagnostica integrate aiutano a garantire un rapido recupero e la produzione riprende quando le operazioni vengono interrotte

## Collision Avoidance

- IRB 14000 ha una funzionalità incorporata, chiamata **Sistema anticollisione**, che è attiva sia durante lo spostamento manuale sia durante l'esecuzione dei programmi

# Collision Avoidance

- IRB 14000 ha una funzionalità incorporata, chiamata **Sistema anticollisione**, che è attiva sia durante lo spostamento manuale sia durante l'esecuzione dei programmi
- Il sistema anticollisione tiene sotto controllo i modelli geometrici delle braccia e del corpo del robot e il robot si arresta se qualcuna di queste parti si avvicina troppo ad un'altra, chiaramente non si arresta in caso si avvicinasse a parti esterne al robot

## Collision Avoidance

- Collision Avoidance è un valore di azione del parametro Action di tipo System Input nell'argomento I/O System

## Collision Avoidance

- Collision Avoidance è un valore di azione del parametro Action di tipo System Input nell'argomento I/O System
- Il valore azione Collision Avoidance viene usato per attivare la funzione di prevenzione della collisione. Questa funzione è normalmente off nei robot

## Collision Avoidance

- Collision Avoidance è un valore di azione del parametro Action di tipo System Input nell'argomento I/O System
- Il valore azione Collision Avoidance viene usato per attivare la funzione di prevenzione della collisione. Questa funzione è normalmente off nei robot
- Collision Avoidance monitora un modello geometrico dettagliato del robot

## Collision Avoidance

- Collision Avoidance è un valore di azione del parametro Action di tipo System Input nell'argomento I/O System
- Il valore azione Collision Avoidance viene usato per attivare la funzione di prevenzione della collisione. Questa funzione è normalmente off nei robot
- Collision Avoidance monitora un modello geometrico dettagliato del robot
- Se due corpi del modello si avvicinano troppo, il controller avvisa della potenziale collisione e blocca il robot. Il parametro di sistema Coll-Pred Safety Distance determina a quale distanza i due oggetti devono essere considerati in rotta di collisione

# Collision Avoidance: Esempio

RobotStudio Software Interface:

- File**: Home, Modellazione, Simulazione, Controller, RAPID, Add-in.
- Risorse**: Aggiungi controller, Risolvi backup, Eventi, Tracciato del convegolatore, Visione integrata, Sincronizzazione, Modalità operativa, Operatore Windows, Configurazione movimenti, Vai Offline, Creare una relazione, Apri una relazione, Tracciamento.
- Accesso**: Richiedere l'accesso in scrittura, Rilasciare l'accesso in scrittura, Autentico.
- Controller**: Controller1, Controller2 (Stazioni).
- Configurazione**: Access Level, Cross Connection, Device Trust Level, DeviceNet Command, DeviceNet Device, DeviceNet Internal Device, EtherNet/IP Command, EtherNet/IP Device, EtherNet/IP Internal Device, Industrial Network, Route, Signal, Signal Safe Level, System Input, System Output.
- RAPID**: ICELab-OpeningDemo\_progetto\_robottica\_Task\_double\_arms\_20200827-164700\_20200827-172030\_20200827-173255\_20200828-100401\_View1.
- I/O System**: Man Machine Communication, Motion, PROC, Registi eventi, Sistema di I/O, Relazioni.
- Stazioni attuali**: Controller2.

**ICELab-OpeningDemo\_progetto\_robottica\_Task\_double\_arms\_20200827-164700\_20200827-172030\_20200827-173255\_20200828-100401\_View1 Controller2 (Stazioni) X**

Nome	Type of Signal	Assigned to Device	Signal Identification Label	Device Mapping	Category	Access Level	Default Value	Filter Time Pass
AS1	Digital Input	PANEL	Automatic Stop chain(X5 11 to X5:5) and (X5 9 to X5:1)	13	safety	ReadOnly	0	
AS2	Digital Input	PANEL	Automatic Stop chain backup(X5 5 to X5:6) and (X5 3 to X5:1)	14	safety	ReadOnly	0	
AUTO1	Digital Input	PANEL	Automatic Mode(X5:6)	6	safety	ReadOnly	0	
AUTO2	Digital Input	PANEL	Automatic Mode backup(X9:2)	22	safety	ReadOnly	0	
CH0	Digital Input	PANEL	Run Clean 1	23	safety	ReadOnly	0	
CH1	Digital Input	PANEL	Run Clean 2	24	safety	ReadOnly	0	
Custom_D_0	Digital Output	D6S2_10		0	All	Default	0	
Custom_D_1	Digital Output	D6S2_10		1	All	Default	0	
Custom_D_2	Digital Output	D6S2_10		2	All	Default	0	
Custom_D_3	Digital Output	D6S2_10		3	All	Default	0	
Custom_D_4	Digital Input	D6S2_10		4	All	Default	0	
Custom_D_5	Digital Input	D6S2_10		5	All	Default	0	
Custom_D_6	Digital Input	D6S2_10		6	All	Default	0	
Custom_D_7	Digital Input	D6S2_10		7	All	Default	0	
Custom_D_8	Digital Input	D6S2_10		8	All	Default	0	
Custom_D_9	Digital Input	D6S2_10		9	All	Default	0	
Custom_D_10	Digital Input	D6S2_10		10	All	Default	0	
Custom_D_11	Digital Output	D6S2_10		11	safety	Default	0	
Custom_D_12	Digital Output	D6S2_10		12	safety	Default	0	
Custom_D_13	Digital Output	D6S2_10		13	safety	Default	0	
Custom_D_14	Digital Output	D6S2_10		14	safety	Default	0	
DRY1BRAKE	Digital Output	DRY_1	Brake-release coil	2	safety	ReadOnly	0	
DRY1BRAKEFB	Digital Input	DRY_1	Brake Feedback(X6) at Contactor Board	11	safety	ReadOnly	0	
DRY1FAN1	Digital Input	DRY_1	Brake Voltage	15	safety	ReadOnly	0	
DRY1CHAN1	Digital Input	DRY_1	Chain 1 Interlocking Circuit	0	safety	ReadOnly	0	
DRY1CHAN2	Digital Output	DRY_1	Chain 2 Interlocking Circuit	1	safety	ReadOnly	0	
DRY1EXTCONT	Digital Input	DRY_1	External customer contactor (Q0) at Contactor Board	4	safety	ReadOnly	0	
DRY1FAN1	Digital Input	DRY_1	Drive Unit FAN(X10 3 to X10:4) at Contactor Board	9	safety	ReadOnly	0	
DRY1FAN2	Digital Input	DRY_1	Drive Unit FAN(X11 3 to X11:4) at Contactor Board	10	safety	ReadOnly	0	

**Usato | Stato controller | Risultati della ricerca**

Muovi i messaggi da	Tutti i messaggi	Ora	Categoria
(I) Controller2 (Stazione) 10150 - Programma avviato [2]	09/10/2020 09:41:31	Registrazione	
(I) Controller2 (Stazione) 10002 - Puntatore del programma resettato [2]	09/10/2020 09:41:31	Registrazione	
(I) Controller2 (Stazione) 10129 - Programma annullato [2]	09/10/2020 09:41:31	Registrazione	
(I) Controller2 (Stazione) 10011 - Motori attivati	09/10/2020 09:41:31	Registrazione	

**Controller2 (Stazione) 10011 - Motori attivati**

Scrivici qui per eseguire la ricerca

09:41 09/10/2020

Figure 3: Collision avoidance active

## Tipo di movimento

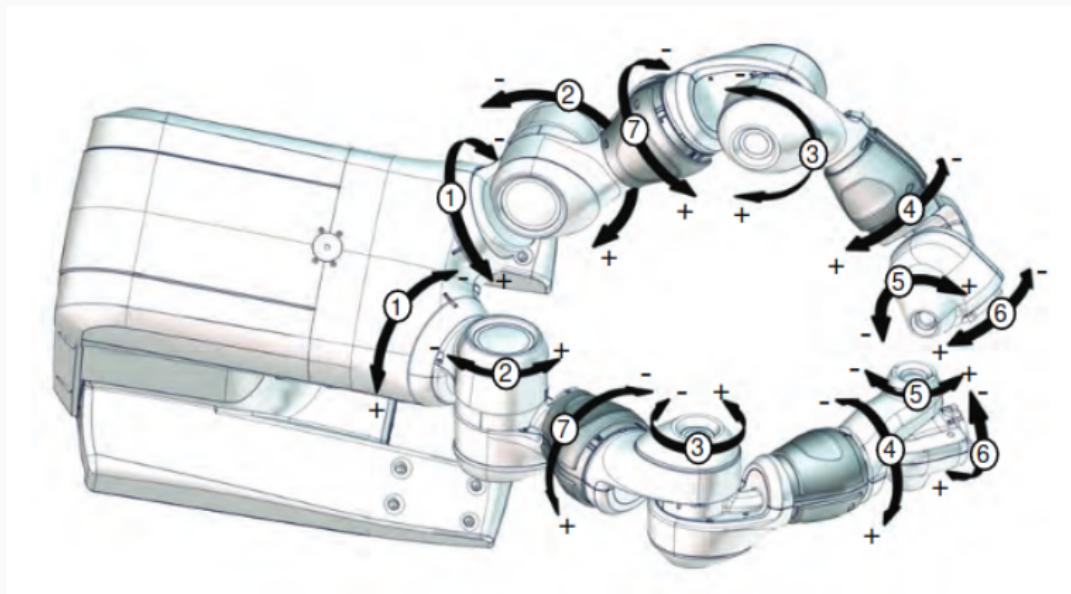
---

# Movimento del Robot

Asse	Tipo di movimento	Grado di movimento
Asse 1	Bracci - Movimento di rotazione	-168.5° to +168.5°
Asse 2	Braccio - Movimento di piegatura	-143.5° to +43.5°
Asse 7	Bracci - Movimento di rotazione	-168.5° to +168.5°
Asse 3	Braccio - Movimento di piegatura	-123.5° to +80°
Asse 4	Polso - Movimento di rotazione	-290° to +290°
Asse 5	Polso - Movimento di piegatura	-88° to +138°
Asse 6	Flangia - Movimento di rotazione	-229° to +229°

Figure 4: Movimento del Robot

# Movimento del Robot



**Figure 5:** Movimento del Robot

## **Tipologia di Gripper**

---

## Informazioni generali

- La pinza IRB 14000 è un utensile intelligente e multifunzionale per la gestione e l'assemblaggio di parti

## Informazioni generali

- La pinza IRB 14000 è un utensile intelligente e multifunzionale per la gestione e l'assemblaggio di parti
- La pinza ha un modulo servoassistito di base e due moduli funzionali, aspirazione e visione

## Informazioni generali

- La pinza IRB 14000 è un utensile intelligente e multifunzionale per la gestione e l'assemblaggio di parti
- La pinza ha un modulo servoassistito di base e due moduli funzionali, aspirazione e visione
- I tre moduli possono essere combinati insieme per offrire cinque diverse combinazioni per le diverse esigenze degli utenti

# Moduli funzionali

	Modulo funzionale	Descrizione
1	Servoassistito	Il modulo servoassistito costituisce il componente base della pinza. È questo modulo a consentire la presa sugli oggetti. Le dita sono installate alla base del modulo servoassistito e il loro movimento e la loro forza possono essere controllati e monitorati.
2	Aspirazione	Il modulo di aspirazione contiene il generatore di vuoto, il sensore della pressione e l'attuatore pneumatico. Quando vengono montati gli utensili di aspirazione, la pinza è in grado prendere gli oggetti per aspirazione e posizionarli per soffiaggio.
3	Visione	Il modulo di visione contiene una videocamera Cognex AE3 In-Sight che supporta tutte le funzioni di ABB Integrated Vision.

**Figure 6:** Moduli funzionali

## Moduli funzionali: combinazione

	Combinazione	Include...
1	Servoassistito	Un modulo servoassistito
2	Servoassistito + Aspirazione	Un modulo servoassistito e un modulo di aspirazione
3	Servoassistito + Aspirazione 1 + Aspirazione 2	Un modulo servoassistito e due moduli di aspirazione
4	Servoassistito + Visione	Un modulo servoassistito e un modulo di visione
5	Servoassistito + Visione + Aspirazione	Un modulo servoassistito, un modulo di visione e un modulo di aspirazione

Figure 7: Moduli funzionali combinazione

## Servoassistito



**Figure 8:** Gripper Servoassistito

## Servoassistito + Aspirazione



**Figure 9:** Servoassistito + Aspirazione

## Servoassistito + Aspirazione1 + Aspirazione2



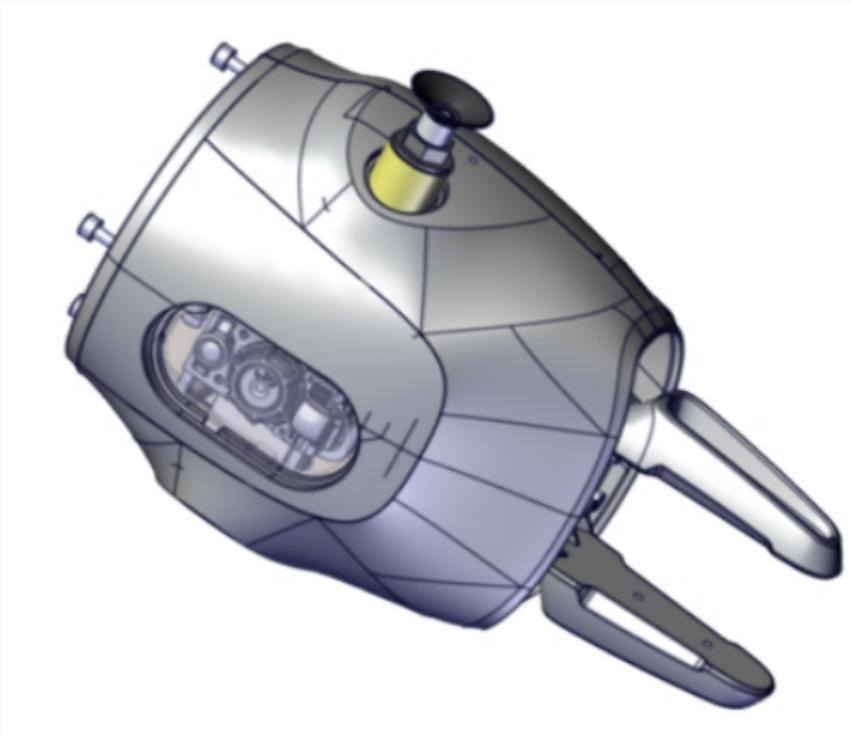
**Figure 10:** Servoassistito + Aspirazione1 + Aspirazione2

