ACTIVITY REPORT IOT

REPORT SULLO SVILUPPO DELLE ATTIVITÀ PER IL PROGETTO

Il seguente report si pone l'obiettivo di descrivere ed elencare le principali modifiche necessarie che sono state apportate per la transizione da ROS1 a ROS2 e la successiva implementazione di un modello in grado di rappresentare un oggetto reale e la successiva simulazione su Gazebo in un ambiente con un mondo complesso.

ACTIVITY REPORT IOT

REPORT SULLO SVILUPPO DELLE ATTIVITÀ PER IL PROGETTO

Il presente report è stato redatto con l'intento di delineare le sostanziali modifiche richieste per la transizione da ROS1 a ROS2, nonché per descrivere il processo di creazione e implementazione di un modello robotico destinato a rappresentare un oggetto reale nell'ambito della simulazione su Gazebo. Questa transizione rappresenta un passo fondamentale per adattarsi alle nuove tecnologie e benefici offerti da ROS2.

L'obiettivo primario di questo report è fornire una panoramica dettagliata delle fasi coinvolte nell'evoluzione da ROS1 a ROS2, evidenziando le modifiche essenziali apportate per garantire la compatibilità e il corretto funzionamento all'interno dell'ecosistema di ROS2.

Inoltre, si procederà a illustrare il processo di sviluppo e implementazione di un modello robotico capace di rappresentare in modo accurato un oggetto reale, non trascurando la complessità di un ambiente simulato in Gazebo. Tale simulazione, all'interno di un contesto ambientale sfaccettato e articolato, mira a testare e validare le funzionalità del modello, fornendo un quadro esaustivo delle prestazioni del robot in scenari realistici.

Attraverso l'analisi dettagliata di queste fasi di transizione tecnologica e sviluppo di modelli robotici per la simulazione su Gazebo, ci proponiamo di evidenziare le sfide incontrate, le soluzioni implementate e le opportunità emerse durante questo processo di transizione e sviluppo.

PRINCIPALI DIFFERENZE ROS1 E ROS2

La transizione da ROS (Robot Operating System) 1 a ROS 2 è stata guidata da una serie di miglioramenti e necessità che hanno portato all'adozione di ROS 2 rispetto alla versione precedente.

- Architettura migliorata e flessibile: ROS 2 ha una struttura più modulare e flessibile rispetto a ROS 1. Questo consente una maggiore scalabilità e adattabilità ai requisiti dei sistemi robotici moderni, inclusi quelli distribuiti e più complessi.

- Supporto per la comunicazione multipla: ROS 2 offre supporto nativo per diversi protocolli di comunicazione, come DDS (Data Distribution Service). Questo migliora l'interoperabilità tra i nodi di sistema e consente la comunicazione tra robot e dispositivi eterogenei, il che può essere cruciale in scenari complessi.

- Affidabilità e sicurezza migliorata: Uno dei principali punti di forza di ROS 2 è il focus sulla sicurezza e l'affidabilità. Introduce funzionalità come la gestione degli errori migliorata, consentendo ai sistemi robotici di gestire meglio situazioni critiche e garantire un funzionamento più affidabile.

- Supporto per sistemi in tempo reale: ROS 2 ha una maggiore compatibilità con i requisiti dei sistemi in tempo reale. Questo è fondamentale in applicazioni che richiedono tempi di risposta più rapidi e una maggiore precisione temporale.

- Maggiore community e supporto: Con l'evoluzione di ROS 2, la community si è espansa, offrendo un maggiore supporto, risorse e contributi. Ciò significa che ci sono più strumenti, librerie e conoscenze disponibili per aiutare nello sviluppo e nell'implementazione dei progetti.

I FILE URDF

I file URDF (Unified Robot Description Format) sono file di testo XML utilizzati all'interno del framework di Robot Operating System (ROS) per descrivere in modo dettagliato la struttura e la cinematica dei robot.

Scopo dei file URDF:

- Descrizione della struttura del robot: I file URDF contengono informazioni riguardanti la geometria, le dimensioni, i collegamenti fisici e la cinematica del robot. Questi dettagli consentono di rappresentare accuratamente la struttura fisica del robot, inclusi bracci, giunti, sensori e altre componenti.

- Configurazione della cinematica: Definiscono i giunti, i loro tipi e limiti di movimento, insieme alle relazioni tra le diverse parti del robot. Queste informazioni sono essenziali per modellare e simulare il movimento e il comportamento del robot.

- Rappresentazione visiva: I file URDF possono anche includere informazioni visive come texture, colori, trasparenze e materiali per fornire una rappresentazione visiva accurata del robot durante la simulazione.

- Integrazione in ROS: Essendo parte integrante di ROS, i file URDF sono essenziali per integrare il modello del robot all'interno dell'ecosistema ROS. Questo permette di utilizzare il modello durante lo sviluppo, il controllo, la simulazione e altre attività all'interno di ROS.

Utilizzo pratico:

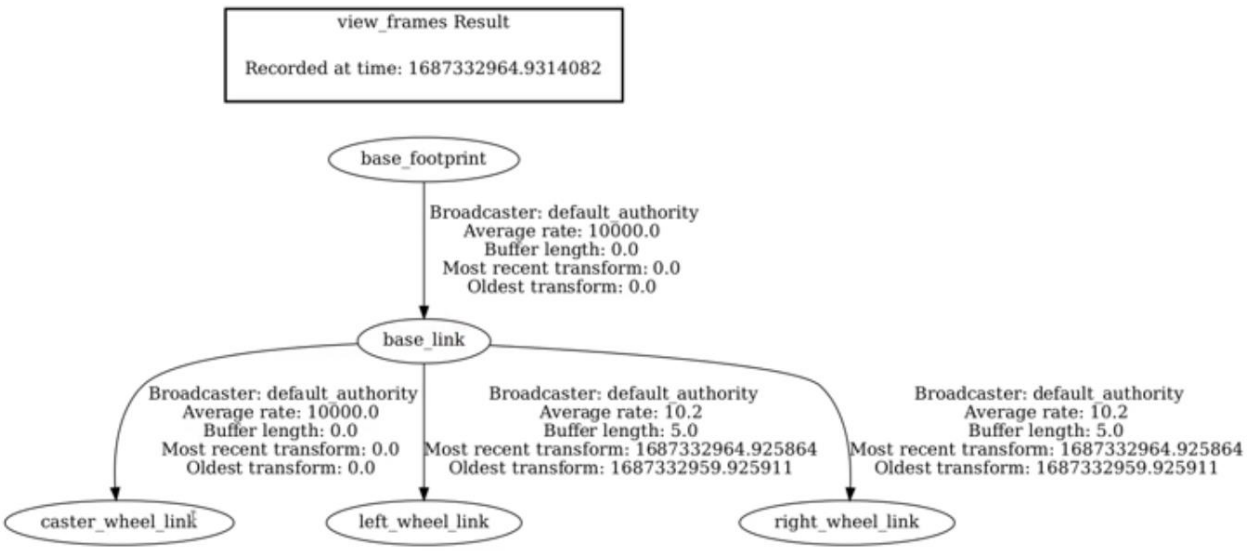
Simulazione: I file URDF sono utilizzati dai simulatori, come Gazebo, per simulare il comportamento e il movimento del robot. Consentono di visualizzare il robot, di eseguire test di controllo e di valutare il suo comportamento in diversi scenari.

