Università Politecnica delle Marche

Ingegneria Informatica e dell'Automazione

# RoboSimCloth


Corso di

Laboratorio di Automazione

Anno accademico 2024-2025

Studenti:
Pizzuto Andrea
Meloccaro Lorenzo
Percipalle Noemi

Professore:
Andrea Bonci

Dottorandi:
Serafini Andrea
Pellicani Ilaria
Di Biase Alessandro

Dipartimento di Ingegneria dell'Informazione

# Indice

# 1 Introduzione

*Il capitolo fornisce una panoramica del progetto, descrivendo l'obiettivo principale di sviluppare e simulare la movimentazione di un robot TM-900 con un tessuto, attraverso l'integrazione di ROS2 e Unity. Viene fornita una panoramica delle motivazioni per l'uso di questi software e delle principali fasi del progetto.*

Il presente progetto ha come obiettivo lo sviluppo e la simulazione della manipolazione dei tessuti con il robot collaborativo TM5-900, utilizzando l'integrazione di software avanzati come ROS2 e Unity. Il contesto applicativo del progetto si colloca nell'ambito industriale e robotico, ponendo particolare attenzione sull'ottimizzazione dei processi di smantellamento di oggetti deformabili, come abiti e tessuti. L'azienda coinvolta utilizza il software Cloth 3D per la modellazione degli abiti, mentre Unity e Gazebo sono impiegati per simulare la movimentazione del robot e l'interazione con gli oggetti. Uno degli obiettivi di questo progetto è quello di analizzare in dettaglio l'integrazione tra ROS2 e Unity, scelta motivata dalle potenzialità di questi due strumenti. ROS2 offre una gestione efficiente della comunicazione tra il robot e il sistema di simulazione, mentre Unity fornisce la possibilità di simulare l'effetto della gravità sui tessuti e i movimenti del robot in tempo reale tramite una grafica avanzata. L'integrazione di questi due software è stata scelta per soddisfare la necessità di creare un ambiente simulativo realistico, utile non solo per eseguire test simulativi, ma anche per implementare successivamente il sistema su un robot reale. In questo modo, l'integrazione tra ROS2 e Unity consente di gestire il robot e simularlo in ambiente virtuale, migliorando l'efficienza e la precisione nelle operazioni robotiche. Il progetto affronta diversi task, tra cui la valutazione e la compatibilità dei tessuti con i vari software, la simulazione della movimentazione del robot e dell'oggetto da disassemblare in un ambiente condiviso, l'acquisizione e la definizione di una sequenza di fasi di smantellamento, l'esecuzione delle fasi in simulazione con il robot e, infine, l'esecuzione delle stesse fasi su un robot reale.

# 2 Materiali e Metodi

*In questo capitolo saranno presentati i materiali(hardware e software) e i metodi, quindi le tecniche utilizzate, inoltre vengono dettagliate le tecnologie impiegate e le procedure seguite, in modo da permettere la riproducibilità del progetto. In particolare la prima sezione,* ***Hardware*** *comprende la descrizione del robot utilizzato in simulazione Omron TM5-900 e della camera RealSense D435i. Successivamente saranno presentati i vari* ***Software****, utilizzati all'interno del progetto e che hanno permesso la riuscita dell'applicazione, tra i quali Ubuntu, ROS2, Moveit, Gazebo, Unity e in particolare lo strumento Cloth di Unity. Infine, verranno descritte nel dettaglio la* ***Predisposizione dei Software nel PC*** *e le modalità di* ***Interfacciamento tra ROS 2 e Unity****, con focus su strumenti fondamentali come ROS-TCP Connector, ROS-TCP Endpoint e URDF Importer.*

## 2.1 Hardware

*In questa sezione verranno presentati i principali componenti hardware utilizzati nel progetto, in particolare il robot Omron TM5-900 e la Stereo-Camera RealSense D435i. Saranno forniti dettagli tecnici e funzionali su ciascun componente, evidenziando le loro caratteristiche principali e il loro ruolo all'interno del sistema di simulazione.*

### 2.1.1 Omron TM5-900

Un robot è un manipolatore riprogrammabile e multifunzionale progettato per spostare materiali, parti, strumenti o dispositivi specializzati attraverso vari movimenti programmati per l'esecuzione di una varietà di compiti. Il **robot OMRON TM5-900** è un robot industriale collaborativo (cobot) di ultima generazione, progettato per applicazioni che richiedono flessibilità, precisione e la capacità di operare in ambienti dinamici e collaborativi. È dotato di una serie di caratteristiche avanzate che lo rendono adatto a una varietà di compiti in ambienti industriali, inclusi il montaggio, il pick-and-place, e operazioni di smontaggio come quelle previste in questo progetto.

Tra le caratteristiche principali vi sono:

- **Design compatto e flessibile**: Il TM5-900 è progettato per lavorare in spazi ristretti, con un braccio robotico che può essere facilmente integrato in linee di produzione esistenti senza necessitare di ampi spazi operativi. La sua flessibilità gli consente di essere facilmente usato per diverse applicazioni.

- **Portata e capacità di carico**: Il TM5-900 ha una capacità di carico utile fino a 5 kg, che lo rende adatto a manipolare una varietà di oggetti e strumenti, come i tessuti utilizzati in questo progetto.

- **Gradi di libertà (DOF)**: Il TM5-900 è dotato di 6 gradi di libertà (DOF), che gli consentono di eseguire movimenti complessi e articolati, simili a quelli di un braccio umano.


**Figure 2.1:** *Omron TM5-900*

Questo permette al robot di raggiungere e manipolare oggetti in spazi tridimensionali con alta precisione.

- **Giunti e movimenti**: Il robot è equipaggiato con giunti motorizzati, che consentono di controllare i movimenti di rotazione in modo fluido e preciso. I giunti, insieme ai sensori di forza e coppia, permettono di eseguire operazioni delicate, come il manipolamento di oggetti morbidi (ad esempio i tessuti) senza danneggiarli.
- **Facilità di programmazione e utilizzo**: Questo robot è dotato di un'interfaccia utente intuitiva e di software di programmazione che ne facilita l'uso anche da parte di personale non specializzato. La programmazione è resa semplice grazie alla modalità di "teaching", che consente di insegnare al robot i movimenti direttamente sul campo.
- **Tecnologia di sensori avanzati**: Il TM5-900 è equipaggiato con sensori di forza e coppia, che permettono di eseguire operazioni delicate con una precisione millimetrica, ideale per manipolare oggetti senza danneggiarli.
- **Compatibilità con ROS2**: Il robot è compatibile con il sistema ROS2, che consente di integrarlo facilmente in ambienti di simulazione come Gazebo e Unity, come previsto nel progetto. Questo permette di controllare il robot tramite comandi remoti e di simulare il suo comportamento in ambienti virtuali.

In questo progetto, il robot OMRON TM5-900 è stato scelto per la sua versatilità e capacità di manipolare oggetti come tessuti in modo preciso e sicuro. Il suo utilizzo si concentra sulla simulazione della movimentazione del robot, dove l'integrazione con ROS2

assicura una gestione ottimale della comunicazione e delle operazioni. Inoltre, l'interazione con Unity permette di visualizzare in tempo reale il comportamento del robot, monitorando e ottimizzando il suo funzionamento durante le diverse fasi del progetto.