## Servoassistito + Visione



**Figure 11:** Servoassistito + Visione

## Servoassistito + Visione + Aspirazione



**Figure 12:** Servoassistito + Visione + Aspirazione

# Panoramica RobotStudio ABB

---

## RobotStudio ABB

- RobotStudio è un utensile tecnico per la configurazione e la programmazione di Robot ABB, sia di tipo reale per applicazioni sul campo che di tipo virtuale per PC. Per raggiungere la vera programmazione fuori linea, RobotStudio utilizza ABB VirtualRobot™ Technology

- RobotStudio è un utensile tecnico per la configurazione e la programmazione di Robot ABB, sia di tipo reale per applicazioni sul campo che di tipo virtuale per PC. Per raggiungere la vera programmazione fuori linea, RobotStudio utilizza ABB VirtualRobot™ Technology
- RobotStudio consente di lavorare con un controller **offline**, ovvero un controller IRC5 virtuale in esecuzione locale sul PC. Tale controller offline viene anche indicato come **controller virtuale (VC)**

- RobotStudio è un utensile tecnico per la configurazione e la programmazione di Robot ABB, sia di tipo reale per applicazioni sul campo che di tipo virtuale per PC. Per raggiungere la vera programmazione fuori linea, RobotStudio utilizza ABB VirtualRobot™ Technology
- RobotStudio consente di lavorare con un controller **offline**, ovvero un controller IRC5 virtuale in esecuzione locale sul PC. Tale controller offline viene anche indicato come **controller virtuale (VC)**
- RobotStudio consente inoltre di lavorare con il controller IRC5 fisico reale, che viene semplicemente indicato come **controller reale**, indicando RobotStudio in modalità **online**

# RobotStudio ABB: RobotWare

- La gestione semplice, flessibile e veloce è dovuta a **RobotWare** che è la famiglia di software che permette di facilitare, per esempio, la programmazione del linguaggio RAPID e di conseguenza l'azionamento del robot.  
Ma non solo, infatti con questa gamma di software si sono ottenuti ottimi risultati nell'ambito della Motion Technology



**Figure 13:** TrueMove Model

## Controller Virtuali

- **Controller virtuale:** Un software che emula un FlexController per consentire allo stesso software (il sistema RobotWare) che controlla i robot di essere eseguito su un PC. Questo consente di ottenere fuori linea lo stesso comportamento dei robot in linea

- **Controller virtuale:** Un software che emula un FlexController per consentire allo stesso software (il sistema RobotWare) che controlla i robot di essere eseguito su un PC. Questo consente di ottenere fuori linea lo stesso comportamento dei robot in linea
- **FlexController:** L'armadietto del controller per i robotIRC5. È composto da un modulo di comando e da un modulo di azionamento per ciascun robot manipolatore nel sistema

# Target

I target (posizioni) e i percorsi (sequenze di istruzioni di movimento verso i target) vengono utilizzati nella programmazione dei movimenti del robot in RobotStudio. Un target è una coordinata che il robot dovrà raggiungere.

Contiene le seguenti informazioni:

- Posizione definita in un sistema di coordinate dell'oggetto di lavoro

# Target

I target (posizioni) e i percorsi (sequenze di istruzioni di movimento verso i target) vengono utilizzati nella programmazione dei movimenti del robot in RobotStudio. Un target è una coordinata che il robot dovrà raggiungere.

Contiene le seguenti informazioni:

- Posizione definita in un sistema di coordinate dell'oggetto di lavoro
- Orientamento relativo all'orientamento dell'oggetto di lavoro.  
Quando il robot raggiunge il target allinea l'orientamento del TCP con quello del target

# Target

I target (posizioni) e i percorsi (sequenze di istruzioni di movimento verso i target) vengono utilizzati nella programmazione dei movimenti del robot in RobotStudio. Un target è una coordinata che il robot dovrà raggiungere.

Contiene le seguenti informazioni:

- Posizione definita in un sistema di coordinate dell'oggetto di lavoro
- Orientamento relativo all'orientamento dell'oggetto di lavoro.  
Quando il robot raggiunge il target allinea l'orientamento del TCP con quello del target
- Configurazione che specifica come il robot deve raggiungere il target

## Target e Percorsi

Un percorso consente al robot di spostarsi su una sequenza di target (mediante istruzioni di movimento).

Un'istruzione di movimento è composta da:

- Un riferimento a un target
- Dati di movimento (velocità e zona (punti di via))
- Un riferimento a dati utensile
- Un riferimento all'oggetto di lavoro
- Un'istruzione di azione per impostare e modificare i parametri

## RobotStudio ABB: RAPID

- Nell'editor presente su RobotStudio è possibile creare del **codice RAPID**

## RobotStudio ABB: RAPID

- Nell'editor presente su RobotStudio è possibile creare del **codice RAPID**
- In generale, quando si crea un task si associa un programma RAPID nel quale verranno utilizzati dei **moduli** per svolgere una determinata funzione, per esempio di saldatura o di un movimento del manipolatore

## RobotStudio ABB: RAPID

- Nell'editor presente su RobotStudio è possibile creare del **codice RAPID**
- In generale, quando si crea un task si associa un programma RAPID nel quale verranno utilizzati dei **moduli** per svolgere una determinata funzione, per esempio di saldatura o di un movimento del manipolatore
- Ad ogni programma viene associato un modulo all'interno del quale si svolgerà una **routine** che deve essere eseguibile

- Nell'editor presente su RobotStudio è possibile creare del **codice RAPID**
- In generale, quando si crea un task si associa un programma RAPID nel quale verranno utilizzati dei **moduli** per svolgere una determinata funzione, per esempio di saldatura o di un movimento del manipolatore
- Ad ogni programma viene associato un modulo all'interno del quale si svolgerà una **routine** che deve essere eseguibile
- Ogni modulo può avere i dati necessari al funzionamento del programma nel modulo stesso o in altri moduli linkati

## RobotStudio ABB: Rapid

- Per routine si intende la definizione di un "**main**" che sarà il punto di partenza del task

- Per routine si intende la definizione di un "**main**" che sarà il punto di partenza del task
- Ciascuna istruzione è una richiesta di esecuzione di un determinato evento, ad esempio "porta il TCP del manipolatore in una certa posizione" o "imposta un output digitale specifico"

# Esempio Codice Rapid

Un target corrisponde a un **RobTarget** in un programma RAPID.

```
CONST robtarget AbovePickPosition:=  
[[330.40595962, -4.47168768, 138.85283443],  
[0,0,1,0],  
[-1,2,-1,4],  
[126.569998941, 9E+09, 9E+09, 9E+09, 9E+09, 9E+09]];
```

Tipo di dati	Nome	Descrizione
pos	trans	coordinate x, y e z
orient	rot	Orientamento
confdata	robconf	Specifica gli angoli degli assi del robot
extjoint	extax	Specifica le posizioni per 6 assi supplementari. Il valore viene impostato a 9E9 se non viene utilizzato alcun asse supplementare.

# Esempio Codice Rapid

## MoveL

Una semplice istruzione di spostamento può apparire così:

- MoveL AbovePickposition,v1000,fine,Servo:=wobj0;

MoveL è un'istruzione che sposta il robot linearmente (in linea retta) dalla sua posizione corrente a quella specificata.

Un'istruzione di movimento è composta da:

# Esempio Codice Rapido

## MoveL

Una semplice istruzione di spostamento può apparire così:

- MoveL AbovePickposition,v1000,fine,Servo:=wobj0;

MoveL è un'istruzione che sposta il robot linearmente (in linea retta) dalla sua posizione corrente a quella specificata.

Un'istruzione di movimento è composta da:

- *Un riferimento a un target, dati di movimento, ad esempio, tipo di movimento, velocità e zona, un riferimento a dati utensile, un riferimento all'oggetto di lavoro*

# Esempio Codice Rapido

## MoveL

Una semplice istruzione di spostamento può apparire così:

- MoveL AbovePickposition,v1000,fine,Servo:=wobj0;

MoveL è un'istruzione che sposta il robot linearmente (in linea retta) dalla sua posizione corrente a quella specificata.

Un'istruzione di movimento è composta da:

- *Un riferimento a un target, dati di movimento, ad esempio, tipo di movimento, velocità e zona, un riferimento a dati utensile, un riferimento all'oggetto di lavoro*
- **AbovePickposition** specifica la posizione verso la quale deve spostarsi il robot

# Esempio Codice Rapid

## MoveL

Una semplice istruzione di spostamento può apparire così:

- MoveL AbovePickposition,v1000,fine,Servo:=wobj0;

MoveL è un'istruzione che sposta il robot linearmente (in linea retta) dalla sua posizione corrente a quella specificata.

Un'istruzione di movimento è composta da:

- *Un riferimento a un target, dati di movimento, ad esempio, tipo di movimento, velocità e zona, un riferimento a dati utensile, un riferimento all'oggetto di lavoro*
- **AbovePickposition** specifica la posizione verso la quale deve spostarsi il robot
- **v1000** specifica che la velocità del robot dovrà essere di 1000 mm/s

## Esempio Codice Rapid

```
MoveL AbovePickposition_right,v1000,fine,Servo:=wobj0;
```

- **fine** specifica che il robot dovrà spostarsi esattamente alla posizione specificata, e non tagliare alcuna curva nel suo percorso verso la posizione successiva

## Esempio Codice Rapid

```
MoveL AbovePickposition_right,v1000,fine,Servo:=wobj0;
```

- **fine** specifica che il robot dovrà spostarsi esattamente alla posizione specificata, e non tagliare alcuna curva nel suo percorso verso la posizione successiva
- **Servo** specifica che la flangia di montaggio sulla punta del robot deve muoversi verso la posizione specificata

# Coordinate

Un **sistema di coordinate** definisce il piano o spazio tramite gli assi da un punto fisso chiamato **origine**. Le destinazioni e le posizioni del robot vengono individuate tramite misurazioni lungo gli assi dei sistemi di coordinate.

## Sistema di coordinate dell'oggetto di lavoro

Normalmente, **l'oggetto di lavoro** rappresenta il pezzo di **lavoro fisico**. È composto da due sistemi di coordinate: il **sistema di riferimento utente** e il **sistema di riferimento oggetto**, dove il secondo è figlio del primo. Quando si programma un robot, tutti i target (posizioni) sono correlati al sistema di riferimento oggetto di un oggetto di lavoro. Se non è stato specificato un altro oggetto di lavoro, i target saranno correlati al **Wobj0** predefinito, che coincide sempre con il sistema di riferimento di base del robot.

## Sistema coordinate Utente

Gli **UCS** (sistemi di coordinate utente) vengono utilizzati per la creazione di punti di riferimento specifici. Ad esempio, è possibile creare sistemi di riferimento utente in punti strategici dell'oggetto di lavoro, per facilitare la programmazione.

## Altri tipi di coordinate

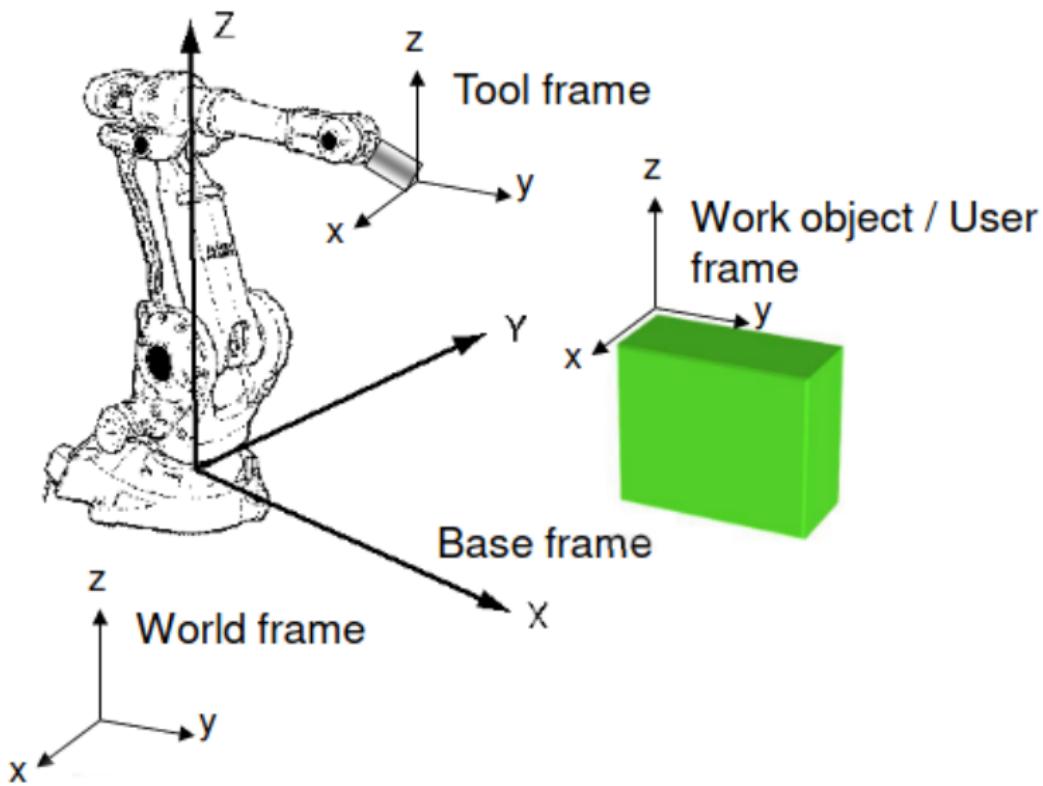
Un robot utilizza vari sistemi di coordinate, oltre a quelli appena descritti anche:

- **Il sistema di coordinate di base** è situato alla base del robot. È il sistema più semplice per spostare il robot da una posizione a un'altra
- **Il sistema di coordinate utensile** definisce la posizione dell'utensile utilizzato dal robot quando raggiunge le destinazioni programmate
- **Sistema di coordinate del TCP** (Tool Center Point) è il punto centrale dell'utensile. Per un robot è possibile che siano definiti diversi TCP. Tutti i robot hanno un TCP predefinito nel punto di montaggio dell'utensile, chiamato **tool0**. Quando si esegue un programma, il TCP viene spostato nella posizione programmata

## Altri tipi di coordinate

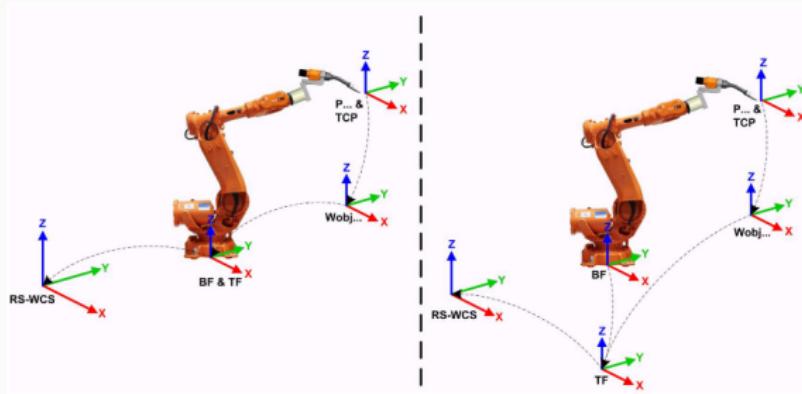
- **Il sistema di coordinate del Task Frame (TF)** rappresenta l'origine del sistema di coordinate universali del controller del robot in RobotStudio
- **Il sistema di coordinate universale** che definisce l'intera stazione o la cella del robot. Tutti gli altri sistemi di coordinate sono direttamente o indirettamente legati al sistema di coordinate universale

## Coordinate



# Esempio Sistema di coordinate

RS-WCS	Sistema di coordinate universali in RobotStudio.
RC-WCS	Sistema di coordinate universali quale definito nel controller del robot. Esso corrisponde al Task Frame di RobotStudio.
BF	Base Frame del robot
TCP	Tool Center Point
P	Robottarget
TF	Task Frame
Wobj	Work object

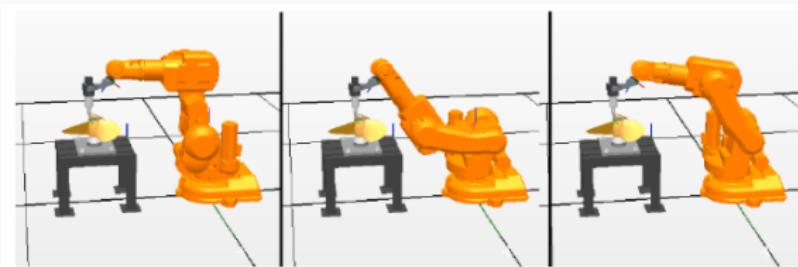


# Configurazione degli assi del Robot

I **target** vengono definiti e memorizzati come **coordinate in un sistema di coordinate dell'oggetto di lavoro**. Quando il controller calcola la posizione degli assi del robot per raggiungere il target, spesso vi sarà più di una soluzione possibile per la configurazione degli assi del robot. Per distinguere tra le varie configurazioni, tutti i target hanno un valore di configurazione che specifica in quale quadrante potrà essere situato ciascun asse.

# Configurazione degli assi del Robot

**Ad ogni target** deve essere assegnata una **configurazione valida** e verificato che il robot possa muoversi lungo ciascun percorso. È anche possibile disattivare il monitoraggio della configurazione in modo da ignorare le configurazioni memorizzate e consentire che il robot trovi quelle funzionanti in fase di runtime. Se questo non viene svolto nel modo corretto, si potrebbero ottenere risultati inaspettati.



**Figure 15:** Differenti Configurazioni

## Configurazione degli assi del Robot

In alcuni casi, potrebbe non essere disponibile alcuna configurazione funzionante. Le soluzioni possibili possono essere quindi il riposizionamento del pezzo di lavoro, il riorientamento dei target, se accettabile nel processo, o l'aggiunta di un asse esterno che sposti il pezzo di lavoro o aumenti la raggiungibilità del robot.

## Segnali d'I/O

I segnali vengono utilizzati per la comunicazione con dispositivi esterni con cui il robot coopera. I segnali d'ingresso vengono impostati dai dispositivi esterni, e si possono utilizzare nel programma RAPID, per iniziare una certa esecuzione da parte del robot. I segnali d'uscita vengono impostati dal programma RAPID, che segnala ai dispositivi esterni che devono eseguire qualche azione.

# Segnali d'I/O

## Ingresso digitale

Un segnale d'ingresso digitale può avere i valori 0 o 1. Il programma RAPID è in grado di leggere il suo valore, ma non può impostarlo.

## Uscita digitale

Un segnale d'uscita digitale può comportare i valori 0 o 1. Il programma RAPID può impostare il valore per un segnale d'uscita digitale, e così influire sui dispositivi esterni. Il valore di un segnale digitale d'uscita viene impostato mediante l'istruzione SetDO.

# Progetto

---

## Primo Task: Pick&Place

- La prima parte del progetto consisteva nella creazione di un task Pick&Place relativo a delle mansioni possibilmente industriali

## Primo Task: Pick&Place

- La prima parte del progetto consisteva nella creazione di un task Pick&Place relativo a delle mansioni possibilmente industriali
- Quindi ho riprodotto lo spostamento di un oggetto da un punto all'altro e il suo ritorno. Come modello ho creato un cubo 20x20x20

## Primo Task: Pick&Place

- La prima parte del progetto consisteva nella creazione di un task Pick&Place relativo a delle mansioni possibilmente industriali
- Quindi ho riprodotto lo spostamento di un oggetto da un punto all'altro e il suo ritorno. Come modello ho creato un cubo 20x20x20
- Il robot parte da una posa iniziale denominata "Home" si posiziona sopra l'oggetto lo prende lo sposta, lo rilascia lo riprende e lo rimette nella posizione iniziale

# Stazione

In RobotStudio si definisce **stazione** l'ambiente in cui si definiscono e si simulano i task. In questo caso, ho lavorato direttamente sulla stazione creata per il laboratorio ICE. Lo Yumi e l'ambiente che lo circonda erano già impostati.

## Controller Virtuali

Una volta caricato l'ambiente e definito la stazione di lavoro, RobotStudio prevede l'inserimento dei tool, *smartGripper*, ovvero, i Gripper che sono stati descritti precedentemente. In particolare, grazie al controller IRC5 RobotStudio carica e avvia i motori, permettendo così di muovere a livello simulativo le braccia del robot.

Una volta collegati i Gripper alle braccia e il controller avviato è stato possibile passare allo step successivo.

## Modifica Meccanismo Gripper

L'immagine successiva mostra la modifica del meccanismo di apertura e chiusura del Gripper.

In questo modo, in base all'oggetto scelto sono andato a definire la posizione del Gripper "in riposo" (HomePose) e la posizione del Gripper quando prende un oggetto (SynchPose).

Ho definito quindi per la HomePose la massima posizione del Gripper, mentre, per la presa dell'oggetto ho impostato il valore di posizione che mi permetesse di agganciare l'oggetto definito come modello.

# Modifica Meccanismo Gripper

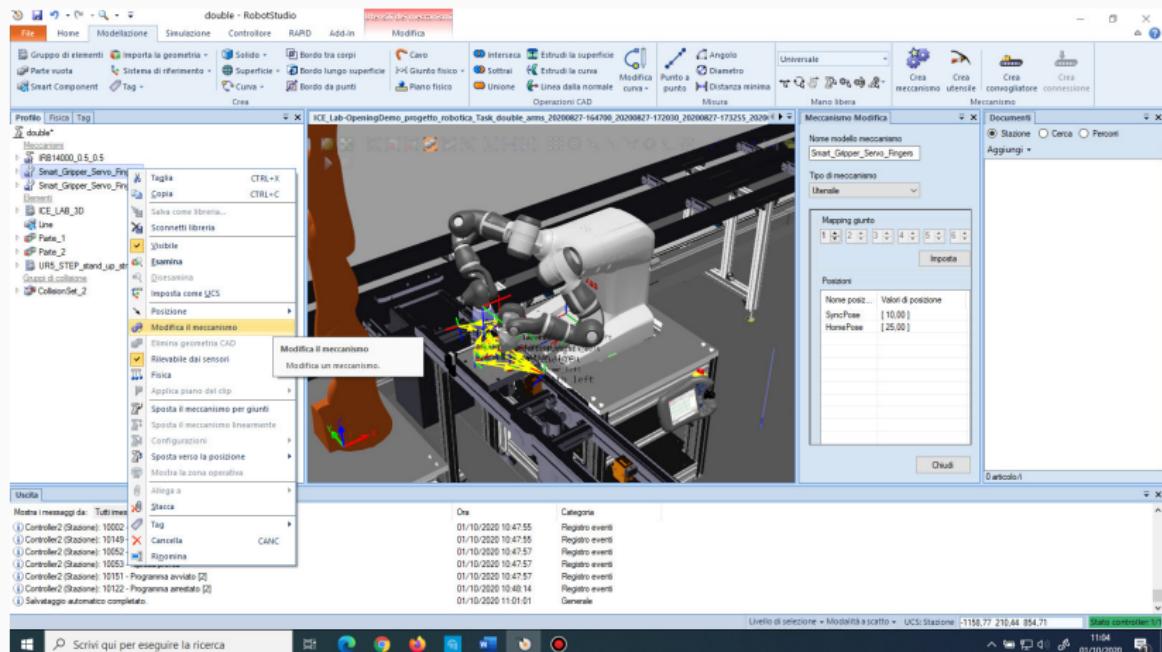


Figure 16: Modifica Meccanismo Gripper

## Creazione del modello (Cubo 20x20x20)

Successivamente sono andato a definire e posizionare l'oggetto all'interno dell'ambiente, spostandolo linearmente lungo la stazione.

Come si vedrà nell'immagine successiva, andando sulla scheda: Modellazione → Solido → Scatola (Box) è possibile definirne le dimensioni.

# Creazione del modello (Cubo 20x20x20)

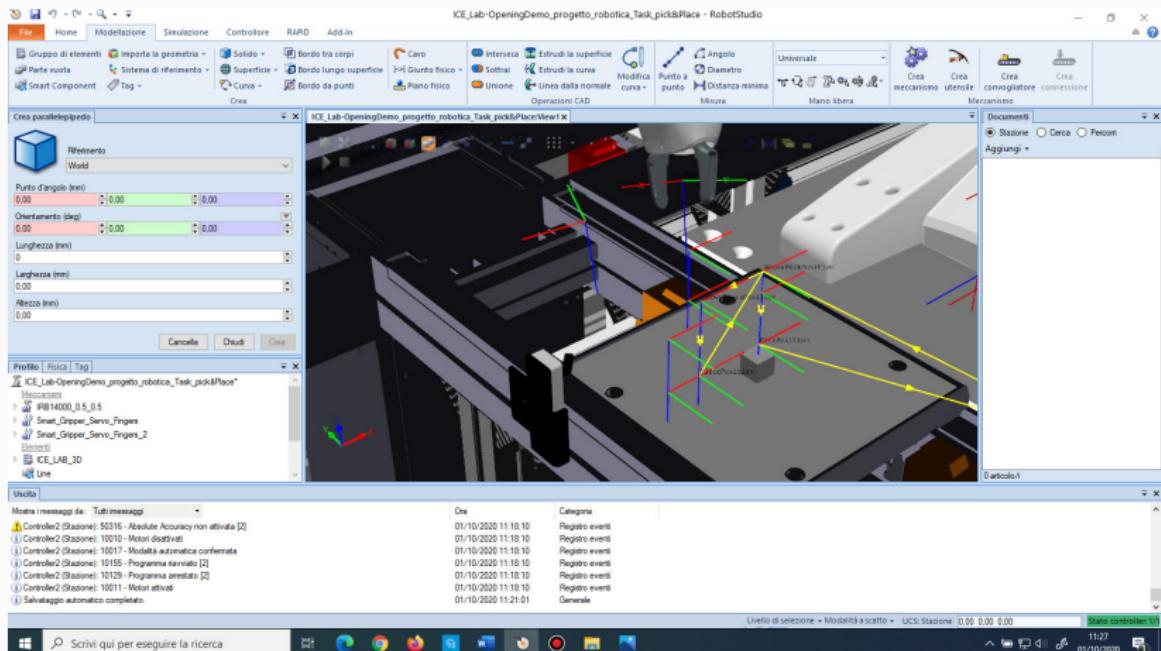


Figure 17: Creazione Modello

## Definizione dei target

Dopo aver definito la posizione del modello è possibile definirne i target, come spiegato precedentemente, sono la definizione dei punti nello spazio.

## Definizione dei target

Sono andato a definire alcuni target per la creazione del primo task Pick&Place.

Ho creato la posa iniziare, definita "Home" nel seguente modo:  
Posizionandomi sulla scheda Home ho selezionato il pulsante  
"Apprendi target", definendo così come si vedrà nell'immagine  
successiva, la posa iniziale del robot.

## Definizione dei target

Sono andato a definire alcuni target per la creazione del primo task Pick&Place.

Ho creato la posa iniziare, definita "Home" nel seguente modo:  
Posizionandomi sulla scheda Home ho selezionato il pulsante  
"Apprendi target", definendo così come si vedrà nell'immagine  
successiva, la posa iniziale del robot.