Controllo e navigazione: Durante lo sviluppo dei controlli del robot e dei sistemi di navigazione, i file URDF sono utilizzati come base per rappresentare la struttura e la cinematica del robot, consentendo ai programmatori di lavorare con un modello accurato del robot.

Visualizzazione e debug: RViz, lo strumento di visualizzazione di ROS, utilizza i file URDF per visualizzare il modello del robot in un ambiente virtuale, aiutando gli sviluppatori a comprendere meglio la disposizione delle componenti e a diagnosticare eventuali problemi.

In sintesi, i file URDF sono essenziali in ROS perché forniscono una rappresentazione dettagliata della struttura e della cinematica del robot, permettendo di simulare, controllare, visualizzare e sviluppare il comportamento del robot all'interno dell'ecosistema ROS.

```
<link name="base_footprint" />

<link name="base_link">
    <visual>
        <geometry>
            <box size="0.6 0.4 0.2" />
        </geometry>
        <origin xyz="0 0 0.1" rpy="0 0 0" />
        <material name="blue" />
    </visual>
</link>

<link name="right_wheel_link">
    <visual>
        <geometry>
            <cylinder radius="0.1" length="0.05" />
        </geometry>
        <origin xyz="0 0 0" rpy="1.57 0 0" />
        <material name="grey" />
    </visual>
</link>

<link name="left_wheel_link">
    <visual>
        <geometry>
            <cylinder radius="0.1" length="0.05" />
        </geometry>
        <origin xyz="0 0 0" rpy="1.57 0 0" />
        <material name="grey" />
    </visual>
</link>
```



RVIZ

RViz è uno strumento di visualizzazione 3D molto potente all'interno del framework di Robot Operating System (ROS). È progettato per visualizzare dati sensoriali, modelli di robot, informazioni sulla posizione, mappe e altro ancora, offrendo una visualizzazione interattiva e in tempo reale degli elementi presenti nell'ambiente robotico.

Principali caratteristiche di RViz:

- Visualizzazione di dati: RViz permette di visualizzare una vasta gamma di dati provenienti dai sensori del robot, come dati da telecamere, laser scanner, sonar, sensori di profondità e molto altro ancora. Questi dati vengono visualizzati in tempo reale.

- Modelli robotici: RViz supporta la visualizzazione dei modelli robotici descritti nei file URDF. Ciò consente di visualizzare il robot e le sue parti, controllare il movimento, esplorare le relazioni tra le parti del robot e personalizzare l'aspetto visivo.

- Strumenti interattivi: RViz offre strumenti interattivi per muovere la visuale, ruotare, zoomare e navigare all'interno dell'ambiente 3D. Ciò consente di esplorare facilmente il modello del robot e gli oggetti circostanti.

- Marker e rappresentazioni: RViz permette di visualizzare i "marker", che sono elementi visivi aggiuntivi utilizzati per indicare punti, traiettorie, zone di interesse o altri dati specifici del robot. Inoltre, offre diverse rappresentazioni visive per i dati, come punti, linee, griglie e molto altro ancora.

- Debugging e sviluppo: RViz è uno strumento fondamentale per il debugging durante lo sviluppo del software di controllo del robot. Aiuta a identificare errori nella percezione, nel movimento o nelle interazioni con l'ambiente.

- Integrazione con ROS: RViz è completamente integrato con ROS, il che significa che può ricevere dati direttamente da nodi e topic all'interno di un sistema ROS, permettendo una visualizzazione e un'interazione dirette con i dati provenienti dai sensori e dal modello del robot.

In definitiva, RViz è una componente fondamentale di ROS che fornisce uno strumento di visualizzazione essenziale per esplorare, analizzare e debuggare i dati provenienti dai sensori e i modelli dei robot all'interno dell'ambiente di sviluppo ROS.

Utilità di creare un file URDF con RViz

Creare un file URDF (Unified Robot Description Format) con l'aiuto di RViz offre diversi vantaggi significativi nell'ambito della robotica e della simulazione.

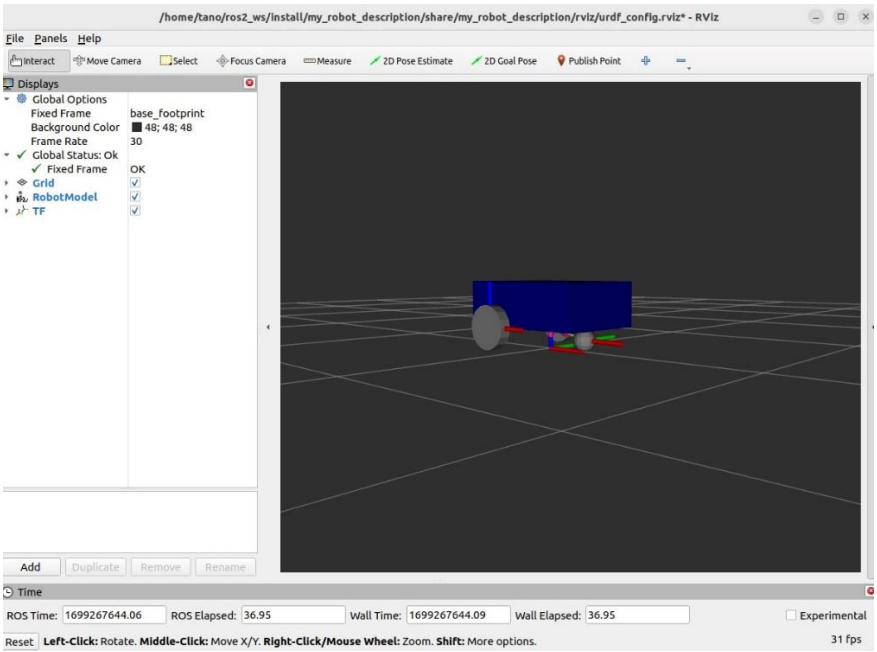
Rappresentazione visiva del robot: RViz consente di visualizzare il robot in un ambiente virtuale. Utilizzando un file URDF, è possibile rappresentare graficamente il robot, includendo informazioni dettagliate sulla sua struttura, sensori, giunti e collegamenti.

Simulazione e analisi: Grazie alla visualizzazione 3D, è possibile simulare il comportamento del robot e valutarne le prestazioni. Questa simulazione può essere utile per comprendere come il robot si comporterà nella realtà, testare algoritmi di controllo e verificare la correttezza della progettazione.

Configurazione dei sensori: RViz consente di simulare il funzionamento dei sensori montati sul robot. È possibile visualizzare i dati dei sensori e valutare il loro impatto sul comportamento complessivo del robot.

Debugging e sviluppo: RViz fornisce strumenti per il debug durante lo sviluppo del software di controllo. È possibile visualizzare in tempo reale i cambiamenti nella configurazione del robot, facilitando l'individuazione e la risoluzione di errori.