### 2.1.2 Stereo-Camera RealSense D435i

## 2.2 Software

*In questo capitolo verranno analizzati i software principali utilizzati per il progetto: Ubuntu, Robot Operating System 2, Moeveit, Gazebo e Unity. Verranno fornite una panoramica generale, informazioni sul loro utilizzo e sulle versioni adottate. Inoltre, si approfondirà l'uso del componente Cloth di Unity, che ha rivestito un ruolo centrale nella simulazione della fisica dei tessuti.*

### 2.2.1 Ubuntu

*Per consentire l'esecuzione degli strumenti necessari allo sviluppo del progetto, è stato necessario configurare un ambiente Linux compatibile con ROS2. A tal fine, si è scelto di utilizzare Ubuntu, installato tramite WSL (Windows Subsystem for Linux), una soluzione che permette di lavorare in ambiente Linux direttamente da un sistema operativo Windows.*


**Figure 2.2:** *Logo Ubuntu*

Per lo sviluppo del progetto è stato utilizzato il sistema operativo **Ubuntu 22.04.5**, una distribuzione Linux molto diffusa nel campo della robotica grazie alla sua ampia compatibilità con **ROS2 (Robot Operating System 2)**.

L'installazione di Ubuntu è avvenuta tramite WSL (Windows Subsystem for Linux), una funzionalità disponibile su Windows 10 e 11 che consente di eseguire nativamente un ambiente GNU/Linux all'interno di Windows, senza dover ricorrere a una macchina virtuale o a un sistema dual boot. In particolare, è stata utilizzata la versione WSL2, che offre prestazioni migliorate rispetto a WSL1 grazie all'integrazione di un vero e proprio kernel Linux. Questa soluzione permette di combinare la potenza e la flessibilità di Ubuntu con la praticità degli strumenti di sviluppo offerti dall'ambiente Windows.

Per seguire passo dopo passo l'installazione, è disponibile una guida video al seguente link: https://www.youtube.com/watch?v=bOwsWabST-0

**Di seguito, i principali passaggi eseguiti per l'installazione:**

1. Abilitazione delle funzionalità WSL e Virtual Machine Platform
   - Aprire il menu Start e cercare *Windows PowerShell*.
   - Avviare PowerShell come amministratore.
   - Digitare il comando:

     ```
     wsl --install
     Questo comando installa WSL2 come versione predefinita.
     ```

Al termine dell'installazione, sarà necessario riavviare il computer.

1. Installazione di Ubuntu 22.04
   - Dopo aver abilitato WSL2, è possibile installare Ubuntu digitando:

     ```
     wsl --install -d Ubuntu-22.04
     ```

Al termine dell'installazione, sarà necessario **riavviare il computer**.
Configurazione iniziale

- Al primo avvio, verrà richiesto di creare un utente e una password Linux.
- Infine, è consigliato aggiornare i pacchetti eseguendo:

  ```
  sudo apt update && sudo apt upgrade
  ```

Questa configurazione ha permesso l'installazione di ROS2 e l'integrazione con librerie e strumenti specifici per la robotica, mantenendo una buona compatibilità con gli altri software utilizzati nel progetto, come Unity e Gazebo.

### 2.2.2 Robot Operating System 2


**Figure 2.3:** *Logo ROS2*

**ROS 2** (Robot Operating System 2) [1] è un framework open-source per lo sviluppo di applicazioni robotiche. Fornisce una serie di strumenti, librerie e convenzioni per facilitare la comunicazione tra i componenti software di un sistema robotico. ROS 2 è progettato per migliorare la scalabilità, la sicurezza e la compatibilità con i sistemi distribuiti rispetto al suo predecessore, ROS 1. Il suo utilizzo è diffuso in ricerca e industria per il controllo di robot mobili, bracci robotici e veicoli autonomi. In questo progetto, ROS 2 è stato utilizzato per gestire la comunicazione tra il robot simulato in Unity e i componenti di controllo. ROS 2 permette lo scambio di messaggi tra i nodi, abilitando il controllo del robot da Unity e viceversa. In particolare, il framework ha permesso l'integrazione con il sistema di simulazione, la gestione delle traiettorie e il controllo dei giunti del manipolatore. Inizialmente, il progetto è stato avviato con ROS 2 Jazzy, ma si sono riscontrati diversi problemi di compatibilità e instabilità, in particolare con l'integrazione del ROS-TCP Connector per Unity. Per questo motivo, è stato deciso di passare a ROS 2 Humble, una versione più stabile e ampiamente supportata, che ha garantito una migliore compatibilità con gli strumenti di sviluppo utilizzati.

### 2.2.3 Moveit


**Figure 2.4:** *Logo Moveit*

**MoveIt** [2] [3] è una delle librerie più utilizzate per la pianificazione del movimento in ROS. Consente di controllare bracci robotici, eseguire pianificazioni di traiettoria, gestire collisioni e implementare strategie di manipolazione avanzate. Grazie alla sua integrazione con ROS 2, MoveIt permette di generare e simulare movimenti in modo efficiente, facilitando l'implementazione di algoritmi di pianificazione e controllo.

MoveIt utilizza il modello del robot descritto tramite URDF per comprendere la struttura cinematica del sistema e genera automaticamente il corrispettivo SRDF (Semantic Robot Description Format) contenente la generazione di movimenti validi evitando collisioni. Il suo framework è composto da diversi moduli, tra cui:

- Motion Planning: per la generazione di traiettorie.
- Collision Checking: per evitare urti tra il robot e l'ambiente.
- Kinematic Solvers: per calcolare i movimenti articolari necessari a raggiungere una determinata posizione.
- Trajectory Execution: per inviare comandi al robot fisico o alla simulazione.

Nel nostro progetto, MoveIt è stato utilizzato per pianificare e simulare i movimenti del manipolatore Omron TM5-900. Tramite il MoveIt Setup Assistant è stato possibile definire i gruppi cinematici, i giunti controllabili e le geometrie di collisione.

MoveIt è stato integrato con ROS 2 Humble tramite la versione Moveit2 e tramite questa libreria sono state svolte le seguenti operazioni:

- Creazione del file SRDF tramite MoveIt Setup Assistant.
- Definizione dei controller nel file $ros2_controllers.yaml$.
- Definizione delle pose iniziali e finali del robot.
- Invio delle traiettorie a Unity per visualizzare e simulare il movimento del robot in un contesto 3D realistico.

Di seguito sono riportati i passaggi svolti per la configurazione del package moveit_config_tm5_900 utilizzando il MoveIt Setup Assistant [4]:

1. Lanciamo il MoveIt Setup Assistant con il comando

```
roslaunch moveit_setup_assistant setup_assistant.launch
```

2. Clicchiamo su "Create New MoveIt Configuration Package" e selezioniamo il file URDF del robot
3. Nella sezione "Self Collision" è possibile generare automaticamente il controllo delle auto-collisioni del robot, selezionando "Generate Collision Matrix".
4. Tramite "Virtual Joint" è possibile definire il giunto virtuale del robot, che rappresenta la connessione tra il robot e il mondo esterno. In questo caso, abbiamo selezionato "fixed" come tipo di giunto.
5. Nella sezione "Planning Groups" aggiungere il gruppo "Manipulator" che comprende tutti i giunti a partire da "shoulder_1_ joint" fino a "wrist_3_joint". Se nel URDF è presente la pinza è possibile aggiungere anche il gruppo "Gripper".
6. Successivamente tramite "Robot Poses" sono state definite due pose, una di riposo(Home) e una di lavoro(Work). In caso di aggiunta della pinza si possono definire due pose della pinza in modo che il robot possa afferrare oggetti e manipolarli.
7. Infine generiamo i controllori "Ros2 Controllers" e "Ros2 MoveIt Controllers" per la comunicazione tra il robot e MoveIt. Per il nostro progetto abbiamo selezionato rispettivamente il controller "JointTrajectoryController" e "FollowJointTrajectory" per il gruppo "Manipulator".
8. A questo punto rimane solo da generare il package tramite "Genera Package" nella sezione "Configurations Files".

MoveIt ha svolto un ruolo centrale nell'interazione tra ROS 2 e Unity e ci ha permesso di simulare la movimentazione del robot solo tramite due codici, uno in C# per Unity e uno in Python per ROS2, che verranno poi approfonditi nella sezione 6. Le traiettorie generate da MoveIt sono state pubblicate sul topic $/manipulator_controller/joint_trajectory$,

successivamente sottoscritto in Unity tramite il pacchetto ROS-TCP Connector. Questo ha permesso una rappresentazione visiva coerente tra la pianificazione in ROS 2 e l'esecuzione nella simulazione 3D.