- Nota: La definizione della posa iniziale denominata **Home** è stata possibile grazie allo spostamento del giunto (spalla), lineare e di orientamento per il braccio e per il Gripper

# Definizione del target Home

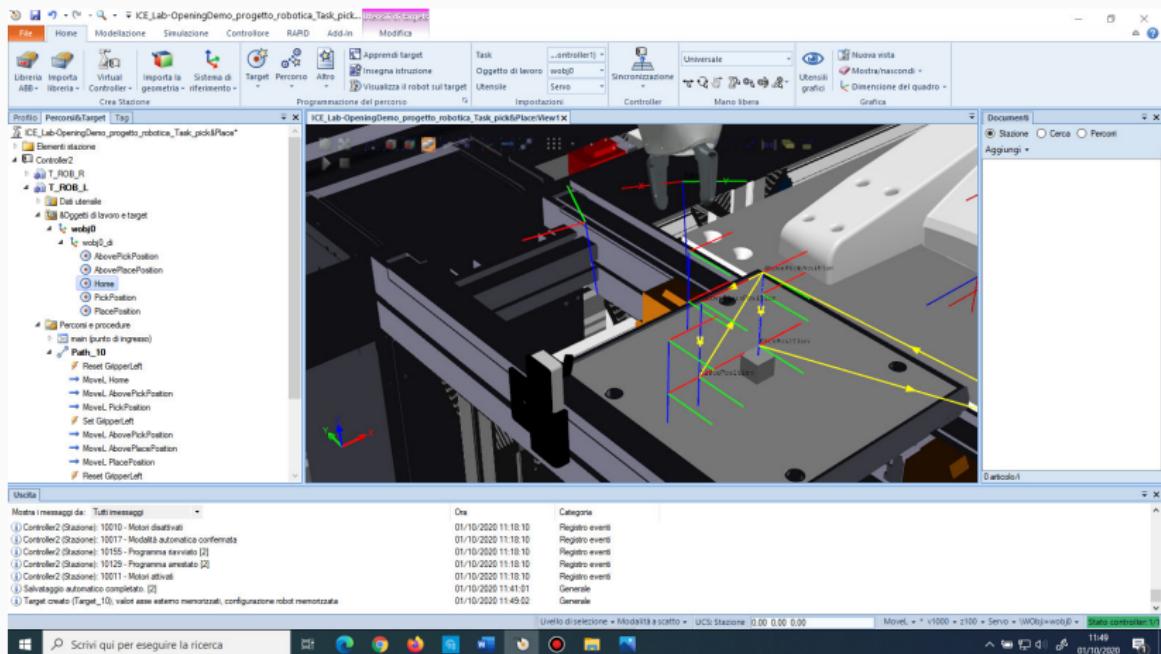


Figure 18: Apprendi target

## Definizione dei target

Successivamente ho definito gli altri target per la definizione del task nel seguente modo:

Ho selezionato il punto di mio interesse, ho definito il Pick e ho creato il target corrispondente al punto selezionato.

Andando sulla scheda Home e selezionando "Crea target" ho creato e definito il punto selezionato.

Come si vede nella figura successiva

- Nota: **Ogni target è stato configurato in coordinate cartesiane**

# Definizione dei target

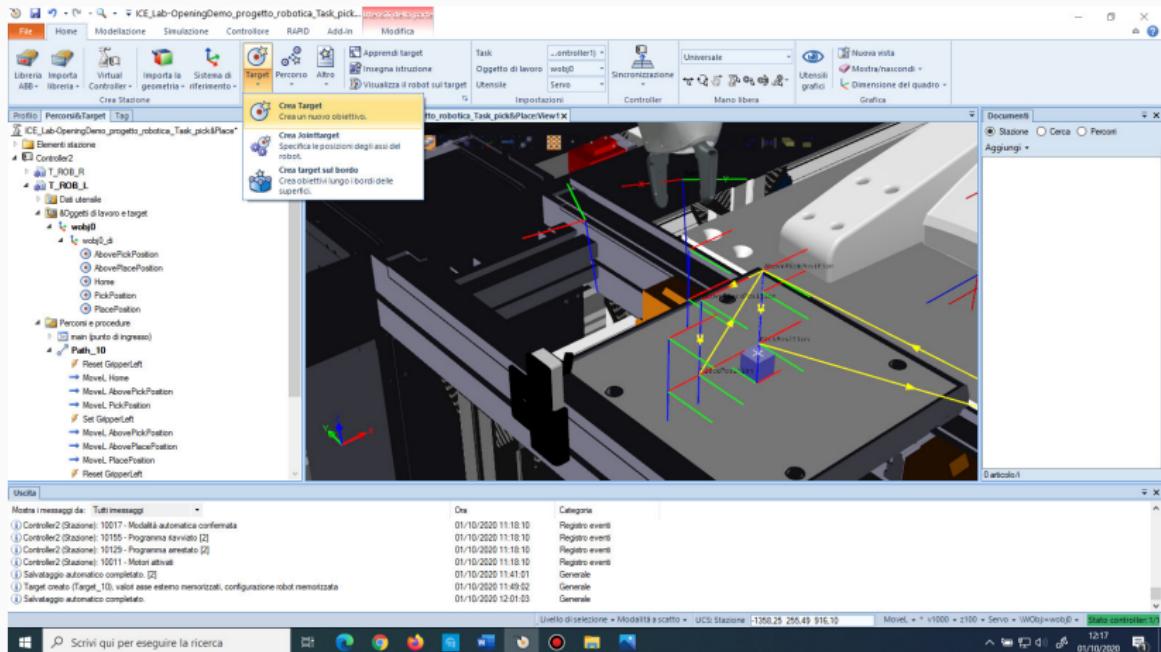


Figure 19: Creazione Target

# Definizione dei target

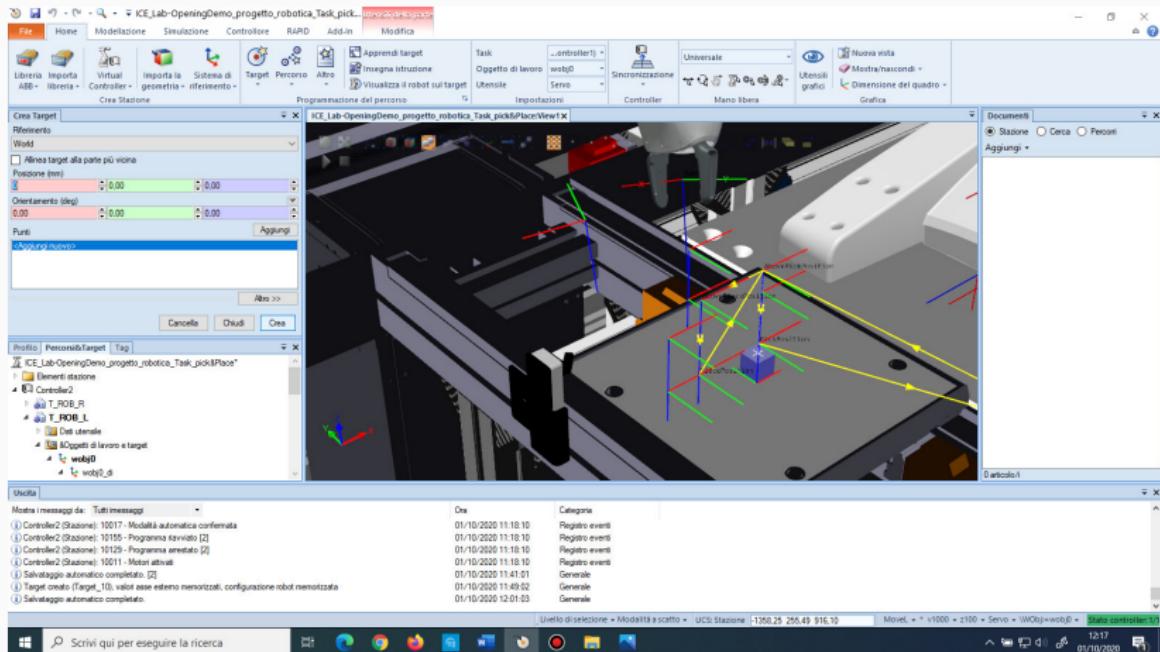


Figure 20: Creazione target

## Definizione dei target

Nel creare tutti i successivi target, ho dovuto cambiare l'orientamento del gripper. Nel momento in cui andiamo a definire i target, c'è la possibilità di impostare la visualizzazione del robot sul target oppure del solo Gripper.

Questa funzione ti permette di capire quale sia **l'orientamento** del robot rispetto al quel target. Così per tutti i target, escluso la Home, sono andato a visualizzare il gripper sul target e cambiare l'orientamento, come mostrato nella figura seguente.

# Orientamento Gripper

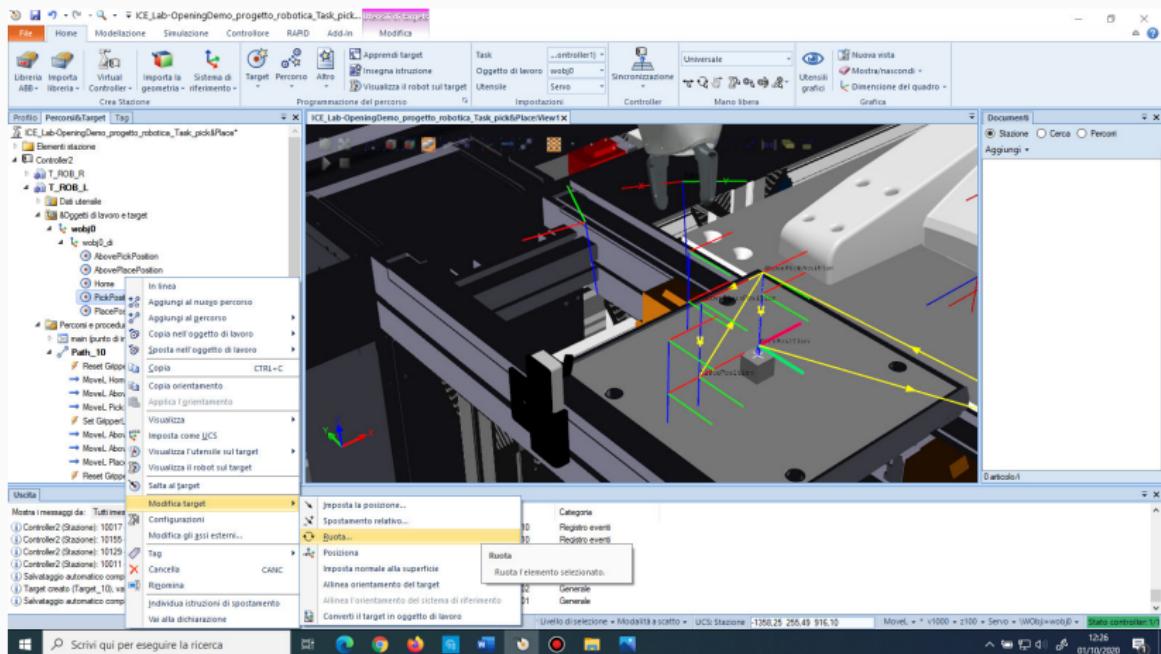


Figure 21: Orientamento Gripper

## Configurazione Cinematica Inversa

Successivamente è stato necessario impostare la cinematica inversa. In questo caso, una volta definito il target e l'orientamento degli assi è stato necessario definirne la cinematica. Impostandola come si può vedere dall'immagine seguente