In definitiva, l'utilizzo di RViz per creare e visualizzare un file URDF offre un ambiente flessibile e interattivo per la progettazione, la simulazione e lo sviluppo dei robot, consentendo agli sviluppatori di comprendere e migliorare le prestazioni del robot in modo efficiente.

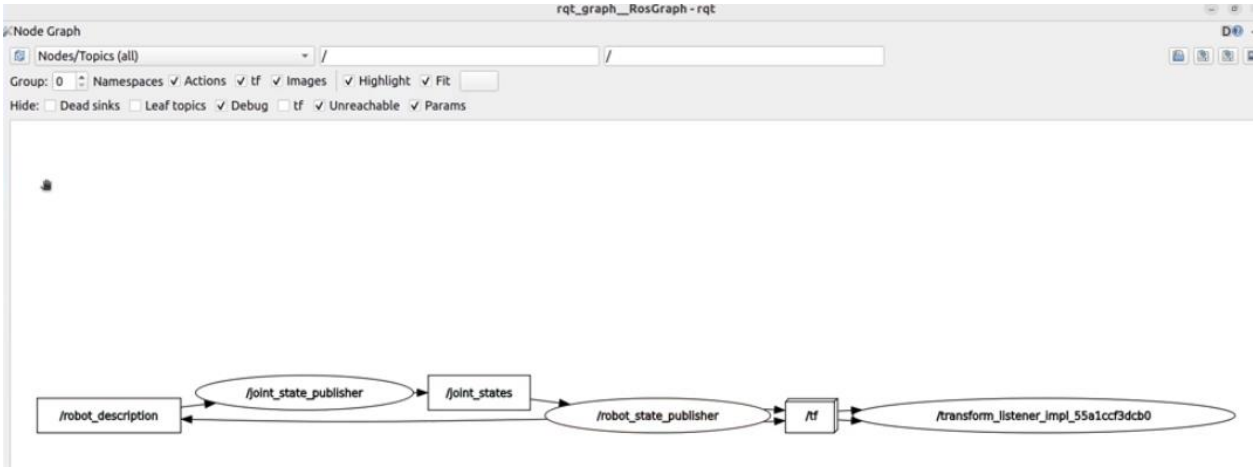


RQT GRAPH

Il RQT Graph è uno strumento estremamente utile all'interno di ROS (Robot Operating System) e viene spesso utilizzato insieme a RViz, il visualizzatore 3D di ROS. RQT Graph fornisce una rappresentazione visiva dei nodi attivi all'interno di un sistema ROS e delle connessioni tra di essi.

Con RQT Graph puoi visualizzare i nodi attivi, i topic dei messaggi che vengono pubblicati e sottoscritti, e le connessioni tra i vari nodi. Questo ti consente di avere una visione d'insieme della comunicazione tra i diversi componenti di un sistema ROS in tempo reale, aiutandoti a comprendere e debuggare il comportamento dei nodi.

È particolarmente utile durante lo sviluppo e il debug di robot o sistemi robotici, in quanto offre una panoramica chiara e dinamica delle interazioni tra i nodi ROS, consentendo agli sviluppatori di individuare facilmente eventuali problemi di comunicazione o flussi di dati indesiderati.



LE TF ED I MATERIAL NEGLI URDF

Le TF (Transformations) in ROS (Robot Operating System) sono un sistema fondamentale per gestire le trasformazioni di coordinate tra diversi frame di riferimento all'interno di un sistema robotico.

Concetto delle TF:

Le TF gestiscono le relazioni spaziali tra diversi componenti di un sistema robotico, consentendo di sapere la posizione e l'orientamento di un oggetto rispetto a un altro all'interno dello spazio tridimensionale. Questi frame di riferimento possono rappresentare parti del robot, sensori o altri elementi.

Utilità delle TF nei file URDF:

Definizione delle relazioni spaziali: Nei file URDF, le TF vengono utilizzate per definire le relazioni spaziali tra le varie parti del robot. Ad esempio, è possibile definire la posizione e l'orientamento di un sensore rispetto al telaio del robot o la posizione di una giuntura rispetto a un'altra.

Coordinate trasformate: Le trasformazioni definite dalle TF sono essenziali per convertire le coordinate da un frame di riferimento a un altro. Ciò è fondamentale quando si lavora con più sensori o parti del robot che hanno i loro sistemi di coordinate locali.

Sistema di supporto per il controllo e la navigazione: Le informazioni spaziali fornite dalle TF sono utilizzate nei sistemi di controllo e navigazione del robot. Ad esempio, nel controllo del movimento del braccio robotico o nella navigazione di un robot mobile, conoscere la posizione relativa delle diverse parti del robot è cruciale per il funzionamento corretto del sistema.

RViz e visualizzazione: RViz, in combinazione con le TF, permette di visualizzare in modo intuitivo le relazioni spaziali tra le diverse parti del robot. Ciò consente agli sviluppatori di comprendere meglio la configurazione e la disposizione del robot durante la progettazione e la simulazione.

In breve, le TF sono fondamentali nei file URDF perché forniscono le informazioni necessarie sulle relazioni spaziali tra i componenti del robot, consentendo di gestire le coordinate e le

trasformazioni nello spazio tridimensionale, nonché di supportare il controllo, la navigazione e la visualizzazione nel contesto di ROS.

Nei file URDF (Unified Robot Description Format), i tag principali sono utilizzati per definire i vari componenti e le caratteristiche del robot. Includono informazioni sulla geometria, la cinematica, i sensori e anche la resa visiva tramite i tag Material.

Principali tag in un file URDF:

<link>: Definisce un collegamento fisico del robot. Questo tag contiene informazioni sulla geometria del collegamento, come mesh, forme geometriche o file CAD, specificando anche le trasformazioni e le proprietà fisiche.

<joint>: Descrive un giunto che collega due collegamenti. Può rappresentare giunti di vario tipo come fissi, rotazionali o lineari, e specifica i limiti, le trasformazioni e altre proprietà relative al movimento.

<sensor>: Questo tag viene utilizzato per definire i sensori montati sul robot. Può includere sensori come fotocamere, lidar, accelerometri, etc., insieme alle relative caratteristiche e parametri.

<material>: Questo è uno dei tag chiave per la rappresentazione visiva del robot. Il tag Material definisce le proprietà visive come colore, riflessività, texture, etc. È cruciale per la resa grafica del modello del robot in RViz o simulatori, consentendo di dare al robot un aspetto realistico.

Importanza dei tag Material:

I tag Material sono estremamente importanti per definire l'aspetto visivo del robot nei simulatori o negli ambienti di visualizzazione come RViz. Essi permettono di:

Personalizzare l'aspetto visivo: Il tag Material consente di definire il colore, la riflessività e altre caratteristiche visive per ciascuna parte del robot, permettendo di distinguere le diverse componenti e rendere più realistica la rappresentazione.