Inoltre è stato possibile utilizzare il package generato per avviare RViz, un potente strumento di visualizzazione per ROS, che consente di monitorare e analizzare i movimenti del robot in tempo reale. RViz è stato utilizzato per verificare la correttezza della pianificazione delle traiettorie e per eseguire test preliminari prima di passare alla simulazione in Unity.

Tramite il comando:

```
ros2 launch moveit_config_tm5_900 demo.launch.py
```

è possibile avviare il package generato e visualizzare il robot in RViz, in quanto l'URDF del robot è già presente nel package.

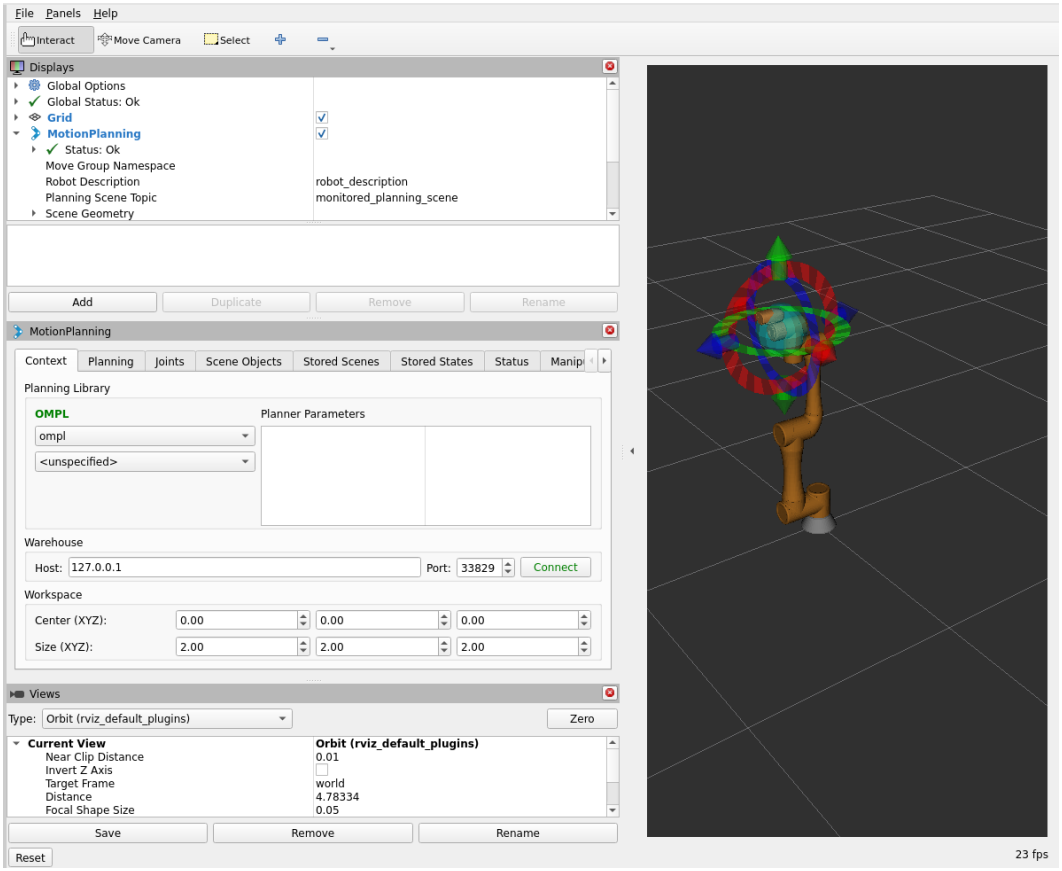


**Figure 2.5:** *Schermata iniziale RViz*

Dalla figura 2.5 è possibile vedere la schermata iniziale una volta lanciato il comando sopra indicato. A questo punto, nella sezione "Planning", indicando il gruppo di giunti, una posizione iniziale e finale del robot, come mostrato in figura 2.6, RViz salverà le posizioni dei giunti di entrambe le pose. Successivamente, cliccando su "Plan & Execute" il robot si muoverà nella posizione finale indicata, simulanod la traiettoria.

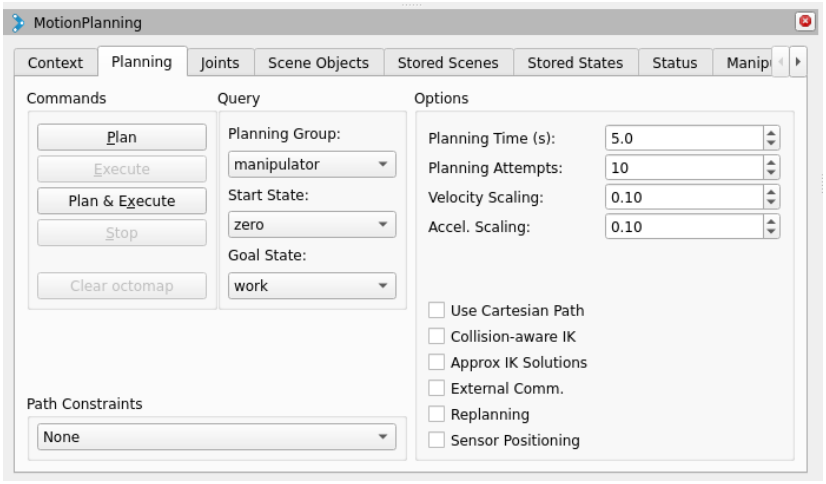


**Figure 2.6:** *Configurazione della pianificazione ed esecuzione della traiettoria in RViz*

Non solo, tramite RViz è inoltre possibile creare nuove pose come mostrato in figura 2.7, e testare le traiettorie in breve tempo.

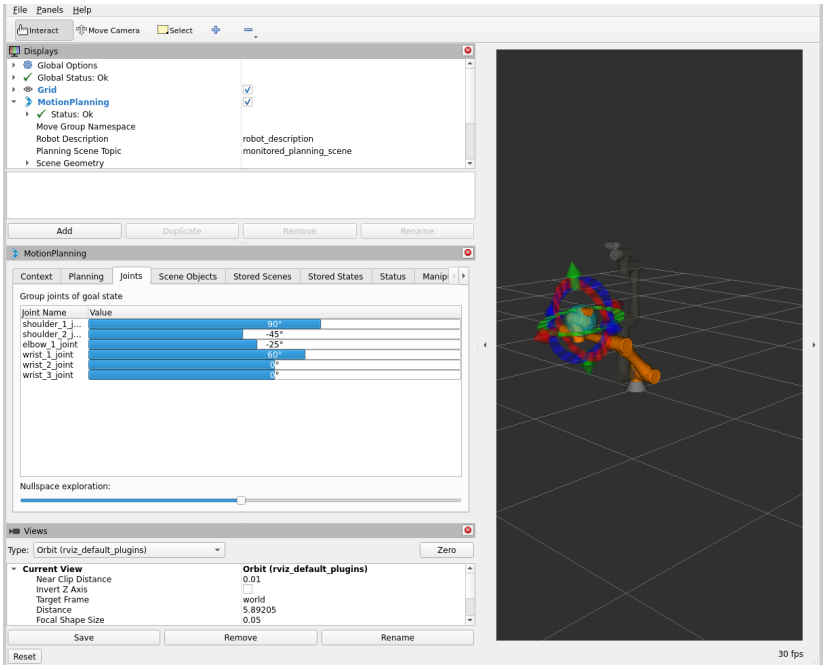


**Figure 2.7:** *Configurazione della posizione dei giunti in RViz*

Nel mentre, è risultato molto utile controllare il terminale durante queste operazioni perché Moveit riportava eventuali errori o problemi di collisione tra il robot e l'ambiente circostante. Questo ha permesso di ottimizzare le traiettorie e garantire un movimento fluido e sicuro del robot.