# Configurazione Cinematica Inversa

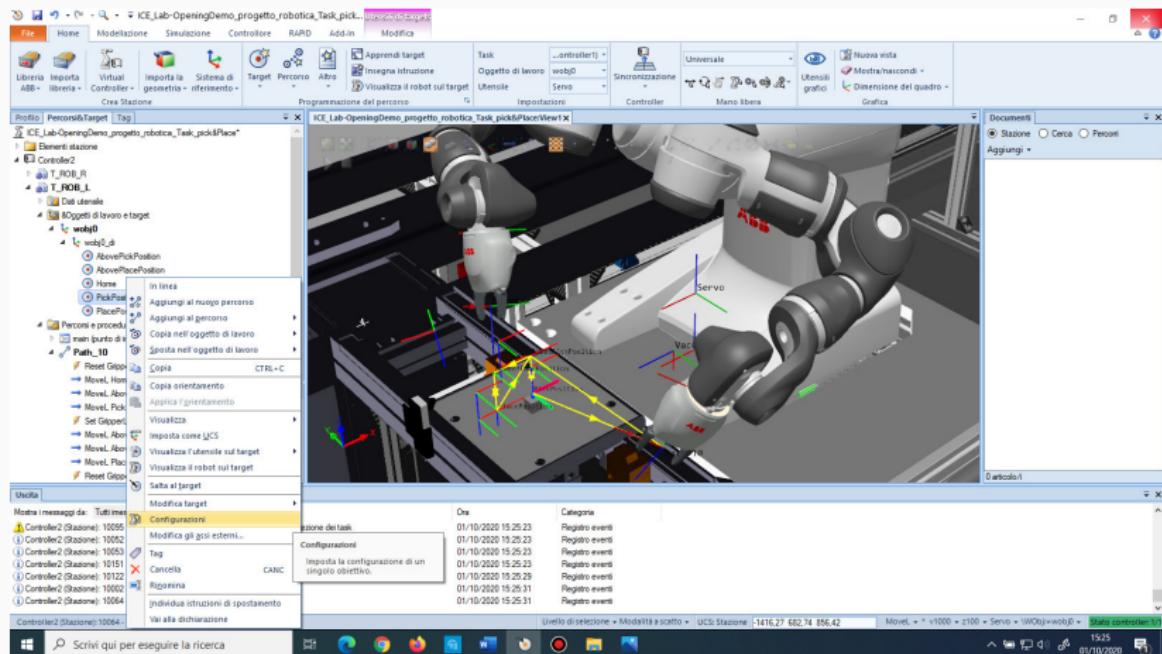


Figure 22: Cinematica inversa

# Configurazione Cinematica Inversa

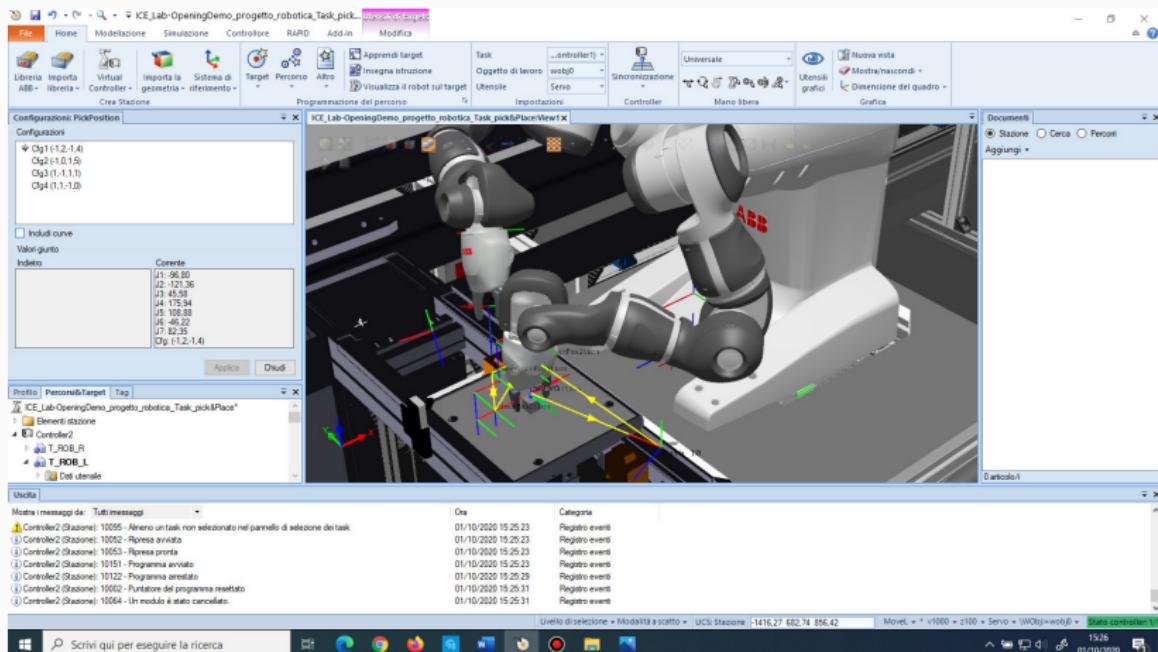


Figure 23: Configurazione cinematica inversa

# Configurazione Cinematica Inversa Errata

## Percorso

A questo punto è possibile delineare il percorso che il robot deve fare per la creazione del task.

Quindi per prima cosa sono andato nella scheda Home e ho creato un nuovo percoso, il quale, conterrà tutti i movimenti affinchè il robot possa realizzarli. *Il percorso rappresenterà il mio task che in questo caso contiene solamente 5 target; **Home**, **AbovePickPosition**, **PickPosition**, **AbovePlacePosition**, **PlacePosition**.*

## Percorso

A questo punto è possibile delineare il percorso che il robot deve fare per la creazione del task.

Quindi per prima cosa sono andato nella scheda Home e ho creato un nuovo percorso, il quale, conterrà tutti i movimenti affinchè il robot possa realizzarli. *Il percorso rappresenterà il mio task che in questo caso contiene solamente 5 target; **Home**, **AbovePickPosition**, **PickPosition**, **AbovePlacePosition**, **PlacePosition**.*

- Nel creare gli altri target il procedimento è stato identico, solamente che per i target **AbovePickPosition** e **AbovePlacePosition** oltre al cambio di orientamento, ho dovuto traslare lungo l'asse z per poter posizionarmi correttamente sopra l'oggetto prima di fare il **Pick** e il **Place**

# Percorso

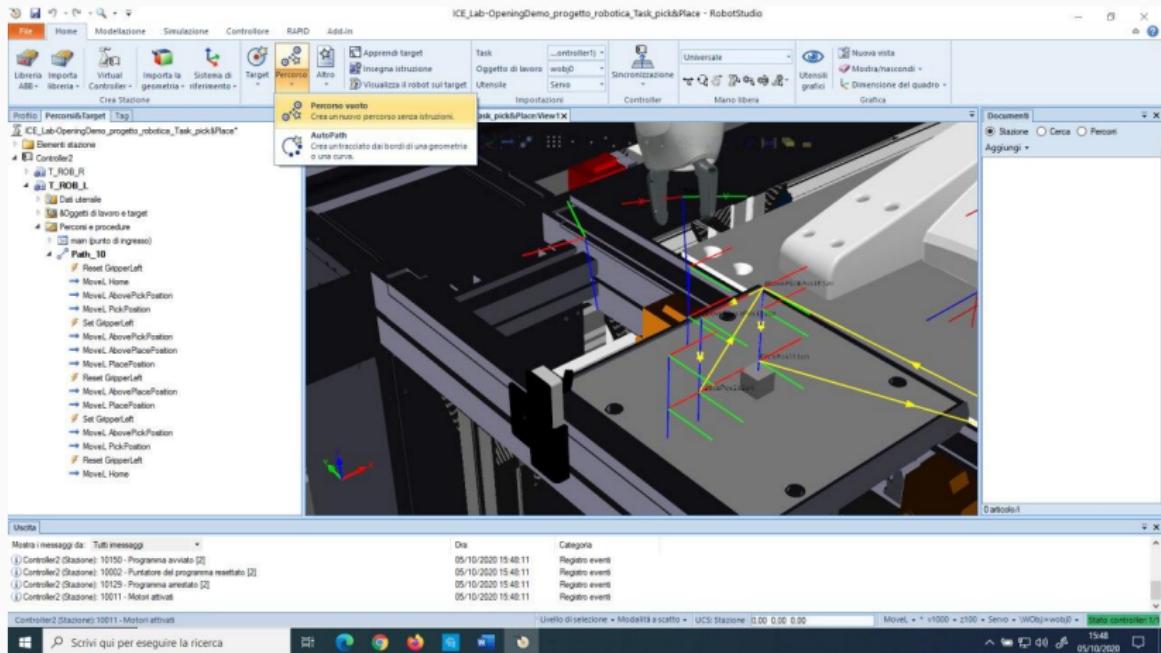


Figure 24: Crea Percorso vuoto

## Percorso e segnali DO

Dopo aver configurato e creato il percorso, sono andato a creare i segnali **Digital Output** per poter simulare l'apertura e la chiusura del Gripper. Sono andato nella scheda Controllore → Configurazione → I/O System. Creando quindi un nuovo tipo di Segnale digitale.

# Segnali DO

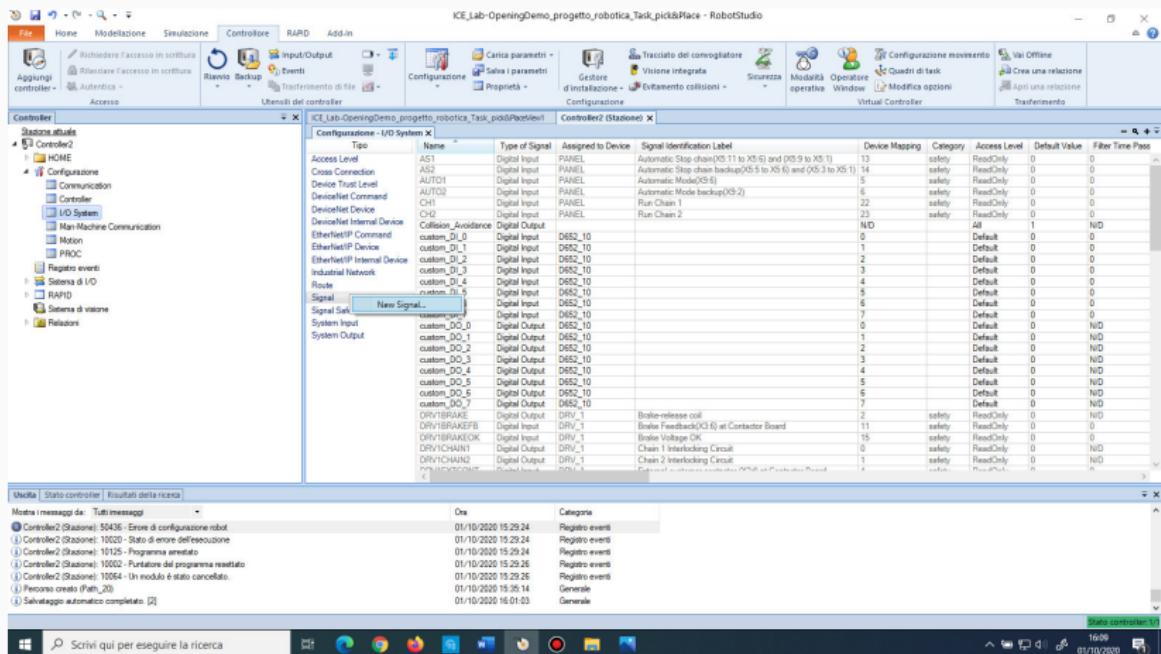


Figure 25: Definizione segnali DO

# Segnali DO

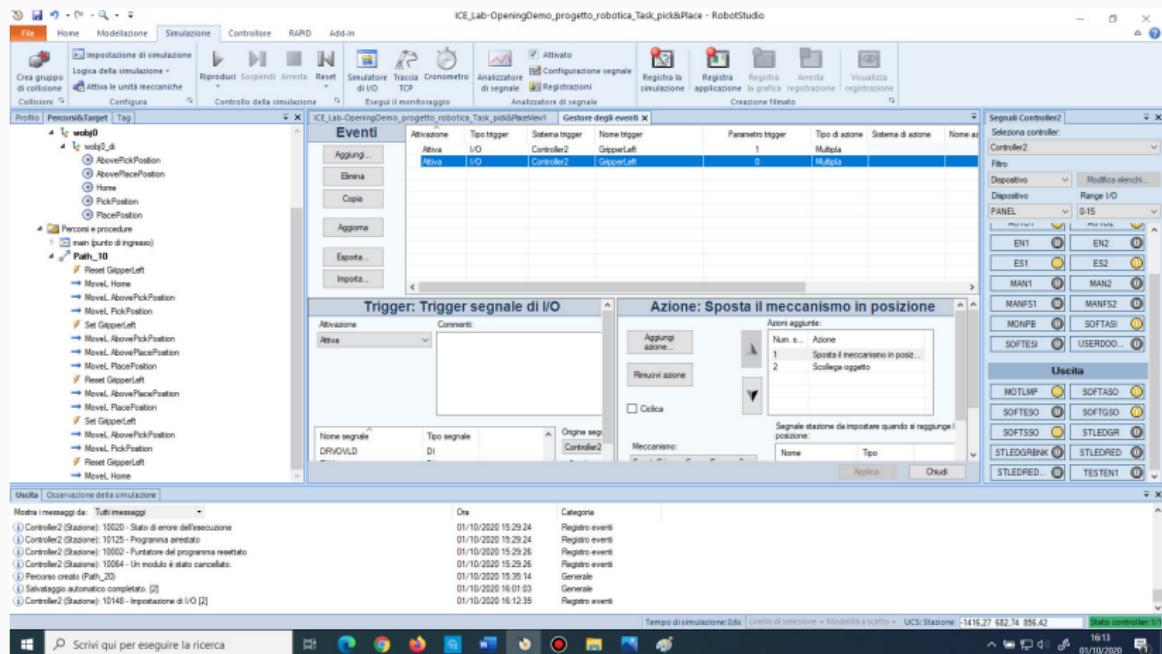


Figure 26: Segnale DO

# Segnali DO

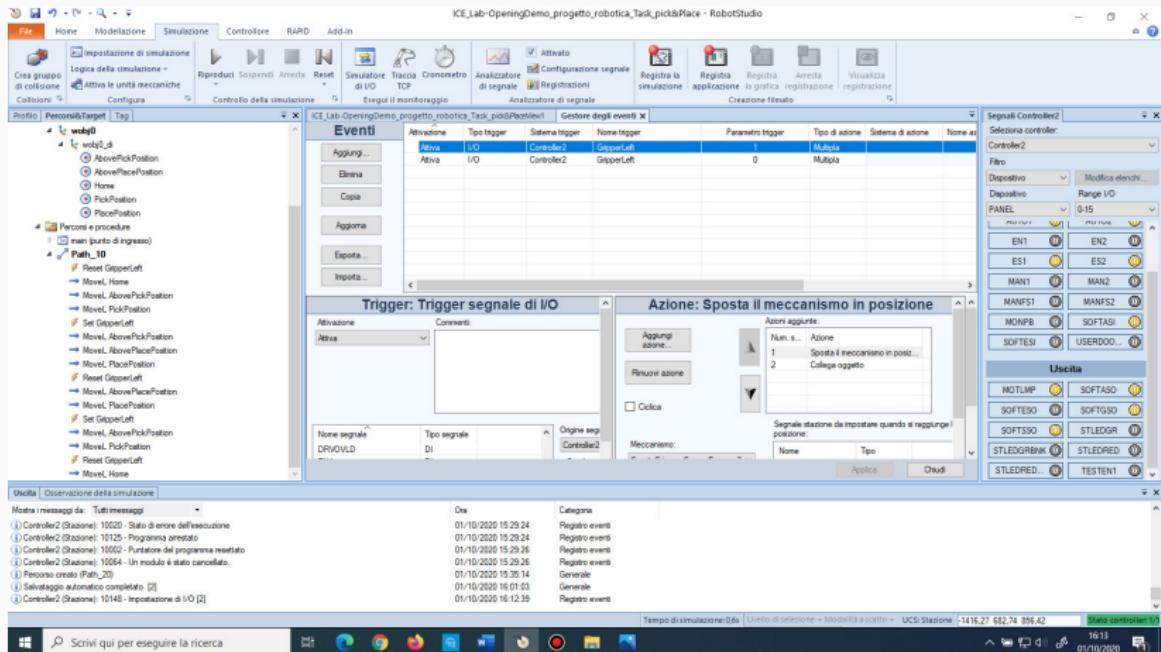


Figure 27: Segnale DO

## Percorso e segnali DO

Una volta creati i segnali necessari, ho riavviato il controllore virtuale e inserito all'interno del task i segnali di **Set Gripper** per definire la **presa dell'oggetto** e **Reset Gripper** per definire il **rilascio dell'oggetto**.