Migliorare l'esperienza visiva: Una rappresentazione accurata e realistica del robot nei simulatori può facilitare lo sviluppo e il debug del software di controllo, permettendo agli sviluppatori di comprendere meglio il comportamento del robot nel contesto della simulazione.

Dettagliare il prototipo: Per presentare un prototipo realistico, i tag Material possono essere utilizzati per assegnare texture o riflessività alle superfici del modello, rendendolo più fedele al prototipo fisico e facilitando la comprensione delle caratteristiche materiali del robot.

In conclusione, i tag Material nei file URDF sono cruciali per la resa visiva del robot, consentendo di personalizzare l'aspetto del modello e migliorare l'esperienza visiva nei simulatori e negli strumenti di visualizzazione, essenziali per lo sviluppo e la comprensione del comportamento del robot.

Online esistono librerie e database di materiali che offrono una forma di standardizzazione per la ricerca e l'uso di materiali all'interno dei file URDF. Questi database sono estremamente utili per gli sviluppatori che creano modelli robotici in quanto semplificano l'assegnazione di proprietà visive e fisiche ai componenti del robot.

```
<material name="blue">
    <color rgba="0 0 0.5 1" />
</material>

<material name="grey">
    <color rgba="0.5 0.5 0.5 1" />
</material>
```

Ruolo delle librerie di materiali per i file URDF:

Standardizzazione dei materiali: Le librerie online di materiali forniscono una vasta gamma di opzioni predefinite per materiali comuni come plastica, metallo, vetro, tessuti, ecc. Questi materiali predefiniti possono essere facilmente utilizzati per dare un aspetto realistico ai modelli robotici.

Facilità di accesso e utilizzo: Queste librerie sono accessibili attraverso piattaforme online e offrono una vasta gamma di materiali con relative proprietà visive e fisiche già definite. Gli

sviluppatori possono accedere a queste risorse per assegnare rapidamente proprietà visive ai componenti del robot, riducendo così il tempo necessario per la configurazione dei modelli.

Consistenza e coerenza: Utilizzando materiali predefiniti da queste librerie, si ottiene una maggiore coerenza visiva e una migliore rappresentazione dei materiali reali. Ciò consente una maggiore uniformità e coerenza nell'aspetto dei modelli, semplificando la visualizzazione e il confronto tra differenti modelli di robot.

Aggiornamenti e ampliamenti: Queste librerie vengono continuamente aggiornate e arricchite con nuovi materiali in base alle esigenze e alle nuove scoperte nel campo della robotica. Ciò garantisce agli sviluppatori un accesso continuo a nuove opzioni e a materiali sempre più realistici.

XACRO

Xacro è un linguaggio di markup utilizzato in ROS (Robot Operating System) per semplificare e rendere più flessibile la creazione di file URDF (Unified Robot Description Format). Si basa su XML e consente di generare modelli robotici complessi in modo modulare e parametrizzato.

Principali caratteristiche di Xacro:

Modularità: Xacro permette la creazione di modelli robotici suddivisi in parti più piccole e riutilizzabili. Queste parti possono essere definite separatamente e poi integrate all'interno di un unico file, facilitando la gestione e la manutenzione del codice.

Parametrizzazione: È possibile definire parametri all'interno dei file Xacro, rendendo i modelli robotici più flessibili e personalizzabili. Questo consente di adattare facilmente le proprietà dei componenti del robot, come dimensioni, colori, e altre caratteristiche, senza dover modificare direttamente il codice.

Include file esterni: Xacro supporta l'importazione di file esterni, permettendo di dividere il modello del robot in componenti separati e riutilizzabili. Ciò favorisce una migliore organizzazione del codice e una più facile gestione di parti ripetitive o comuni tra diversi modelli.

Condizioni e logica di programmazione: Xacro permette l'uso di condizioni e logica di programmazione all'interno dei file, consentendo di definire comportamenti o caratteristiche del robot in base a determinate condizioni o parametri.

Integrazione con URDF: I file Xacro vengono compilati in file URDF prima dell'utilizzo effettivo nel sistema ROS. Questo processo di compilazione trasforma il codice Xacro in un file URDF standard, che può essere facilmente utilizzato all'interno di ROS.

Utilità di Xacro in ROS:

Semplificazione della creazione dei modelli: Xacro rende più efficiente e modulare la creazione di modelli robotici complessi, permettendo agli sviluppatori di dividere il codice in parti più gestibili e riutilizzabili.

Parametrizzazione e flessibilità: L'uso di parametri e la modularità consentono una maggiore flessibilità nella definizione delle caratteristiche del robot, agevolando l'adattamento del modello a diverse configurazioni o esigenze specifiche.

Manutenzione e gestione semplificate: La modularità dei file Xacro semplifica la manutenzione e la gestione dei modelli robotici, consentendo una migliore organizzazione del codice e facilitando la risoluzione di eventuali problemi o aggiunte.

In definitiva, Xacro è utilizzato in ROS per semplificare la creazione e la gestione dei modelli robotici, rendendo più modulare, flessibile e parametrizzato il processo di definizione dei componenti del robot all'interno dei file URDF.

I costrutti principali di Xacro sono strumenti fondamentali utilizzati per creare modelli robotici complessi e parametrizzati in ROS. Eccoli qui:

1. **<xacro:macro>**

Il tag <xacro:macro> consente di definire un blocco di codice riutilizzabile. È simile a una funzione in programmazione e accetta parametri che possono essere utilizzati all'interno del macro per generare parti del modello.

2. **<xacro:property>**

Il tag <xacro:property> permette di definire una proprietà che può essere utilizzata in tutto il file Xacro. Questo consente di definire parametri globali per il modello robotico.

3. **<xacro:include>**

Il tag <xacro:include> permette di importare e integrare altri file Xacro all'interno del file corrente. Questo agevola la modularità e la riutilizzabilità del codice.

4. Variabili e sostituzione di testo

Xacro supporta l'uso di variabili, consentendo di sostituire testo o valori all'interno dei file. Questo è particolarmente utile per parametrizzare il modello del robot.

Le macro in Xacro svolgono un ruolo chiave nella creazione di modelli robotici complessi all'interno di ROS. Ecco perché sono importanti:

5. Riutilizzo del codice:

Le macro permettono di definire blocchi di codice riutilizzabili. Questi blocchi possono essere chiamati più volte in diversi punti del modello, riducendo la duplicazione del codice e facilitando la manutenzione. Invece di ripetere le stesse porzioni di codice per definire parti simili del robot, è possibile definire una macro una volta e chiamarla ogni volta che è necessario.

6. Modularità e leggibilità:

Utilizzando le macro, è possibile suddividere il modello del robot in parti più piccole e gestibili. Questo migliora la leggibilità del codice, facilita la comprensione della struttura del robot e semplifica la gestione di parti ripetitive o comuni tra diversi modelli.

7. Parametrizzazione:

Le macro possono accettare parametri, permettendo la creazione di blocchi di codice flessibili e parametrizzati. Questo consente di generare parti del modello che possono essere personalizzate attraverso l'immissione di parametri, come dimensioni, colori o altre caratteristiche, facilitando l'adattamento del modello a varie configurazioni o esigenze specifiche.