### 2.2.4 Gazebo

### 2.2.5 Unity e UnityHUB

**Figure 2.8:** *Logo Unity*

**Unity** [5] è un motore di gioco e simulazione ampiamente utilizzato per la creazione di ambienti interattivi in tempo reale. Sebbene sia nato per lo sviluppo di videogiochi, il suo utilizzo si è esteso ad applicazioni di simulazione, robotica, realtà virtuale e aumentata. Unity permette di creare ambienti 3D realistici e interattivi grazie al suo motore grafico avanzato e alle sue funzionalità di scripting basate su C#. In questo progetto, Unity è stato utilizzato come ambiente di simulazione per il robot. Grazie alla sua compatibilità con ROS2 tramite il ROS-TCP Connector, Unity ha permesso di visualizzare il robot, simulare i suoi movimenti e interagire con l'ambiente circostante. Il motore fisico di Unity è stato sfruttato soprattutto per replicare le dinamiche fisiche della maglietta, garantendo una simulazione più realistica delle interazioni tra un indumento e gli oggetti della scena. Per lo sviluppo del progetto, abbiamo utilizzato Unity 6, la versione più recente al momento di sviluppo del progetto.

### 2.2.6 Strumento Cloth di Unity

Il sistema **Cloth di Unity** offre una soluzione basata sulla fisica per simulare tessuti e materiali flessibili all'interno di ambienti 3D. Sebbene sia stato progettato principalmente per rappresentare abbigliamento su personaggi, può essere utilizzato anche per altri scopi, come bandiere, tende o qualsiasi altro oggetto che richieda una simulazione realistica del comportamento dei tessuti[6]. In questo progetto, il componente Cloth riveste un ruolo centrale, poiché il compito finale prevede che l'indumento manipolato dal robot presenti una fisica il più possibile realistica, simulando accuratamente il comportamento del tessuto. A tal fine, sono stati forniti tre file principali relativi a una maglietta: un file **OBJ**, un file **FBX** e un file **Collada**. L'analisi di questi file ha permesso di ottenere informazioni utili riguardo al formato da utilizzare e alle caratteristiche della mesh per applicare efficacemente il componente Cloth.
Tra la documentazione di Unity è possibile trovare anche quella relativa al componente Cloth e in seguito verranno riassunti i passaggi fondamentali per utilizzarlo al meglio. Per implementare una simulazione di tessuto in Unity, è necessario aggiungere il componente Cloth a un oggetto mesh. Ecco i passaggi fondamentali:

- **Preparazione della Mesh**: è importante assicurarsi che l'oggetto a cui si desidera applicare il tessuto abbia una mesh adeguata, e quindi, come visto in precedenza, anche un numero adeguato di poligoni della mesh.
- **Aggiunta del Componente**: una volta importato il file su Unity è sufficiente selezionare l'oggetto nella gerarchia di Unity e, nel pannello Inspector, cliccare su "Add Component", selezionare "Physics" e poi "Cloth" dalla lista dei componenti disponibili.
- **Configurazione**: Una volta aggiunto il componente, sarà possibile modificare vari parametri per ottenere una simulazione personalizzata e accurata.

Affinchè la maglietta abbia una movimentazione guidata dalla fisica, deve chiaramente avere una parte della maglietta fissata e che quindi non è soggetta alla gravità. Questi su Unity vengono chiamati "Cloth Constraint" e quindi "vincoli del Cloth". Di seguito una descrizione dei passaggi per applicarli correttamente alla maglietta:

1. sul componente Cloth si clicca su "Edit Cloth Constraint" come possiamo vedere dalla figura 2.9

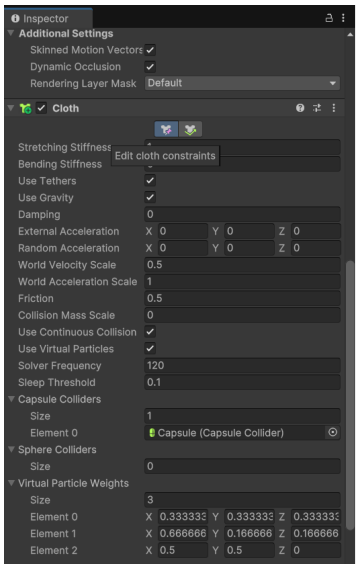


**Figure 2.9:** *Schermata Unity per l'applicazione dei vincoli*

2. successivamente si aprirà una schermata che permetterà di selezionare (tramite "Select") oppure colorare (tramite "Paint") le parti che saranno vincolate e quindi rimarranno maggiormente rigide. E' importante specificare una distanza minima nel parametro "Max Distance", in figura 2.10 è stato assegnato il valore 0.2 ed ha permesso un buon risultato.

Infine una descrizione delle principali proprietà del componente Cloth e di come sono state utilizzate nell'implementazione del tessuto realistico della maglietta:

- **Stretching Stiffness**: Determina la resistenza del tessuto all'allungamento. Valori più alti rendono il tessuto meno incline a estendersi. Un parametro accettabile per la maglietta è risultato compreso tra 0,5 e 0,7.

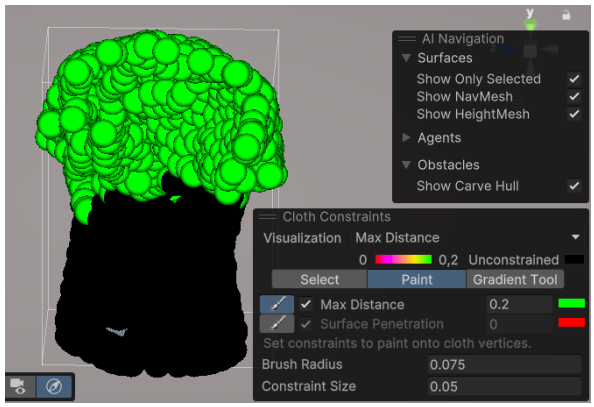


**Figure 2.10:** *Schermata Unity per la selezione delle parti della mesh a cui applicare i vincoli*

- **Bending Stiffness**: Controlla la rigidità alla flessione del tessuto. Un valore elevato riduce la capacità del tessuto di piegarsi. Un parametro accettabile per la maglietta è risultato compreso tra 0,7 e 0,9.
- **Use Tethers**: Applica vincoli che aiutano a prevenire che le particelle mobili del tessuto si allontanino troppo da quelle fisse, riducendo l'eccessiva elasticità.
- **Use Gravity**: Indica se la gravità deve influenzare il tessuto. Da applicare nel caso in cui si voglia vedere una movimentazione fisica della maglietta e condizionata dalla gravità.
- **Damping**: Coefficiente che determina quanto velocemente il movimento del tessuto si smorza nel tempo. Un parametro accettabile per la maglietta è risultato compreso tra 0,2 e 0,4.
- **External Acceleration**: Applica un'accelerazione costante esterna al tessuto, utile per simulare effetti come il vento.
- **Random Acceleration**: Introduce un'accelerazione casuale al tessuto, aggiungendo variazioni imprevedibili nel movimento.
- **World Velocity Scale**: Determina quanto il movimento in spazio globale dell'oggetto influisce sui vertici del tessuto.
- **World Acceleration Scale**: Controlla l'influenza dell'accelerazione globale dell'oggetto sui vertici del tessuto.
- **Friction**: Imposta il coefficiente di attrito del tessuto durante le collisioni. Un parametro accettabile per la maglietta è risultato compreso tra 0,5 e 0,7.
- **Collision Mass Scale**: Determina l'incremento di massa delle particelle durante le collisioni.
- **Use Continuous Collision**: Abilita la collisione continua per migliorare la stabilità delle interazioni.