I segnali DO di Set e Reset, li ho relazionati con l'apertura e chiusura del meccanismo del Gripper, rispettivamente attraverso la SynchPose e HomePose.

Nel fare ciò che ho appena descritto, ho premuto il tasto destro sul percorso, Cliccando su "Inserisci istruzione azione".

# Percorso e segnali DO

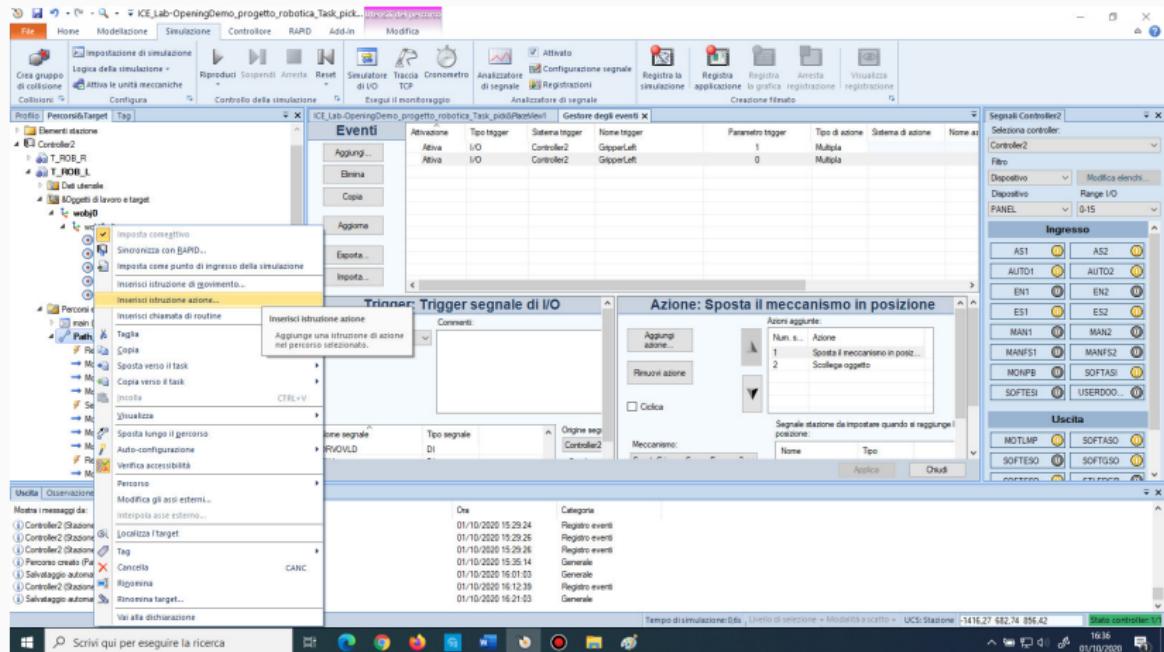


Figure 28: Percorso

# Percorso e segnali DO

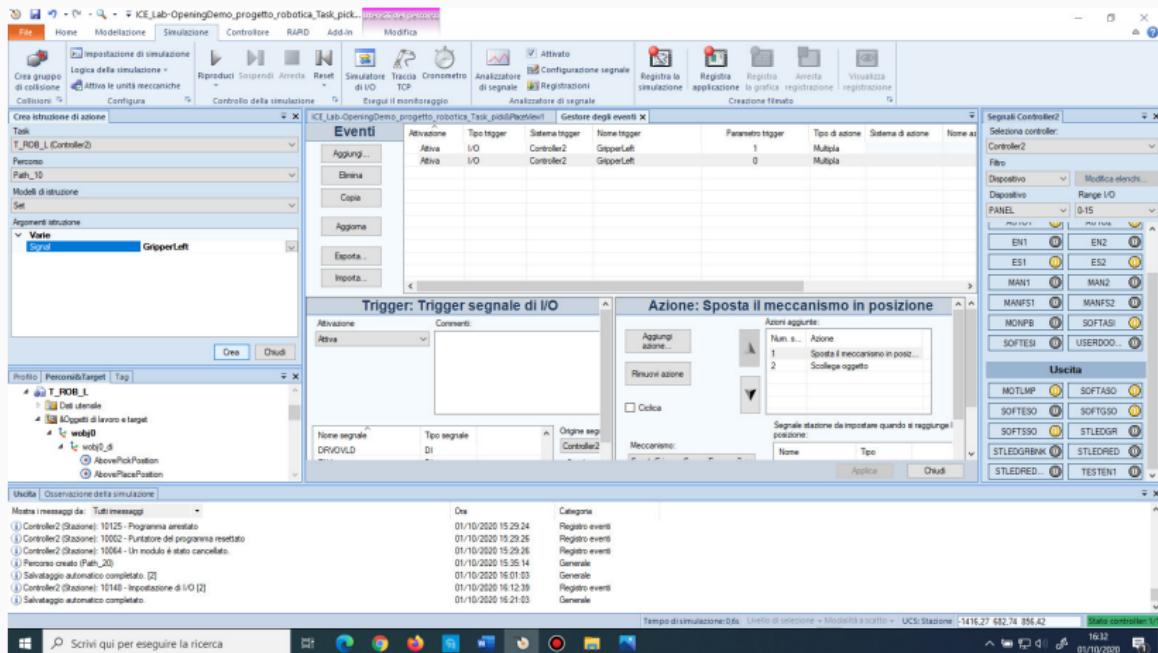


Figure 29: Segnale Set DO

# Percorso e segnali DO

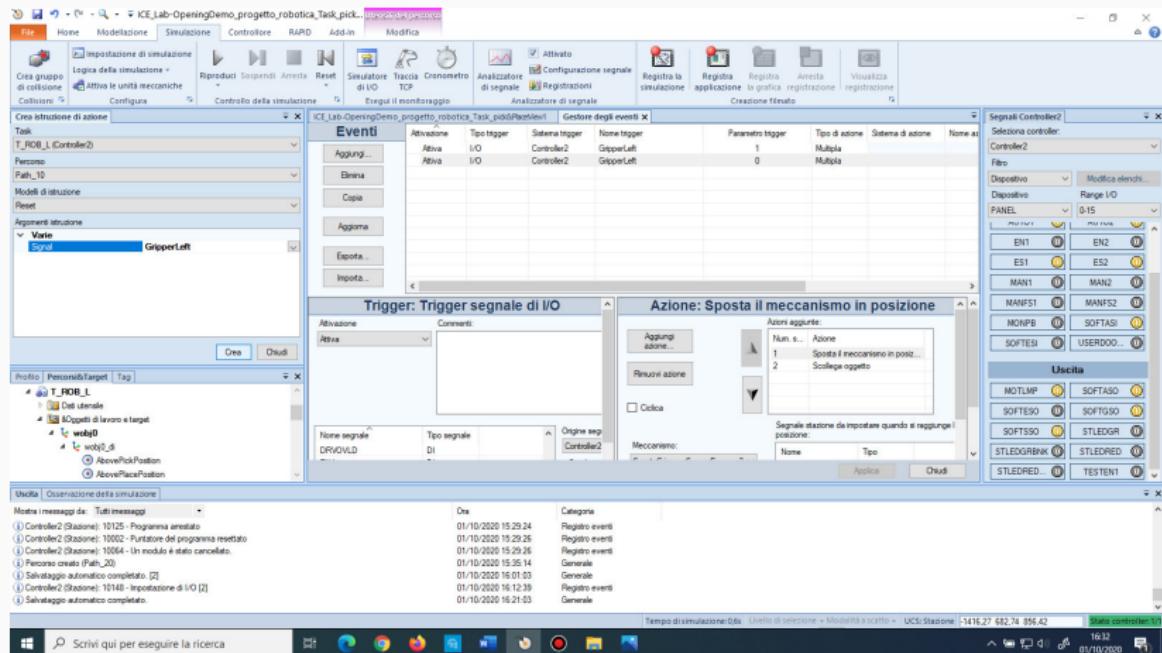


Figure 30: Segnale Reset DO

# Percorso e segnali DO

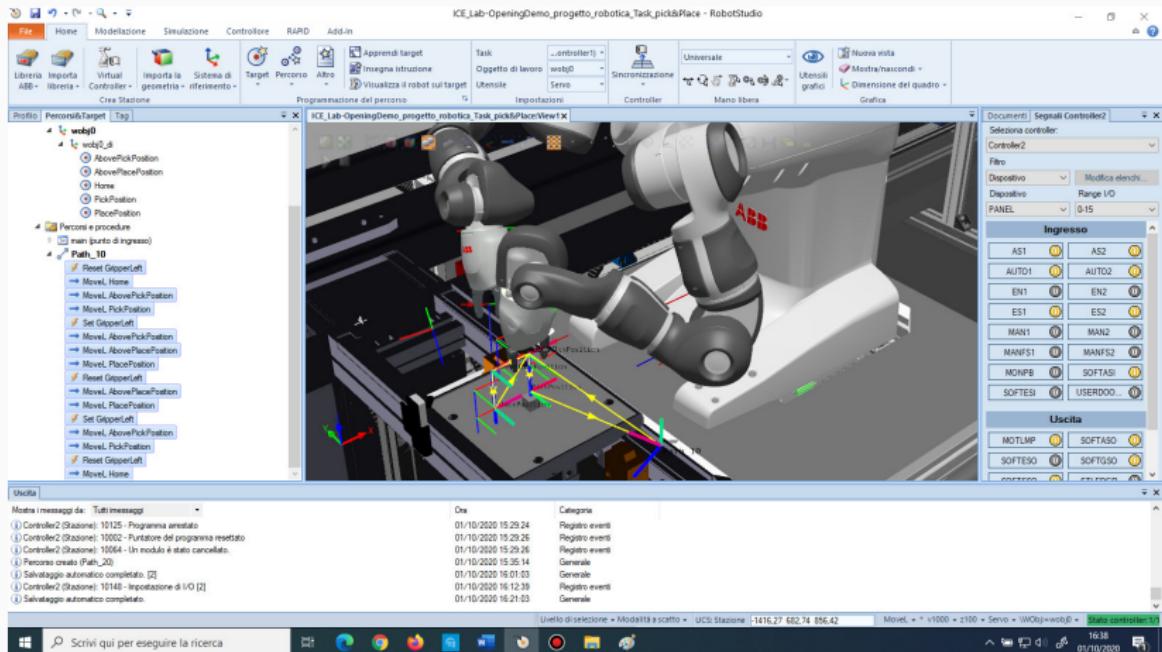


Figure 31: Percorso

## Simulazione

Una volta delineato il percorso è stato possibile creare una simulazione vera e propria.

La simulazione in questo caso, richiede dei moduli definiti in linguaggio RAPID, i quali verranno richiamati ciclicamente, se impostato, un metodo che definisce il percorso descritto in precedenza. Sono andato nella scheda Home e cliccando su "Sincronizzazione" e poi "Sincronizzazione RAPID" è possibile ottenere il percorso dei target scritto in RAPID.

# Percorso e segnali DO

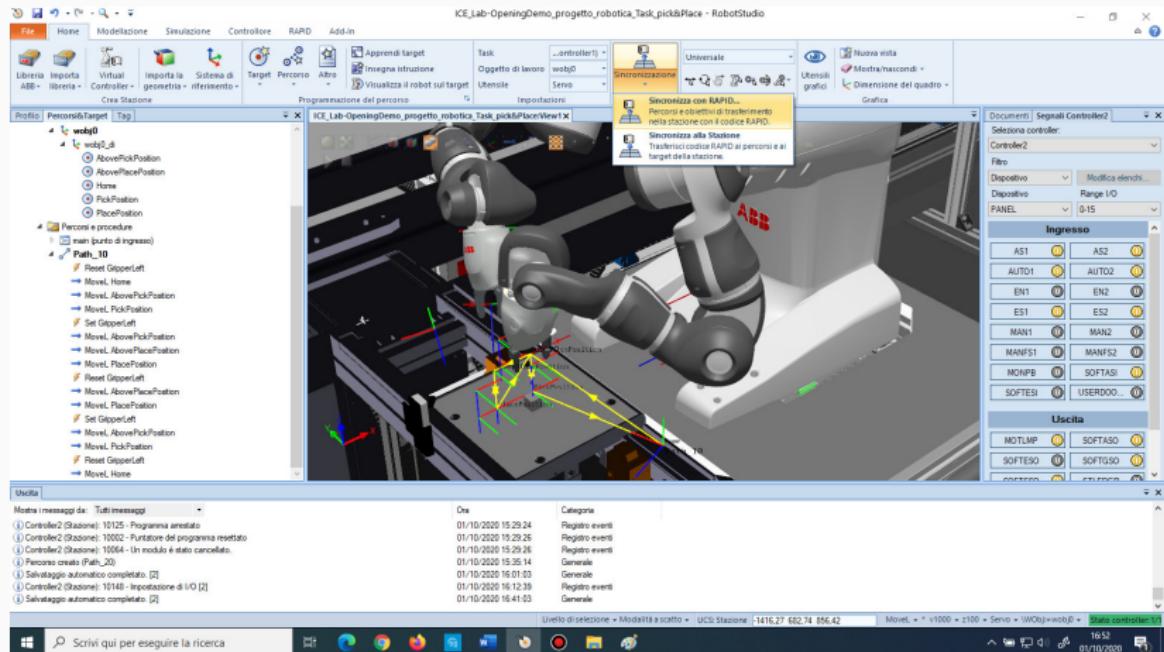


Figure 32: Sincronizzazione Rapid

Recandomi poi nella scheda RAPID e cliccando sul modulo main è stato possibile aprire l'editor in RAPID ed eventualmente modificare alcune istruzioni.