8. Aggiornamenti e manutenzione semplificati:

Quando si utilizzano macro, eventuali modifiche o aggiornamenti alle parti comuni del modello possono essere apportati modificando la definizione della macro stessa. Questo riduce il rischio di errori dovuti a modifiche multiple in parti simili del codice e semplifica la manutenzione del modello.

9. Consistenza e coerenza:

L'uso delle macro aiuta a mantenere una maggiore coerenza e uniformità nell'aspetto e nella struttura del modello robotico. Poiché le parti comuni del robot sono definite in un'unica macro, tutti i riferimenti a quella macro genereranno parti identiche, garantendo una maggiore coerenza nel modello.

```
<?xml version="1.0"?>
<robot name="my_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">

    <xacro:include filename="common_properties.xacro" />
    <xacro:include filename="mobile_base.xacro" />

</robot>
```

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

    <xacro:property name="base_length" value="0.6" />
    <xacro:property name="base_width" value="0.4" />
    <xacro:property name="base_height" value="0.2" />
    <xacro:property name="wheel_radius" value="0.1" />
    <xacro:property name="wheel_length" value="0.05" />

    <link name="base_footprint" />

    <link name="base_link">
        <visual>
            <geometry>
                <box size="${base_length} ${base_width} ${base_height}" />
            </geometry>
            <origin xyz="0 0 ${base_height / 2.0}" rpy="0 0 0" />
            <material name="blue" />
        </visual>
    </link>
```

AGGIUNTA CAMERA AL ROBOT

Nel contesto della simulazione robotica in Gazebo ed in particolar modo per le esigenze richieste dalla guida autonoma, l'implementazione di una videocamera nel file URDF presenta una serie di motivazioni fondamentali che ci hanno spinto a realizzarla:

Validazione dei sensori: Integrare la videocamera nel file URDF consente di simulare in tempo reale la vista prospettica del robot. Questa simulazione permette la validazione e il test delle

funzionalità della telecamera in un ambiente virtuale, replicando le condizioni reali di acquisizione delle immagini.

Sviluppo e debug: La simulazione della videocamera all'interno di Gazebo consente agli sviluppatori di testare e perfezionare algoritmi di visione artificiale e di elaborazione delle immagini. Ciò riduce la dipendenza dall'hardware fisico e accelera il processo di sviluppo e ottimizzazione dell'algoritmo.

Interazione con l'ambiente virtuale: La presenza della videocamera permette al robot di interagire in modo più sofisticato con l'ambiente simulato. Le immagini catturate possono essere utilizzate per prendere decisioni in tempo reale, permettendo al robot di reagire dinamicamente agli stimoli visivi.

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

    <xacro:property name="camera_length" value="0.00" />
    <xacro:property name="camera_width" value="0.0" />
    <xacro:property name="camera_height" value="0.05" />

    <link name="camera_link">
        <visual>
            <geometry>
                <box size="${camera_length} ${camera_width} ${camera_height}" />
            </geometry>
            <material name="red" />
        </visual>
        <collision>
            <geometry>
                <box size="${camera_length} ${camera_width} ${camera_height}" />
            </geometry>
        </collision>
        <xacro:box_inertia m="0.1" l="${camera_length}" w="${camera_width}" h="${camera_height}" xyz="0 0 0" rpy="0 0 0" />
    </link>

    <joint name="base_camera_joint" type="fixed">
        <parent link="base_link" />
        <child link="camera_link" />
        <origin xyz="${(base_length + camera_length) / 2} 0 ${base_height / 2}" rpy="0 0 0" />
    </joint>

    <gazebo reference="camera_link">
        <material>Gazebo/Red</material>
        <sensor name="camera_sensor" type="camera">
            <pose>0 0 0 0 0 0</pose> <!--Position and orientation-->
            <visualize>true</visualize>
            <update_rate>30.0</update_rate>
            <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
                <frame_name>camera_link</frame_name>
            </plugin>
        </sensor>
    </gazebo>

</robot>
```

SIMULAZIONE: GAZEBO

In ROS 2, Gazebo è un simulatore di robot open-source ampiamente utilizzato per testare algoritmi, sviluppare codice e simulare ambienti complessi. È una piattaforma versatile e potente che consente agli sviluppatori di testare e validare i loro algoritmi di controllo e comportamento dei robot in un ambiente virtuale prima di implementarli su hardware fisico.

Gazebo offre un ambiente di simulazione 3D altamente realistico in cui è possibile modellare robot, sensori e ambienti complessi, consentendo agli sviluppatori di testare scenari realistici e complessi poiché è anche in grado di gestire simultaneamente più robot all'interno della stessa simulazione, consentendo lo studio delle interazioni tra più entità.

Essendo fortemente integrato con ROS 2, si ha un facile scambio di dati tra simulazione e codice reale, facilitando lo sviluppo e il debug dei sistemi robotici, anche grazie alla compatibilità con diverse piattaforme hardware, consentendo agli sviluppatori di testare e validare le proprie applicazioni su un'ampia gamma di dispositivi.

Gazebo, insieme a ROS 2, svolge un ruolo fondamentale nel ciclo di sviluppo dei robot, consentendo agli sviluppatori di ridurre il tempo e i costi necessari per lo sviluppo e il testing di algoritmi e applicazioni robotiche complesse.

Trasposizione del modello

Per le ragioni scritte in precedenza il modello sviluppato è stato trasposto in Gazebo per permettere la simulazione e il testing nel simulatore. Questo coinvolge l'adattamento del modello URDF/Xacro al formato compatibile con Gazebo, inclusi i dettagli come le interfacce dei sensori, l'interazione con il mondo virtuale e altri aspetti necessari per la simulazione.

Prima di eseguire la simulazione, però è stato necessario studiare l'ambiente in cui il robot opererà ed in modo da permettere una corretta validazione del modello in un ambiente simulato, fornendo informazioni essenziali sulle prestazioni del robot e consentendo la correzione di eventuali difetti nel modello o nelle strategie di controllo..

Il modello inizialmente testato è semplificato, esso include due ruote motrici e una terza chiamata "caster wheel" che aiuta a mantenere l'equilibrio. Questo modello semplificato può essere un passo iniziale importante per comprendere e testare le caratteristiche relative all'equilibrio prima di implementare un modello più complesso.

Il modello semplificato fornisce un ambiente di test per comprendere le dinamiche di equilibrio del robot. Questo è fondamentale per identificare e affrontare le sfide legate al mantenimento dell'equilibrio, prima di passare a un modello più complesso.

Per poter permettere il movimento del Robot all’interno di Gazebo, è stato necessario inserire all’interno del file URDF un apposito plugin che ben si adattava al tipo di movimento che desideravamo ottenere.

Il plugin "diff_drive" è progettato per simulare il movimento unidirezionale di un robot, tipicamente con una configurazione a trazione differenziale, come ad esempio molti robot a ruote. Questo plugin si interfaccia con il modello del robot nel file URDF e consente di controllare la cinetica del movimento, applicando comandi di velocità alle ruote o ai motori del robot all'interno della simulazione.