- **Use Virtual Particles**: Aggiunge particelle virtuali per migliorare la stabilità delle collisioni.
- **Solver Frequency**: Specifica il numero di iterazioni del solver per secondo, influenzando la precisione della simulazione. Un parametro accettabile per la maglietta è risultato 60 Hz.
- **Sleep Threshold**: Definisce la soglia sotto la quale il tessuto entra in stato di "sonno", interrompendo la simulazione fino a nuove interazioni. Un parametro accettabile per la maglietta è risultato 0,1.
- **Capsule Colliders**: Array di collisori a capsula con cui il tessuto può interagire.
- **Sphere Colliders**: Array di coppie di collisori sferici con cui il tessuto può interagire.

## 2.3 Predisposizione dei Software nel PC

*In questo capitolo viene presentata la configurazione dell'ambiente di lavoro adottato per lo sviluppo del progetto, con particolare attenzione alle versioni e alle modalità di integrazione dei principali software utilizzati: Ubuntu, ROS2, Gazebo, Unity (insieme a Unity Hub) e MoveIt2.*

Per la parte robotica e di simulazione, è stato scelto di adottare **Ubuntu 22.04** installato tramite una macchina virtuale WSL2, in quanto offre un ambiente stabile e supportato ufficialmente per lo sviluppo con **ROS2 Humble** e **MoveIt2**. ROS2 Humble è stato selezionato per la sua affidabilità e le migliorate capacità di integrazione, che hanno risolto i problemi riscontrati con precedenti versioni (come Jazzy).**Gazebo Fortress**, la cui scelta della versione è dovuta alla compatibilità con ROS2[7], garantisce simulazioni avanzate e un'elevata accuratezza nella riproduzione delle dinamiche robotiche, tuttavia non ha un ruolo centrale nel progetto in quanto non è possibile applicare la fisica ai tessuti come in Unity.

Parallelamente, per lo sviluppo dell'ambiente 3D e della visualizzazione in tempo reale, si è mantenuto **Unity 6**(ultima versione) su Windows, supportato da **Unity Hub** che permette una gestione semplificata dei vari progetti Unity. La scelta di utilizzare Windows come sistema operativo per Unity è motivata dalla sua eccellente compatibilità con gli strumenti grafici e i driver necessari per un'interfaccia utente fluida e performante, in quanto Unity è un software nativo Windows. La comunicazione tra l'ambiente robotico gestito tramite Ubuntu e quello grafico su Windows è resa possibile attraverso il ROS-TCP Connector, che funge da ponte tra le due piattaforme, garantendo lo scambio efficiente dei dati.

Questa configurazione ibrida, che sfrutta al massimo le potenzialità di ogni sistema operativo e software specifico, consente di ottimizzare sia le prestazioni delle simulazioni che l'interattività dell'interfaccia grafica, contribuendo a rendere il progetto complessivamente robusto e flessibile.

## 2.4 Interfacciamento ROS2-Unity

### 2.4.1 ROS-TCP Connector & ROS-TCP Endpoint

Il **ROS-TCP Endpoint** è il nodo ROS2 complementare al **ROS-TCP Connector**, che permette a Unity di comunicare con ROS tramite la rete. Questo nodo gestisce le connessioni TCP in ingresso provenienti da Unity e inoltra i messaggi ROS corrispondenti ai topic.

## Installazione del ROS-TCP Endpoint

Per utilizzare il **ROS-TCP Endpoint**, è necessario installarlo all'interno del proprio workspace ROS2:

- **Clonare il repository**: `cd /tuo_workspace/src`
- **Compilare il workspace**: `cd /tuo_workspace`

```
colcon build --symlink-install
source install/setup.bash
```

## Avvio del nodo

Per avviare il nodo **ros_tcp_endpoint**, bisogna eseguire il comando:

```
ros2 launch ros_tcp_endpoint endpoint.launch.py
```

In questo modo si avvia il server che ascolta sulla porta TCP (di default 10000) per ricevere le connessioni da Unity.

### Verifica del nodo

Per assicurarsi che il nodo sia in esecuzione, è possibile utilizzare il comando:

```
ros2 node list
```

In questo modo il nodo **/ros_tcp_endpoint_node** dovrebbe essere visibile nella lista.

### Verifica della connessione

Per testare la connessione tra Unity e ROS2 bisogna:

- **1.** Avviare ROS2 ed eseguire il nodo `ros_tcp_endpoint`.
- **2.** Eseguire la simulazione in Unity e verificare che non vengano generati errori di connessione.
- **3.** Utilizzare `ros2 topic echo` per controllare i messaggi pubblicati da Unity verso ROS2 e viceversa.

Questa configurazione permette di integrare Unity e ROS2 in un ambiente di simulazione realistico, abilitando lo scambio di dati tra il motore grafico e il framework robotico per la pianificazione e il controllo dei movimenti.

### ROS-TCP Connector

Il **ROS-TCP Connector** è un pacchetto sviluppato da Unity Technologies che permette di mettere in comunicazione ROS2 e Unity attraverso una connessione TCP. Questa integrazione è fondamentale perché consente di inviare e ricevere messaggi tra i due ambienti, facilitando lo sviluppo e la simulazione di robot. Grazie a questo pacchetto, Unity diventa un ambiente in cui è possibile visualizzare i dati provenienti da ROS2, testare il controllo e verificare il comportamento del robot in scenari simulati prima di passare alla fase di implementazione su un sistema reale.

Il pacchetto ROS-TCP Connector include:

- **ROS-TCP Connector**: permette l'invio e la ricezione di messaggi tra Unity e ROS2.
- **VISUALIZATIONS PACKAGE**: utile per la visualizzazione dei messaggi in entrata e in uscita nella scena di Unity, facilitando il debug e l'analisi dei dati.

Inoltre, il pacchetto offre diverse funzionalità:

- **ROSConnection**: gestisce l'intero processo di comunicazione tra Unity e ROS2.
- **Message Generation**: permette di generare automaticamente classi in C# che rappresentano i messaggi ROS2, facilitando l'interazione tra i due ambienti.
- **Visualizations**: un set di API e configurazioni predefinite per rappresentare le informazioni scambiate.
- **ROSGeometry**: una serie di estensioni utili per la conversione delle geometrie tra Unity e altri sistemi che semplificano la compatibilità tra gli ambienti di simulazione.

### Installazione del ROS-TCP Connector

Per utilizzare il ROS-TCP Connector, è necessario installarlo all'interno di Unity e configurarlo correttamente affinché possa dialogare con ROS2.

Il pacchetto è disponibile su GitHub e può essere importato in Unity tramite il Package Manager. Ecco come fare:

- Aprire Unity e accedere al menu "Window", poi selezionare "Package Manager".
- Nella finestra del Package Manager, cliccare sul pulsante "+" in alto a sinistra e selezionare "Add package from git URL...".
- Inserire l'URL del pacchetto da installare:
  - **ROS-TCP Connector**: `https://github.com/Unity-Technologies/ROS-TCP-Connector.git?path=/com.unity.robotics.ros-tcp-connector`

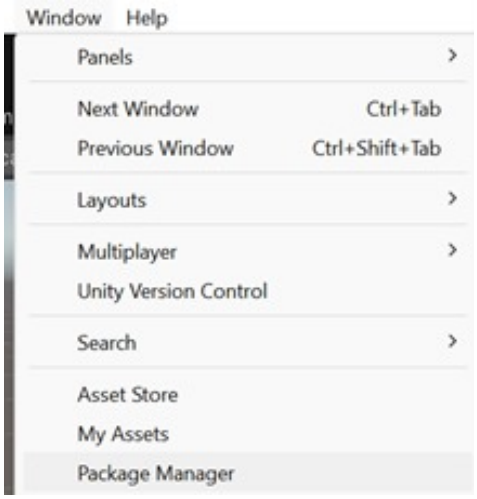


**Figure 2.11:** *Package Manager in Unity*

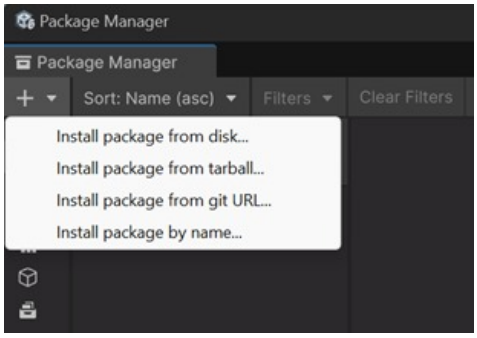


**Figure 2.12:** *Package Manager in Unity*

  - **Visualizations Package**: `https://github.com/Unity-Technologies/ROS-TCP-Connector.git?path=/com.unity.robotics.visualizations`

- In alternativa, se si ha una copia locale del pacchetto, è possibile installarlo seguendo la guida ufficiale di Unity per l'installazione di pacchetti locali.