# Modulo RAPID

main.mod

## Simulazione

Una volta che i moduli RAPID sono correttamente compilabili è possibile avviare la simulazione.

Per prima cosa ho sincronizzato la stazione con le eventuali modifiche apportate al modulo main e successivamente andando sulla scheda Simulazione e cliccando il pulsante "Riproduci" è stato possibile far eseguire la simulazione.

Di seguito alcuni **video** del task completato

# Simulazione

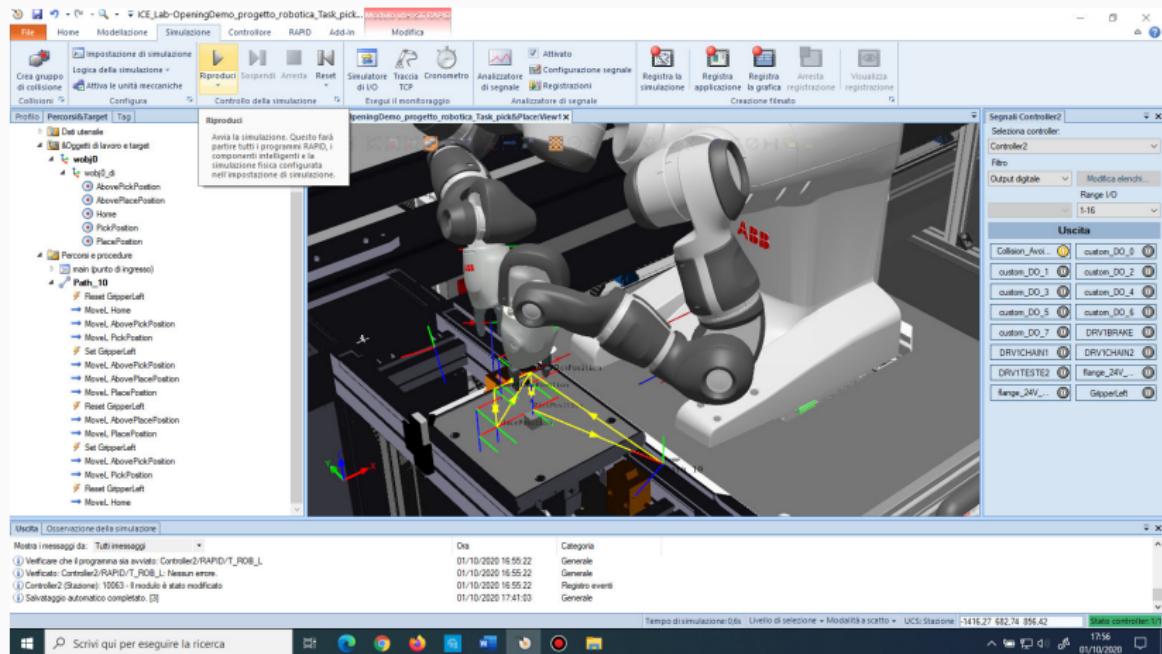


Figure 33: Avvio simulazione

# Video Pick&Place

# Video Pick&Place

# Video Pick&Place

## Secondo Task

- Nel secondo task ho cercato di riprodurre lo scambio di due oggetti differenti creando anche qui come prima due cubi 20x20x20 di colore differente (rosso e verde).  
Nel dettaglio, il braccio sinistro posiziona l'oggetto in un lato del piano di lavoro, successivamente attende che il braccio destro posizioni l'oggetto al centro e prenda quello che il braccio sinistro ha posizionato nel lato del piano riportandolo nella posizione iniziale. Il braccio sinistro prenderà poi l'oggetto posto nel mezzo e lo rimette nella propria posizione. Anche qui, come prima, ho definito differenti target per poter creare tutte queste azioni e ho configurato di conseguenza la cinematica inversa e i differenti orientamenti del Gripper.

## Temporizzazione

A differenza del precedente task qui la complessità è nata sullo scambio degli oggetti in uno spazio molto ristretto.  
Questo ha comportato una sorta di "**sincronizzazione**", (teoricamente definita come **temporizzazione**), tra le braccia per poter riuscire ad evitare la collisione. Come detto in precedenza se il robot prevede che ci sia una collisione blocca l'esecuzione e non è possibile proseguire con la simulazione.

# Collision Avoidance

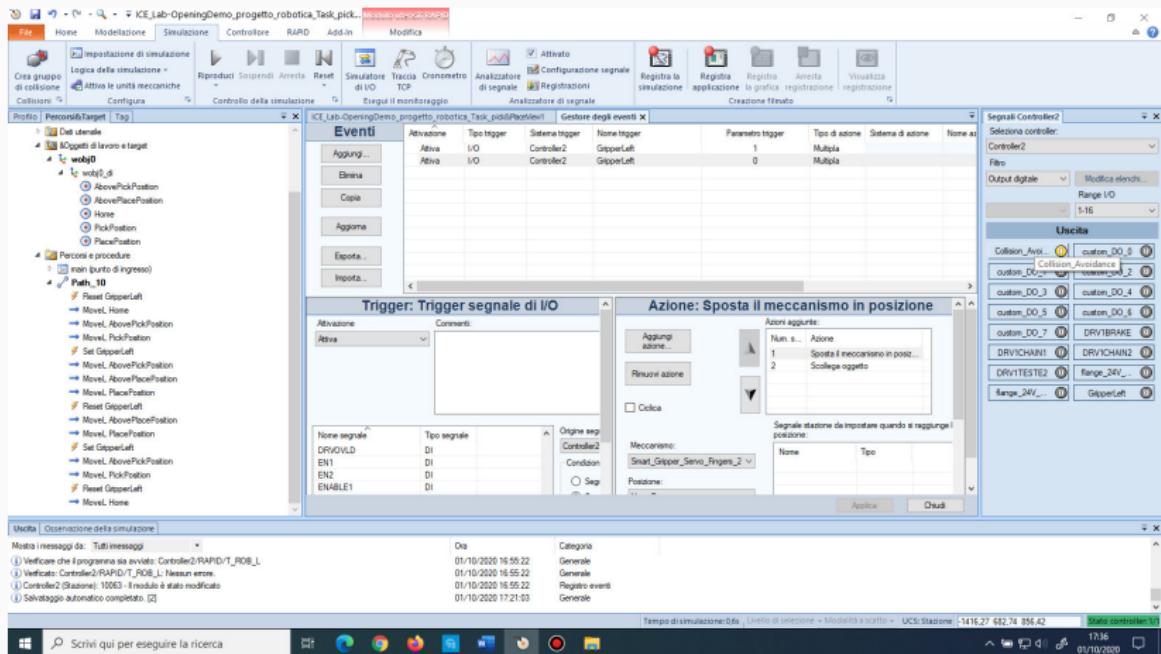


Figure 34: Collision Avoidance

## Percorsi

In questo caso, a differenza del task precedente ho definito due differenti percorsi, uno per il braccio destro e uno per quello sinistro.

# Percorso braccio destro

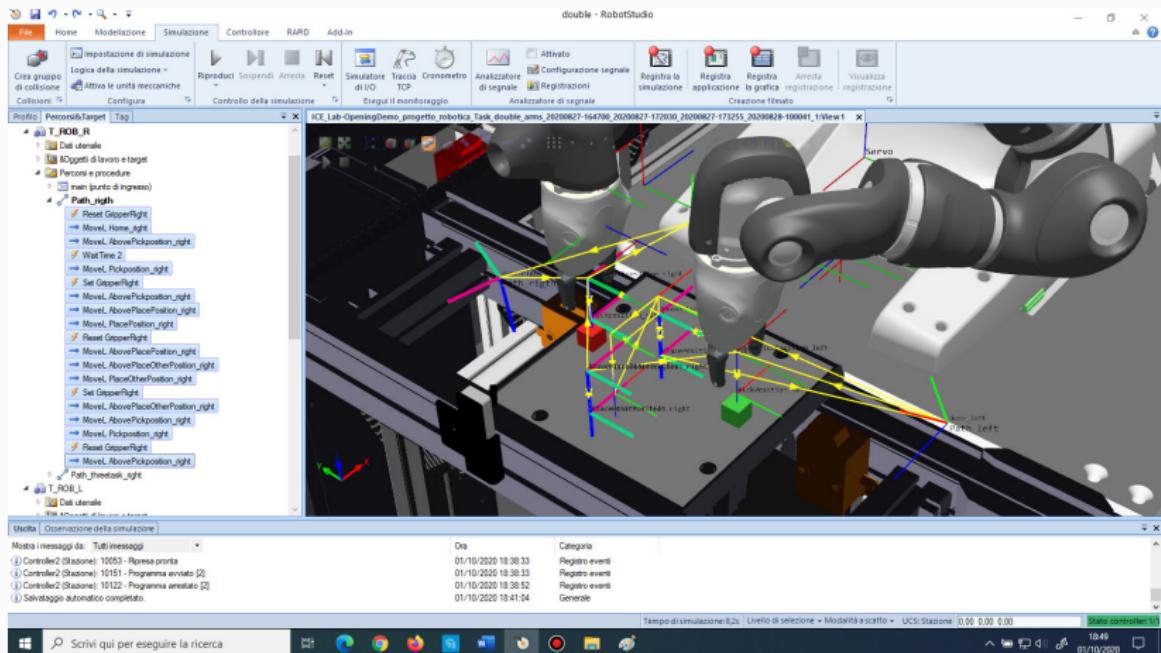


Figure 35: Percorso Braccio destro

# Percorso braccio sinistro

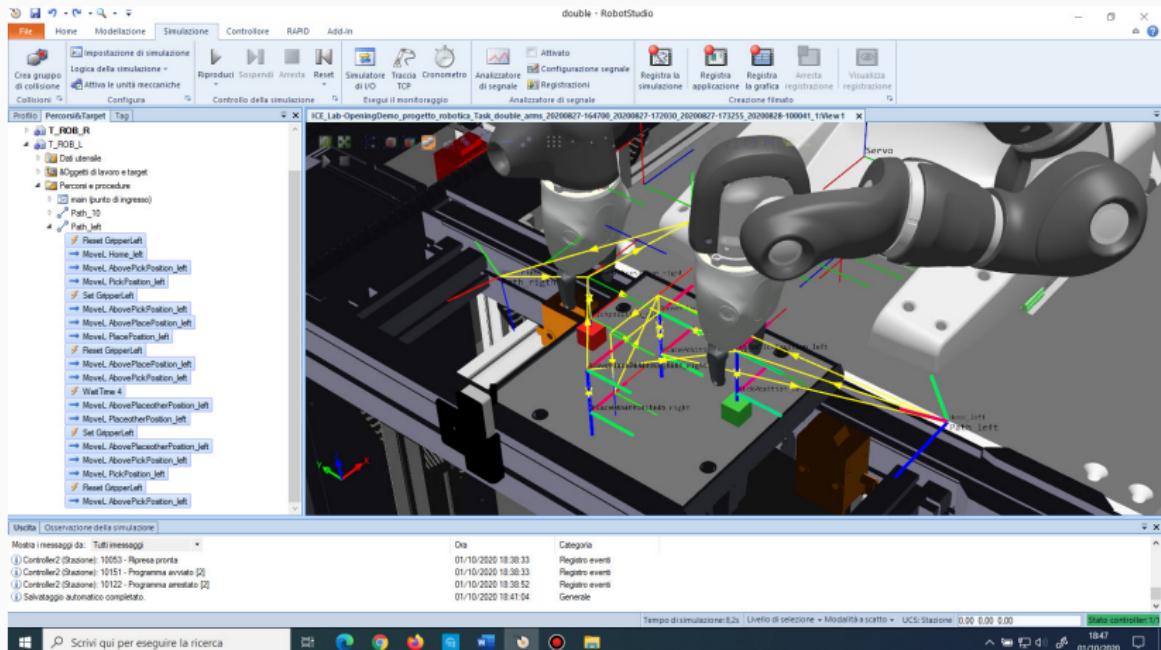


Figure 36: Percorso braccio sinistro

Ancora una volta ho creato il modulo in RAPID per avviare la simulazione.

A differenza di prima, però, in questo caso ho definito due moduli, uno per il braccio destro e uno per il sinistro. Quando la simulazione verrà riprodotta verranno richiamati entrambi i moduli main, rispettivamente per il braccio destro e sinistro.

codice rapid braccio destro

codice rapid braccio sinistro

# Video

# Video

# Video

# Problemi

La difficoltà in questo task è stata principalmente nel capire come "**sincronizzare**" le due braccia affinché la simulazione non continuasse a fermarsi per **prevenire** la **collisione**.

## Collisione avvenuta

VIDEO collisione

## Terzo task

- In questo task ho ricreato il passaggio di un oggetto, (ho sostituito il cubo con un parallelepipedo di 20x20x40, che sono andato a denominare box), tre le 2 braccia dello yumi, creando un cambio di orientamento in corso, raccogliendo il box in verticale con braccio dx (appoggiato sul lato 20x20), trasferendolo poi in orizzontale (lato lungo parallelo al piano di lavoro) e appoggiandolo in un altro punto con il braccio sx

## Temporizzazione

In questo task come nel precedente, ho dovuto inserire all'interno dei task delle singole braccia dei **temporizzatori** per ricreare la "**Sincronizzazione**". In questo caso è essenziale questa fase, in quanto nel momento in cui un braccio tiene l'oggetto parallelo al piano, dovrà aspettare il secondo braccio prima di poter rilasciare l'oggetto.