Le principali funzionalità offerte da questo plugin includono:

Controllo della velocità: Il plugin "diff_drive" permette di controllare la velocità lineare e angolare del robot, consentendo di simulare il suo movimento in avanti, indietro e di rotazione.

Interfaccia con il modello del robot: Questo plugin si adatta alle caratteristiche specifiche del robot descritte nel file URDF, permettendo di applicare correttamente i comandi di movimento alle parti mobili del modello.

Semplicità di implementazione: L'integrazione del plugin nel file URDF è relativamente semplice, rendendo più agevole la configurazione del movimento del robot all'interno della simulazione.

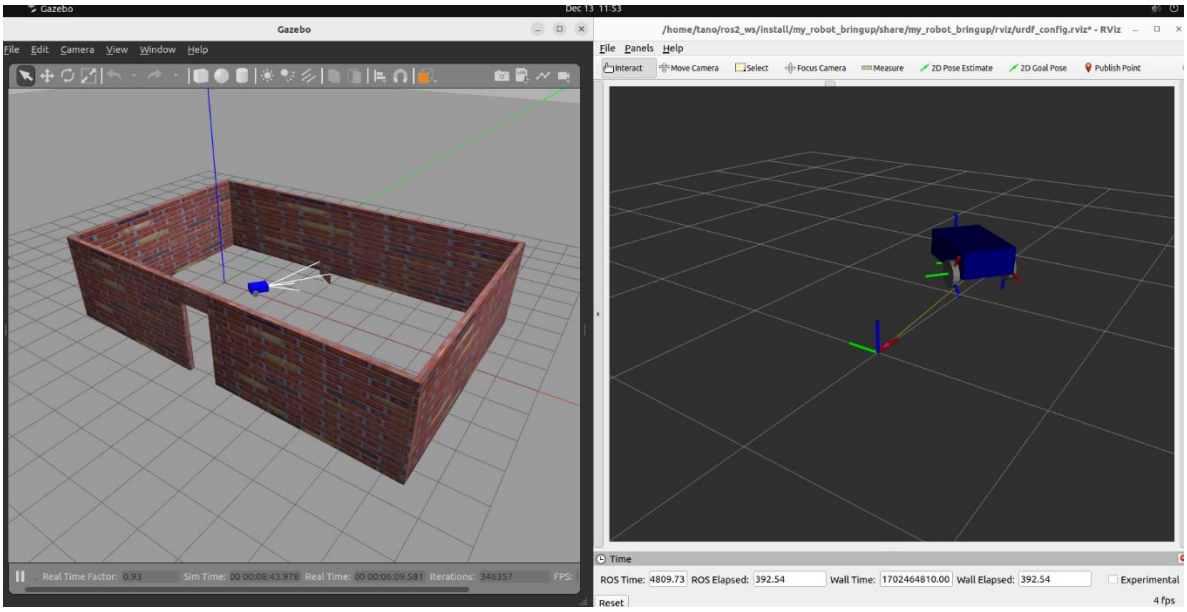
Simulazione realistica: Utilizzando il plugin "diff_drive", è possibile ottenere una simulazione più realistica del movimento del robot, tenendo conto della fisica delle ruote e della cinematica del robot stesso.

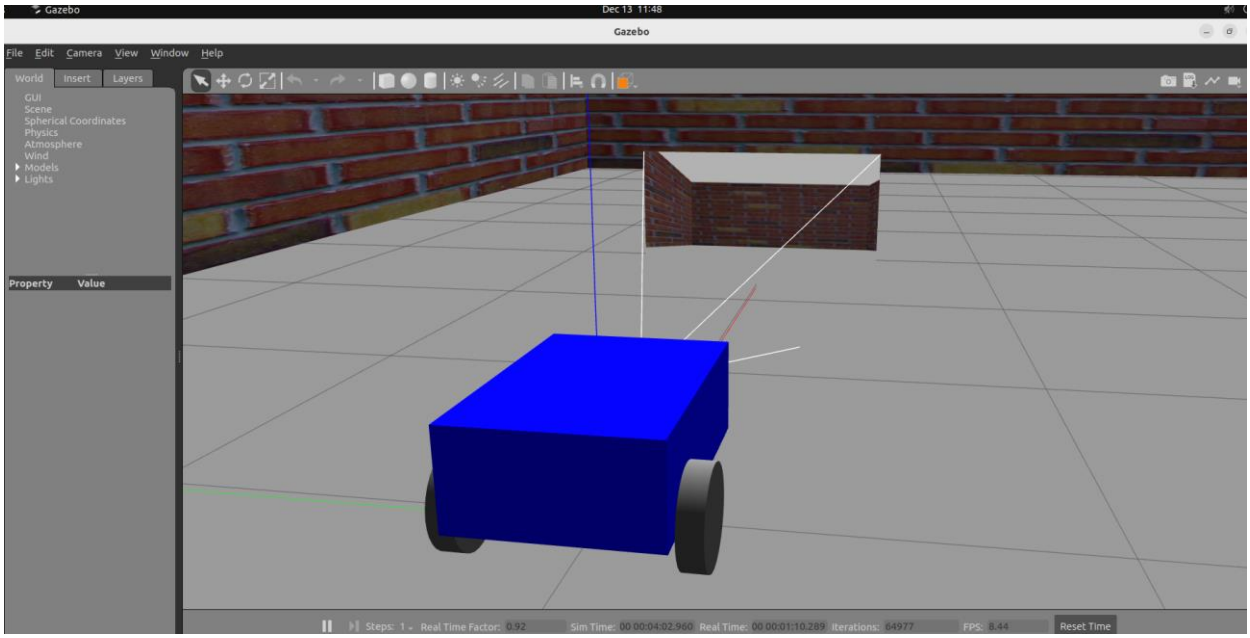
Lista finale dei topic dopo le modifiche:

```
tano@tano-ubuntu:~/ros2_ws$ ros2 topic list
/camera_sensor/camera_info
/camera_sensor/image_raw
/clicked_point
/clock
/cmd_vel
/goal_pose
/initialpose
/odom
/parameter_events
/performance_metrics
/robot_description
/rosout
/tf
/tf_static
```

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  """
    <gazebo reference="base_link">
        <material>Gazebo/Blue</material>
    </gazebo>

    <xacro:macro name="wheel_gazebo" params="prefix">
        <gazebo reference="${prefix}_wheel_link">
            <material>Gazebo/Grey</material>
        </gazebo>
    </xacro:macro>

    <xacro:macro name="caster_wheel_gazebo" params="prefix">
        <gazebo reference="${prefix}_wheel_link">
            <material>Gazebo/Grey</material>
            <mu1 value="0.1" />
            <mu2 value="0.1" />
        </gazebo>
    </xacro:macro>

    <xacro:wheel_gazebo prefix="left" />
    <xacro:wheel_gazebo prefix="right" />
    <xacro:caster_wheel_gazebo prefix="caster" />

    <gazebo>
        <plugin name="diff_drive_control" filename="libgazebo_ros_diff_drive.so">
            <!--Velocity is in m/s and steering is in rad/s-->