### Configurazione del ROS-TCP Connector

Dopo aver installato il pacchetto, è necessario configurarlo per far sì che possa comunicare con ROS2. Questo passaggio è essenziale per garantire che Unity e ROS2 possano scambiarsi dati senza problemi.

Dopo aver caricato l'oggetto, per configurare la comunicazione, bisogna:

- Accedere al menu "Robotics" e selezionare "ROS Settings".

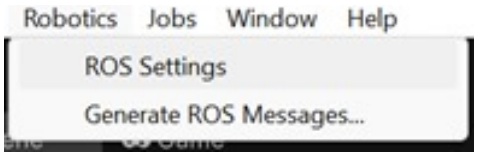


**Figure 2.13:** *Menu "Robotics" e "ROS Settings"*

- Nel campo "ROS IP Address", inserire l'indirizzo IP della macchina su cui è in esecuzione ROS2.

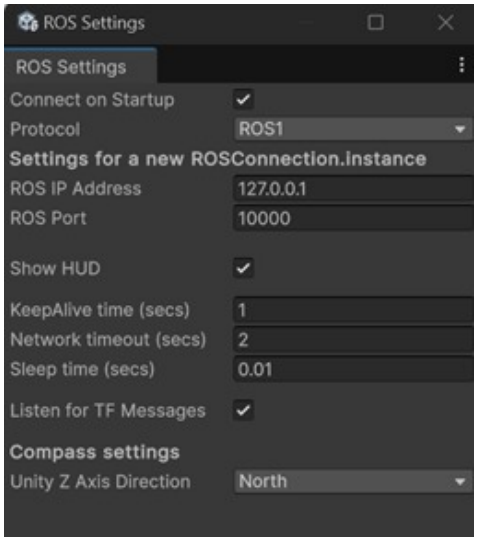


**Figure 2.14:** *Configurazione dell'indirizzo IP di ROS2*

Per reperire l'indirizzo IP di ROS (ROS IP Address) sulla macchina con WSL, è sufficiente aprire un terminale e digitare:

```
hostname -I
```

### Comunicazione tra ROS2 e Unity

Dopo aver completato l'installazione e la configurazione, Unity sarà in grado di comunicare con ROS2 attraverso il protocollo TCP. Questo avviene tramite due modalità principali:

- **Pubblicazione di Messaggi**: Unity può inviare dati a ROS2 utilizzando il componente `ROSPublisher`.
- **Sottoscrizione di Messaggi**: Unity può ricevere dati da ROS2 grazie al componente `ROSSubscriber`.

Per rendere tutto operativo, è necessario avviare il nodo `ros_tcp_endpoint` all'interno di ROS2. Questo nodo funge da intermediario e garantisce che i messaggi possano fluire correttamente tra i due sistemi.

### Testare la connessione tra Unity e ROS2

Una volta configurata la comunicazione, è importante verificare che tutto funzioni correttamente. Ecco alcuni passaggi per effettuare un controllo:

- **1. Avviare ROS2 ed eseguire il nodo `ros_tcp_endpoint`.**
- **2. Digitare il comando per configurare l'ambiente ROS:**

  ```
  source ~/tuo_workspace/install/setup.bash
  ```

- **3. Avviare il nodo tramite il comando:**

  ```
  ros2 launch ros_tcp_endpoint endpoint.launch.py
  ```

  Questo avvierà il nodo `ROS-TCP Endpoint`, che ascolterà le connessioni da Unity sulla porta 10000 (di default).

- **4. È possibile verificare se il nodo è in esecuzione tramite il comando:**

  ```
  ros2 node list
  ```

### 2.4.2 Eseguire simulazione in Unity

- Utilizzare il comando `ros2 topic echo` per verificare se i messaggi vengono effettivamente inviati e ricevuti tra Unity e ROS2.

Se tutto è stato configurato correttamente, Unity e ROS2 saranno perfettamente integrati e potranno scambiarsi dati in tempo reale.

Per concludere, grazie a questa configurazione, sarà possibile avere uno scambio di dati per la pianificazione e il controllo dei movimenti.

### 2.4.3 URDF Importer

L'**URDF Importer** [8] di Unity è uno strumento che permette di importare file URDF (Unified Robot Description Format) direttamente in Unity, consentendo così di visualizzare e interagire con modelli robotici 3D all'interno dell'ambiente di simulazione. Questo strumento risulta particolarmente utile nel progetto perché permette di integrare ROS 2 e Unity tramite il file URDF fortemente utilizzato in robotica, inoltre semplifica la gestione della cinematica e della fisica dei robot senza dover ricostruire manualmente i modelli.

Il formato URDF è ampiamente utilizzato nel mondo della robotica per descrivere la struttura dei robot, inclusi i links, i joints, i materiali e i sensori. L'URDF Importer converte questi modelli in GameObjects di Unity, mantenendo la gerarchia dei componenti e assegnando i giunti come oggetti fisici controllabili.

Nel nostro progetto, il robot utilizzato è un Omron TM5-900, un braccio robotico con sei gradi di libertà. Il modello URDF di questo robot contiene:

- - Link per ogni segmento del braccio, dalla base all'end-effector.
- - Joints di tipo rotoidale che consentono il movimento.
- - Materiali per rappresentare visivamente il robot in Unity.
- - Collisioni e inerzia, che permettono una simulazione fisica realistica.

L'importazione di questo URDF in Unity è essenziale per poterlo visualizzare correttamente e per interagire con esso tramite ROS 2.

Per importare installare lo strumento URDF Importer è possibile seguire i seguenti passaggi:

- Cliccare su "Window" poi "Package Manager"

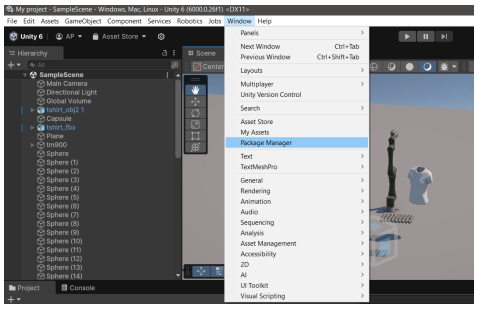


**Figure 2.15:** *URDF Importer Tutorial*

- Cliccare su "+" e selezionare "Add package from git URL..."