# Percorso braccio destro

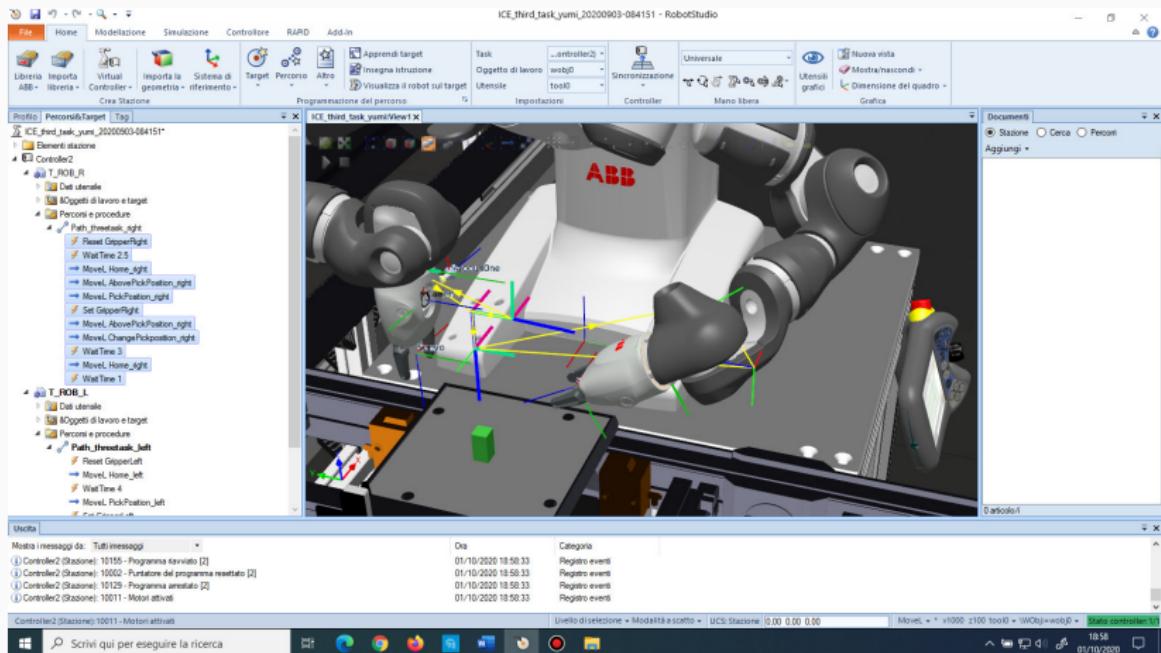


Figure 37: Percorso braccio destro

# Percorso braccio sinistro

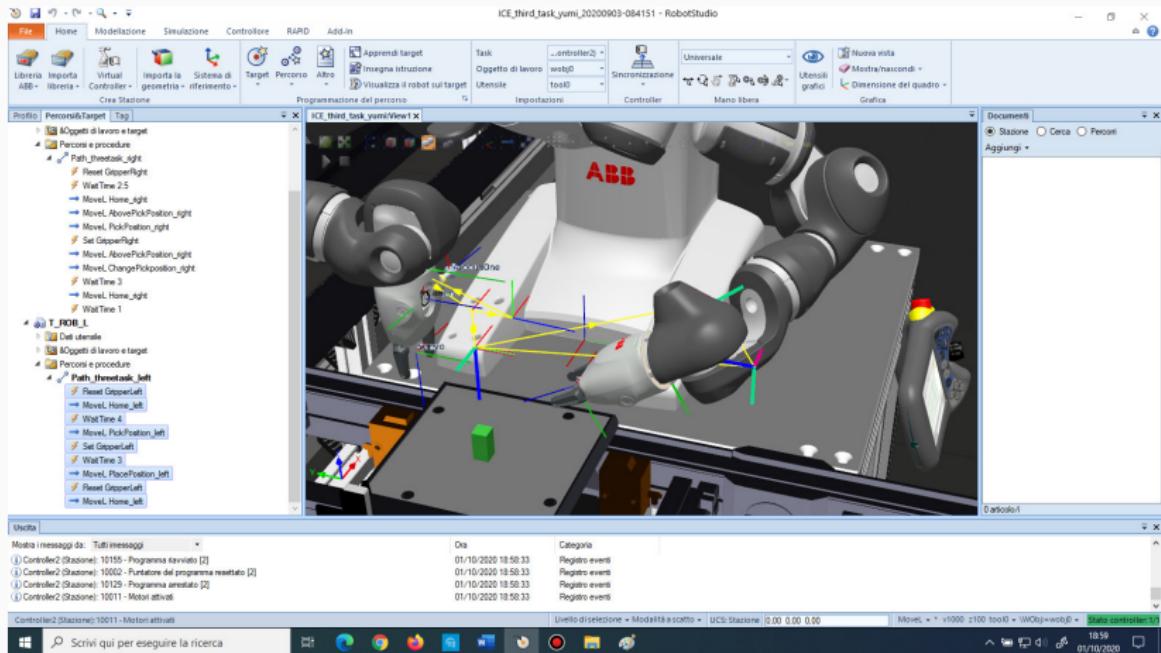


Figure 38: Percorso braccio sinistro

Come prima anche qui ho utilizzato i moduli in linguaggio RAPID per ricreare la simulazione, richiamando i main delle rispettive braccia.

codice rapid braccio destro

codice rapid braccio sinistro

# Video

# Video

# Video

## Problematiche

In questo task a differenza degli altri la difficoltà è stata nel cambio di orientazione di entrambe le braccia. Questo perché come mostrato in precedenza, finché la configurazione non è corretta la simulazione non può essere eseguita correttamente.

## Programmazione Yumi in ambiente reale

L'ultima parte del progetto è stata quella di ricreare i task eseguiti in fase di simulazione, nel mondo reale.

## Programmazione Yumi: Step

Per potermi collegare al Robot reale, ho dovuto connettermi alla rete di Ateneo e successivamente tramite VPN di ICE.

*Successivamente, ho testato i miei task per capire se l'ambiente era o meno conforme con l'ambiente reale.*

Quindi se l'origine del modello coincideva con l'origine del mondo reale (ICE).

## Step

Per poter testare i miei task sul robot reale mi sono collegato al robot e ho creato una **relazione** tra i **miei task** e la **memoria** del robot. In questo modo, ho potuto inserire tutta la mia stazione con i differenti moduli all'interno dello spazio di memoria. Attraverso il **FlexPendant** è stato possibile interagire con il robot e riuscire a simulare i task, stoppare e reimpostare delle coordinate.

## FlexPendant

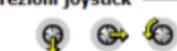
La FlexPendant è un'unità operatore ad azionamento manuale utilizzata per eseguire molti dei task necessari per azionare un sistema robotico, come l'esecuzione dei programmi, lo spostamento manuale del manipolatore, la modifica dei programmi del robot e così via.

La FlexPendant è costituita da hardware e software ed è in realtà un vero e proprio computer.



**Figure 39:** FlexPendant

La modalità di movimento e/o il sistema di coordinate selezionati determinano il modo in cui viene mosso il robot.

Modalità di movimento	Illustrazione del joystick	Descrizione
Lineare	<p>Direzioni joystick</p>  <p>X Y Z</p> <p>en0400001131</p>	La modalità lineare è descritta nel paragrafo <a href="#">Impostazione dell'orientamento dell'utensile a pagina 154</a> .
Assi 1, 2 e 3 (impostazione predefinita per i robot)	<p>Joystick directions</p>  <p>2 1 3</p> <p>en0300000536</p>	La modalità Asse 1-3 è descritta nel paragrafo <a href="#">Spostamento asse per asse a pagina 155</a> .
Assi 4, 5 e 6	<p>Joystick directions</p>  <p>5 4 6</p> <p>en0300000537</p>	La modalità Asse 4-6 è descritta nel paragrafo <a href="#">Spostamento asse per asse a pagina 155</a> .

**Figure 40:** Movimenti FlexPendant

## Step

Con il **FlexPendant** è stato possibile eseguire alcuni movimenti all'interno del task, simulandoli step-by-step.

Come accennato precedentemente, in questo caso l'ambiente non era perfettamente come quello simulato, quindi attraverso la modalità **Enable Lead-through** ho posizionato correttamente la posa del braccio, sovrascrivendo le posizioni che avevo in fase di simulazione, mentre alcune pose del tipo, AbovePick le ho riconfigurate non attraverso il Lead-through, ma spostando solamente con il joystick il braccio linearmente in coordinate cartesiane (x,y,z).

## Step

Con il **FlexPendant** è stato possibile eseguire alcuni movimenti all'interno del task, simulandoli step-by-step.

Come accennato precedentemente, in questo caso l'ambiente non era perfettamente come quello simulato, quindi attraverso la modalità **Enable Lead-through** ho posizionato correttamente la posa del braccio, sovrascrivendo le posizioni che avevo in fase di simulazione, mentre alcune pose del tipo, AbovePick le ho riconfigurate non attraverso il Lead-through, ma spostando solamente con il joystick il braccio linearmente in coordinate cartesiane (x,y,z).

- **Enable/Disable Lead-through** significa che è possibile afferrare le braccia del robot e spostarle manualmente alla posizione desiderata, in alternativa allo spostamento.

## Step

Dopo aver terminato la ri-configurazione dei vari punti dei percorsi che dovevo eseguire per il secondo e terzo task, ho modificato anche il codice RAPID.

In particolare, a differenza della simulazione alcune delle istruzioni viste precedentemente le ho dovute cambiare. I segnali creati per la presa e il rilascio dell'oggetto gli ho dovuti modificare secondo le seguenti istruzioni:

- **Hand\_init();** Posto all'interno del main, serve per riportare il robot nella configurazione iniziale e resettare i Gripper

In particolare, a differenza della simulazione alcune delle istruzioni viste precedentemente le ho dovute cambiare. I segnali creati per la presa e il rilascio dell'oggetto gli ho dovuti modificare secondo le seguenti istruzioni:

- **Hand\_init();** Posto all'interno del main, serve per riportare il robot nella configurazione iniziale e resettare i Gripper
- **Hand\_GripOutward;** è il comando in sostituzione al segnale Reset Gripper

In particolare, a differenza della simulazione alcune delle istruzioni viste precedentemente le ho dovute cambiare. I segnali creati per la presa e il rilascio dell'oggetto gli ho dovuti modificare secondo le seguenti istruzioni:

- **Hand\_init();** Posto all'interno del main, serve per riportare il robot nella configurazione iniziale e resettare i Gripper
- **Hand\_GripOutward;** è il comando in sostituzione al segnale Reset Gripper
- **Hand\_Stop;** posto assieme al comando precedente serve a dire al Robot di aprire e mantenere aperto il Gripper

- **WaitRob InPos;** è l'istruzione che precede la presa dell'oggetto, ed è importante perché determina il fatto che il robot prima di prendere l'oggetto e aprire il Gripper deve essere nella posizione definita

- **WaitRob InPos;** è l'istruzione che precede la presa dell'oggetto, ed è importante perché determina il fatto che il robot prima di prendere l'oggetto e aprire il Gripper deve essere nella posizione definita
- **Hand\_GripInward;** è il comando in sostituzione al segnale Set Gripper

## RAPID e Sincronizzazione

In questo caso è possibile definire la sincronizzazione vera e propria delle due braccia del robot. L'istruzione definita precedentemente, WaitTime [tempo], non è del tutto corretta, in quanto, trattandosi di temporizzazione, non è detto che sia comune a tutti i robot.

Ecco perché ho utilizzato delle **variabili di Sincronizzazione**:

- PERS tasks

`T_syncRobot2:=[[“T_ROB_R”],[“T_ROB_L”]];` Per la definizione del task di tipo Synch

## RAPID e Sincronizzazione

In questo caso è possibile definire la sincronizzazione vera e propria delle due braccia del robot. L'istruzione definita precedentemente, WaitTime [tempo], non è del tutto corretta, in quanto, trattandosi di temporizzazione, non è detto che sia comune a tutti i robot.

Ecco perché ho utilizzato delle **variabili di Sincronizzazione**:

- **PERS tasks**  
`T_syncRobot2:=[[“T_ROB_R”],[“T_ROB_L”]];` Per la definizione del task di tipo Synch
- **VAR syncident T\_sync\_startL;** L'uso di questa variabile task di Synch su una variabile di synch inserita all'interno del codice

## RAPID e Sincronizzazione

In questo caso è possibile definire la sincronizzazione vera e propria delle due braccia del robot. L'istruzione definita precedentemente, WaitTime [tempo], non è del tutto corretta, in quanto, trattandosi di temporizzazione, non è detto che sia comune a tutti i robot.

Ecco perché ho utilizzato delle **variabili di Sincronizzazione**:

- **PERS tasks**  
`T_syncRobot2:=[[“T_ROB_R”],[“T_ROB_L”]];` Per la definizione del task di tipo Synch
- **VAR syncident T\_sync\_startL;** L'uso di questa variabile task di Synch su una variabile di synch inserita all'interno del codice
- **WaitSyncTask T\_sync\_startL, T\_syncRobot;** L'istruzione per la synch delle braccia

# Codice Rapid

codice braccio destro

# Codice Rapid

codice braccio sinistro

## Simulazione

Una volta definiti i moduli RAPID è stato possibile definire la simulazione. In questo caso, tramite il FlexPendant è stato possibile avviare la simulazione.

- Prima di avviare la simulazione ho dovuto selezionare i task che volevo eseguire, premere PP Main per definire la posizione iniziale ed essere sicuro che il robot sia pronto per la simulazione e alla fine premere il pulsante "Play" per avviare la simulazione

# Video

# Video

## Conclusione

---

# Conclusione

Questo progetto è stato molto istruttivo, in quanto mi ha permesso di capire il funzionamento di un tool e la programmazione di un robot in fase di simulazione. È stato possibile per alcuni aspetti ritrovare ciò che in teoria abbiamo studiato nel corso di Robotica.

L'ultima parte del progetto è stata molto interessante perché mi ha fatto capire la differenza del "mondo simulato" e quello "reale", e quali sono le problematiche principali nel passare da un tool simulativo ad un robot reale. L'uso di RobotStudio ABB non permette l'immediata esecuzione della simulazione nell'ambiente reale e le varie problematiche mi hanno fatto capire quale siano le procedure reali che devono essere messe in pratica per poter riuscire a creare i task definiti tramite tool.

**Fine**

---

*Grazie per l'attenzione*