            <!-- Update rate in Hz -->
            <update_rate>50</update_rate>

            <!-- wheels -->
            <left_joint>base_left_wheel_joint</left_joint>
            <right_joint>base_right_wheel_joint</right_joint>

            <!-- kinematics -->
            <wheel_separation>0.45</wheel_separation>
            <wheel_diameter>0.2</wheel_diameter>

            <!-- output -->
            <publish_odom>true</publish_odom>
            <publish_odom_tf>true</publish_odom_tf>
            <publish_wheel_tf>true</publish_wheel_tf>

            <odometry_topic>odom</odometry_topic>
            <odometry_frame>odom</odometry_frame>
            <robot_base_frame>base_footprint</robot_base_frame>

          </plugin>
```

Durante la simulazione su Gazebo è stato inoltre possibile visualizzare in tempo reale l'odometria su Rviz.





MODELLO FINALE: IL MODELLO ACKERMANN

Una volta comprese le dinamiche di base e verificato il funzionamento del sistema con il modello semplificato, possiamo procedere all'implementazione del modello più complesso. Questo modello possiede tutte le componenti necessarie per rappresentare accuratamente la macchina, includendo la gestione avanzata delle ruote, la dinamica di equilibrio e altri dettagli cruciali. Il modello completo sarà in grado di rappresentare con maggiore precisione la macchina, integrando tutti i dettagli necessari per una simulazione più realistica.

Per riuscire a cambiare direzione in modo continuo e senza trascinamento, le ruote anteriori di un veicolo, quando percorrono una curva, devono essere sterzate con angoli leggermente diversi.

La ruota interna, infatti, percorre una circonferenza di raggio più stretto e, se si vuole che non si verifichino strisciamenti nel contatto ruota - terreno, deve essere sterzata con un angolo maggiore rispetto al corpo dell'auto (basta pensare al modo in cui gli atleti che partecipano ad una gara di corsa sui 400 metri sono scaglionati al momento della partenza per rendersi conto di questa necessità). Questa differenza diventa ancora più grande al restringersi della curva.

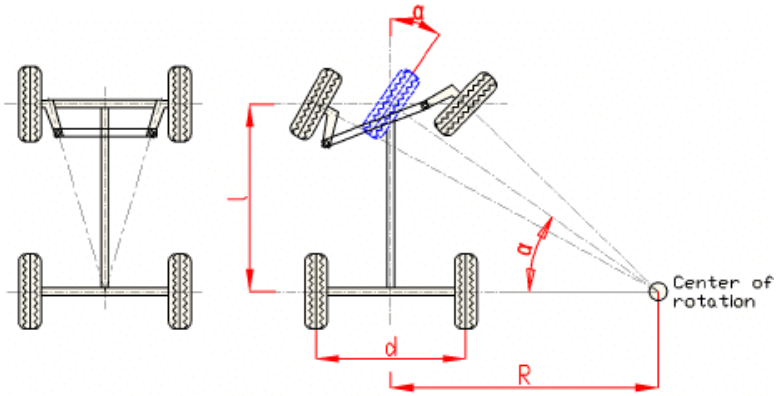
I progettisti delle auto riescono ad ottenere questo risultato predisponendo la geometria dello sterzo in base al sistema a quadrilatero articolato denominato quadrilatero di Ackermann, dal nome del tecnico tedesco Rudolf Ackermann; il sistema è anche conosciuto con il nome di quadrilatero di Jeantaud, dal nome del tecnico francese Charles Jeantaud che nel 1878 ne realizzò una prima versione.

Il principio di Ackermann afferma che, quando un veicolo effettua una sterzata, le linee immaginarie ad angolo retto rispetto al piano di mezzeria delle ruote devono incontrarsi in uno stesso punto, il cosiddetto punto di istantanea di rotazione che è il punto attorno al quale la macchina sta effettivamente girando.

Per permettere alle ruote anteriori di spostarsi in base al principio di Ackermann, i bracci dello sterzo delle ruote anteriori sono inclinati in modo che due rette immaginarie passanti per essi si incontrino al centro dell'assale posteriore.

In realtà, le caratteristiche di tenuta di strada degli pneumatici consentono leggere variazioni rispetto ad una rigida applicazione del principio di Ackermann (che, comunque, rimane valido) e si utilizzano così dei meccanismi che producono un'ottima approssimazione della situazione ideale.

Un angolo di Ackermann pronunciato si traduce in un comportamento di guida dolce e prevedibile: la macchina percorrerà le curve con precisione, senza che le quattro ruote tirino verso direzioni diverse. Un angolo minore, invece, da più direzionalità, specialmente nell'inserimento di curva, ma non è garantito che ogni tanto l'avantreno non "scappi" creando così un raggio di sterzata non uniforme.



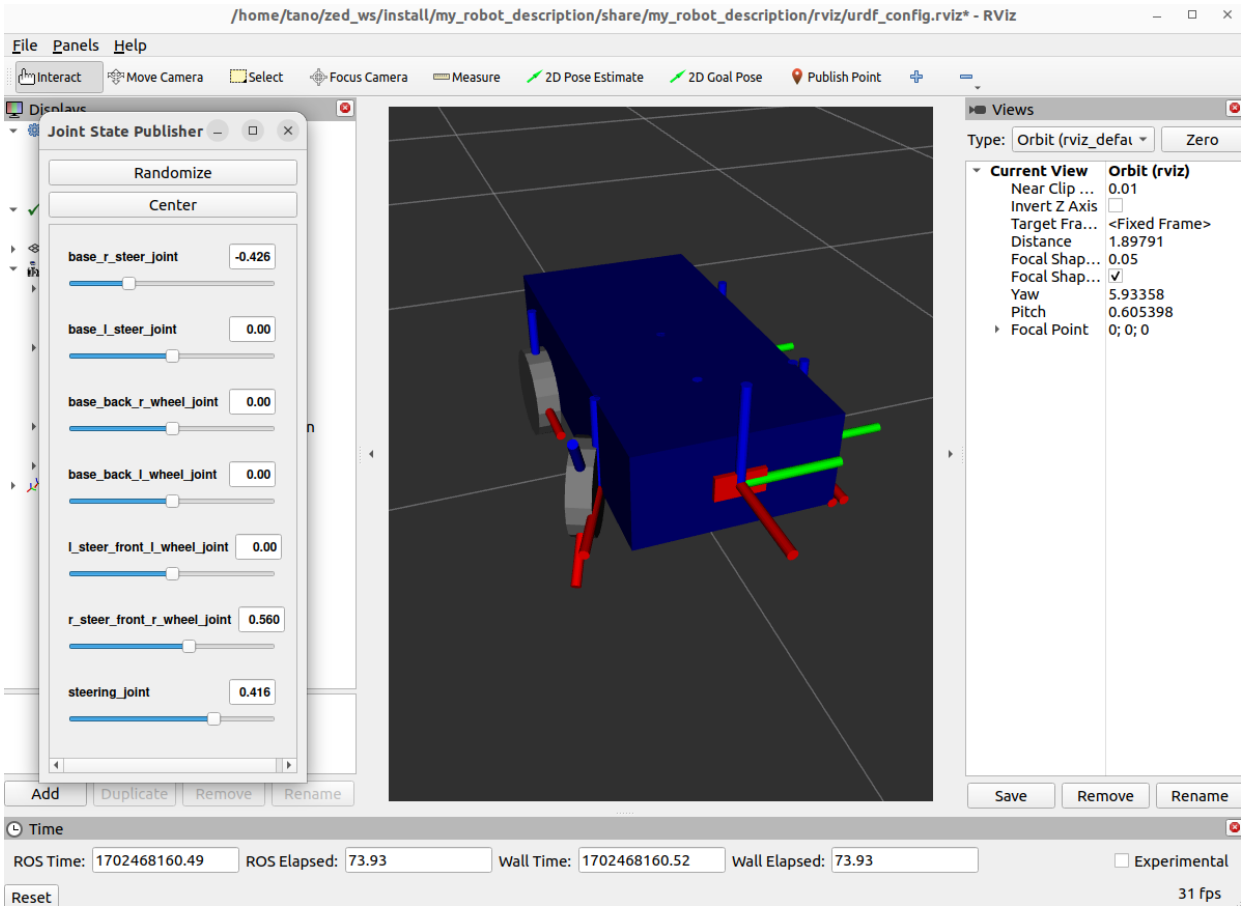
CREAZIONE DEL NUOVO ROBOT E MODIFICA DEL FILE URDF

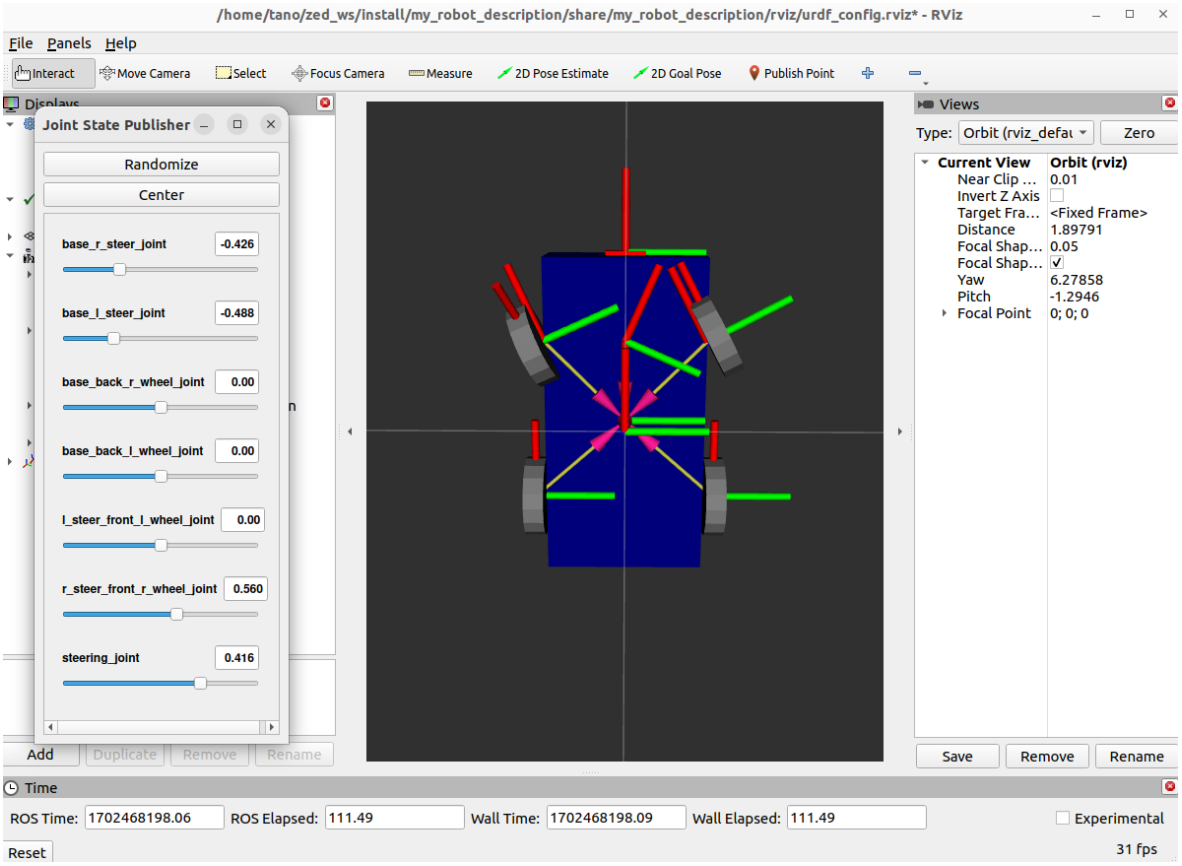
Sulla base delle considerazioni precedenti, è stata necessaria una revisione alla struttura del robot usato per il testing, in modo da adattarlo alle modifiche richieste; perciò, sono state aggiunte le altre due ruote e gli elementi necessari per l'implementazione dell'Ackermann.