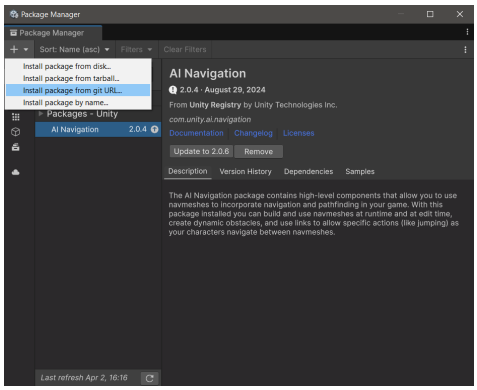


**Figure 2.16:** *URDF Importer Tutorial*

- Inserire il seguente URL della repository: `https://github.com/Unity-Technologies/URDF-Importer.git` e cliccare "install" per avviare l'installazione

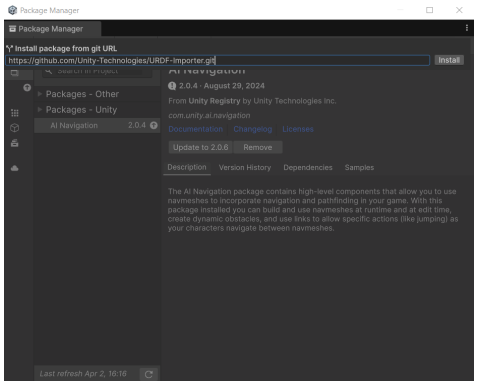


**Figure 2.17:** *URDF Importer Tutorial*

Per importare il modello URDF su Unity invece:

- Importare il file URDF e le varie mesh sia visual che collision nella cartella "Assets" del progetto Unity

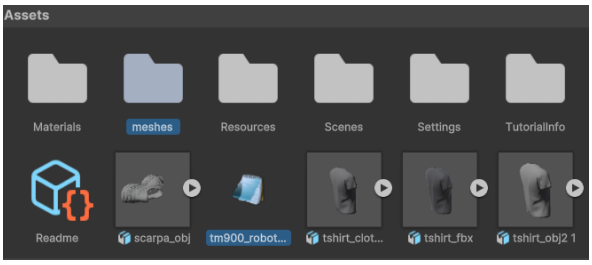


**Figure 2.18:** *URDF Importer Tutorial*

- Verificare che i vari path delle mesh all'interno del file URDF siano del tipo ""package://path/delle/meshes" come mostrato in figura 2.19

```
<link name="base_link">
  <visual>
    <geometry>
      <mesh filename="package://meshes/tm900/visual/Base.STL"/>
    </geometry>
    <material name="Grey">
      <color rgba="0.5 0.5 0.5 1.0"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://meshes/tm900/collision/base.STL"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="1.0"/>
    <insert_block name="origin"/>
    <inertia ixx="0.00110833289" ixy="0.0" ixz="0.0" iyy="0.00110833289" iyz="0.0" izz="0.0018"/>
  </inertial>
</link>
```

**Figure 2.19:** *URDF Importer Tutorial*

- Tasto destro sul file URDF e cliccare su "Import Robot from selected URDF file"

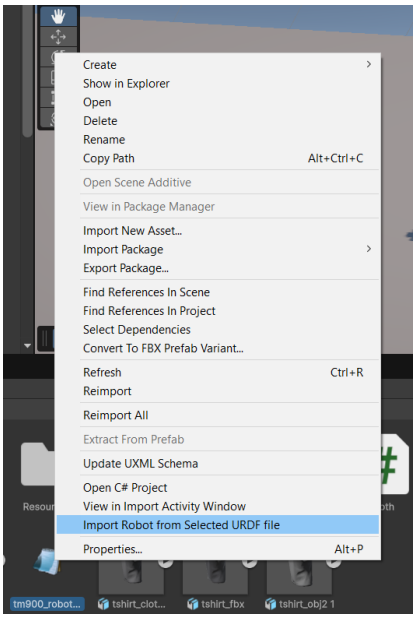


**Figure 2.20:** *URDF Importer Tutorial*

A questo punto, se è stato utilizzato il file urdf del TM5-900 corretto, sarà possibile visualizzare il robot come mostrato in figura 2.21.

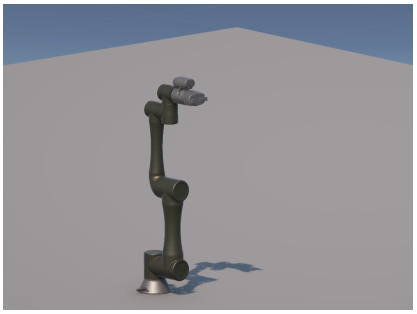

**Figure 2.21:** *URDF Importer Tutorial*

## 2.5 Importazione TM5-900 in Gazebo

## 2.6 Camera

# 3 Analisi Preliminari e Problematiche Riscontrate

## 3.1 Problematiche sulle Compatibilità dei Software Utilizzati

## 3.2 Compatibilità dei File 3D con Unity e Gazebo

Il formato **.obj** è stato testato con successo sia in Unity che in Gazebo, consentendo l'importazione corretta del modello della t-shirt in entrambi i software. Tuttavia, durante le simulazioni in Unity, l'applicazione del componente Cloth alla t-shirt ha evidenziato alcune difficoltà: il computer ha riscontrato problemi nel gestire correttamente i comandi impostati e, una volta avviata la simulazione, la t-shirt appariva deformata e non realistica, come è possibile vedere nella figura 3.1.

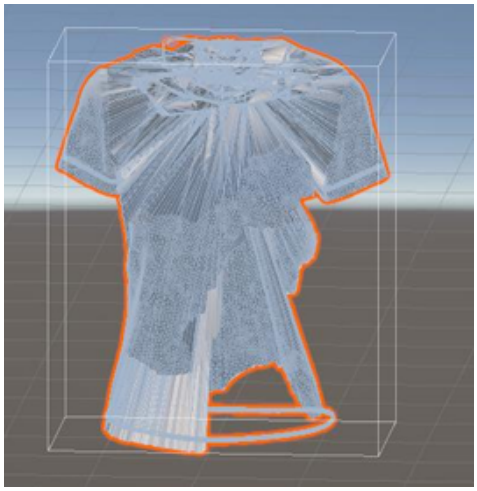

**Figure 3.1:** *Errore nell'applicazione del Cloth di Unity*

Per risolvere questo problema, l'oggetto è stato importato nel software Blender, dove è stato utilizzato il modificatore Decimate per ridurre il numero di poligoni della mesh. Diminuendo il parametro "Ratio" a un valore che ha portato la mesh a circa 5000 poligoni, è stato possibile reimportare l'oggetto in Unity e applicare correttamente il componente Cloth, ottenendo una simulazione più realistica.

Il formato **.fbx** non è supportato da Gazebo; tuttavia, è compatibile con Unity e può essere convertito in altri formati, come .obj o .dae (Collada), utilizzando Blender. Seguendo lo stesso procedimento adottato per il file .obj, quindi riducendo il numero di poligoni delle mesh tramite il modificatore Decimate in Blender, la simulazione in Unity è stata eseguita correttamente.

Il formato **.dae** (Collada) è supportato da Gazebo e risulta essere ampiamente utilizzato in questo contesto. Importando i file .obj della t-shirt e della scarpa, nonché il file .fbx della t-shirt in Blender, è stato possibile esportarli nel formato .dae. Questa operazione ha permesso

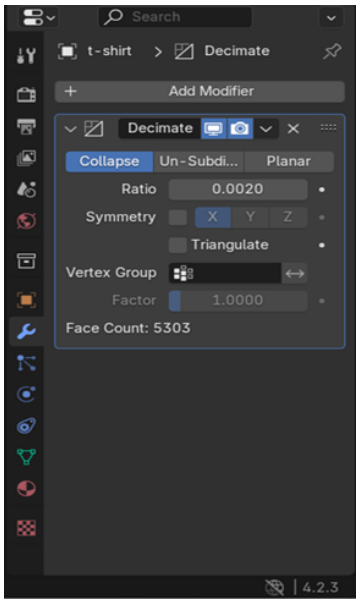


**Figure 3.2:** *Modifica della Mesh nel software Blender*

di caricare correttamente i vari modelli in Gazebo, facilitando la simulazione degli indumenti nel simulatore.

In sintesi, l'utilizzo combinato di Blender per l'ottimizzazione delle mesh e la conversione dei formati, insieme all'applicazione del componente Cloth in Unity, ha consentito di ottenere simulazioni più realistiche del comportamento dei tessuti, migliorando l'interazione tra il robot e gli indumenti nel contesto del progetto.

### 3.2.1 Componente Cloth e collisioni con altri materiali

Il Cloth di Unity oltre a realizzare una buona fisica per la maglietta è in grado di creare delle collisioni con altri materiali. In particolare, come è stato descritto nel capitolo precedente, si può inserire un oggetto Capsule o Sphere nel Cloth e quindi specificare alla maglietta quali e quanti oggetti dovranno avere un'interazione fisica opportuna durante la collisione. Tuttavia non è possibile far si che l'oggetto a cui è applicato il Cloth (in questo caso la maglietta) collida con un qualunque oggetto di qualunque forma geometrica. Infatti come anticipato è possibile utilizzare solo due tipi di oggetti 3D Unity: Capsule e Sphere. Questa restrizione rappresenta una problematica nel caso in cui il tessuto debba interagire con superfici piatte, angoli o forme geometriche più complesse, come un tavolo, una scatola o una struttura irregolare. Nel caso del progetto un'applicazione utile sarebbe quella di poter far interagire la maglietta con l'end effector del robot che poi riesca ad appoggiarla su una superficie piana come un tavolo. Poiché non è possibile definire direttamente colliders di tipo Mesh o Box per il Cloth, è necessario trovare soluzioni alternative per ottenere un comportamento realistico nelle simulazioni.
Una soluzione consiste nel posizionare una griglia di piccole Sphere Colliders sulla superficie con cui il tessuto deve interagire. In questo modo, il tessuto percepirà la presenza di una superficie continua, pur utilizzando solo colliders sferici. Per implementare questa soluzione:

- Creare uno script che generi automaticamente una matrice di sfere sulla superficie desiderata.
- Assicurarsi che le sfere vengano generate prima che il Cloth venga configurato, in modo che possano essere assegnate correttamente ai "Sphere Colliders" della maglietta.
- Ottimizzare il numero e la dimensione delle sfere per ottenere una buona resa senza sovraccaricare il motore fisico.

## 3.3 Problemi di visualizzazione delle meshes in Gazebo e Unity

# 4 Struttura del workspace ROS2

*In questo capitolo viene presentata la struttura del workspace ROS2, con particolare attenzione alla disposizione dei pacchetti e dei file necessari per l'integrazione con Unity e la simulazione del robot. Come sotto riportato, i package sono evidenziati in rosso mentre i file Python sono evidenziati in blu.*

```
Dynamics/
    |-- ros2_humble/
    |   |-- src/
    |   |   |-- moveit_config_tm5_900/
    |   |   |   |-- config/
    |   |   |   |-- launch/
    |   |   |   |   |-- demo.launch.py
    |   |   |   |   |-- move_group.launch.py
    |   |   |   |   |-- setup_assistant.launch.py
    |   |   |   |-- .setup_assistant
    |   |   |   |-- CMakeLists.txt
    |   |   |   `-- package.xml
    |   |   |-- my_ros2_package/
    |   |   |   |-- my_ros2_package/
    |   |   |   |   |-- build/
    |   |   |   |   |-- install/
    |   |   |   |   |-- log/
    |   |   |   |   |-- moveit_controller.py
    |   |   |   |   `-- pose_listener.py
    |   |   |   |-- resource/
    |   |   |   |-- test/
    |   |   |   |-- setup.cfg
    |   |   |   `-- setup.py
    |   |   |-- tm_description/
    |   |   |   |-- config/
    |   |   |   |-- launch/
    |   |   |   |-- meshes/
    |   |   |   |-- urdf/
    |   |   |   |-- CMakeLists.txt
    |   |   |   |-- package.xml
    |   |   `-- tm_moveit_config/
    |   |       |-- ...
    |   |-- install/
    |   |-- build/
    |   `-- log/
```

Nel diagramma sono state riportate solo le cartelle e i file considerati più rilevanti per la comprensione dell'architettura del workspace; la repository completa è disponibile a [9].

Si segnala, inoltre, che il package `tm_moveit_config` non è stato ulteriormente implementato, in quanto è stato creato automaticamente tramite il MoveIt Setup Assistant e presenta la stessa struttura di `moveit_config_tm5_900`, spiegata nel dettaglio alla sezione 2.2.3.

Questi due package influenzano direttamente il funzionamento del robot e sono stati creati per gestire la cinematica e la dinamica del braccio robotico. Tramite il comando

```
ros2 launch moveit_config_tm5_900 setup_assistant.launch.py
```

oppure

```
ros2 launch tm_moveit_config setup_assistant.launch.py
```

per i due package, si avvierà il Moveit Setup Assistant con il quale è possibile variare le pose del robot, le traiettorie e i giunti influenzati da esse. Il file `demo.launch.py` permette di avviare il MoveIt2 demo, che consente di testare le funzionalità del robot in un ambiente simulato (approfondimento alla sezione 2.2.3). Il file `move_group.launch.py` è il nodo principale che gestisce la comunicazione tra ROS2 e MoveIt2, mentre il file `setup_assistant.launch.py`, come già accenato, è utilizzato per configurare il robot e le sue proprietà cinematiche.
La cartella `my_ros2_package` contiene gli script Python che, una volta stabilita la connessione e avviati i controllori di Moveit, inviano i comandi da ROS2 a Unity, in particolare il file `moveit_controller.py` che si occupa di simulare la movimentazione del robot TM5-900 in Unity dalla posa di "Home" a quella di "Work" definite nel Moveit Setup Assistant. Questo file è fondamentale per l'integrazione tra ROS2 e Unity, in quanto consente di controllare il robot direttamente dall'interfaccia grafica di Unity, facilitando la simulazione e il test delle traiettorie.
La cartella `tm_description` contiene i file URDF e i modelli 3D necessari per rappresentare il robot in Gazebo e Unity. In questo package è importante che siano presenti le meshes all'interno della cartella `meshes`, sia di visualizzazione che di collisione, e il file URDF all'interno della cartella `urdf`. In particolare il file URDF è possibile generarlo a partire dal file xacro tramite il comando:

```
ros2 run xacro xacro nome_file.xacro > nome_file.urdf
```

Il file URDF è fondamentale per la rappresentazione del robot in Gazebo e Unity, in quanto definisce la struttura e le proprietà fisiche del robot, inclusi i link, i giunti e le collisioni. Inoltre, il file URDF deve essere correttamente configurato per garantire che il robot venga visualizzato e simulato correttamente in entrambi gli ambienti e quindi che i path relativi alle meshes siano corretti (problematiche riguardo ai path delle meshes nei vari software alla sezione 3.3).

# 5 Test Effettuati e Risultati Ottenuti

# 6 Conclusioni

## 6.1 Applicazione finale

# A Appendice1

Se necessario ricorrete alle appendici per spiegare le parti "di contorno" dell'attività svolta e/o ciò che non riuscite ad inserire nello schema generale dei capitoli della relazione (es acquisizione dei dati con Matlab).

# Bibliografia


[1] "Ros2." https://docs.ros.org/en/humble/index.html.

[2] "Moveit." https://moveit.picknik.ai.

[3] "Moveit installation." https://moveit.picknik.ai/humble/doc/tutorials/getting_started/getting_started.html.

[4] "Moveit setup assistant tutorial." https://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html.

[5] "Unity." https://docs.unity.com/.

[6] "Manual cloth unity." https://docs.unity3d.com/Manual/class-Cloth.html.

[7] "Compatibilità gazebo-ros2." https://gazebosim.org/docs/harmonic/ros_installation/#summary-of-compatible-ros-and-gazebo-combinations.

[8] "Urdf-importer." https://github.com/Unity-Technologies/URDF-Importer.

[9] "Repository di ros2." https://github.com/Piz01/Dynamics.git.