```
<link name="r_steer">
    <inertial>
      <origin rpy="0 0 0" xyz="0.003381 1.5929e-07 0.02162" />
      <mass value="0.1" />
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0" />
    </inertial>
  </link>

<link name="l_steer">
    <inertial>
      <origin rpy="0 0 0" xyz="0.003381 1.5929e-07 0.02162" />
      <mass value="0.1" />
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0" />
    </inertial>
  </link>
```

```
<!--ACKERMANN STEER JOINTS-->
<joint name="base_r_steer_joint" type="revolute">
    <origin rpy="0 0 0" xyz="${base_length / 4.0} ${-base_width / 2.0} 0" />
    <parent link="base_link" />
    <child link="r_steer" />
    <axis xyz="0.0 0.0 1.0" />
    <dynamics damping="0.01" friction="0.01" />
    <limit effort="1000.0" lower="-0.95" upper="0.95" velocity="100" />
  </joint>

<joint name="base_l_steer_joint" type="revolute">
    <origin rpy="0 0 0" xyz="${base_length / 4.0} ${base_width / 2.0} 0" />
    <parent link="base_link" />
    <child link="l_steer" />
    <axis xyz="0.0 0.0 1.0" />
    <dynamics damping="0.01" friction="0.01" />
    <limit effort="1000.0" lower="-0.95" upper="0.95" velocity="100" />
  </joint>
```

```xml
<!--FRONT WHEEL JOINTS-->
<joint name="l_steer_front_l_wheel_joint" type="continuous">
    <parent link="l_steer" />
    <child link="front_l_wheel_link" />
    <origin xyz="0 ${wheel_length / 2} 0" rpy="0 0 0" />
    <axis xyz="0 1 0" />
</joint>

<joint name="r_steer_front_r_wheel_joint" type="continuous">
    <parent link="r_steer" />
    <child link="front_r_wheel_link" />
    <origin xyz="0 ${-wheel_length / 2} 0" rpy="0 0 0" />
    <axis xyz="0 1 0" />
</joint>
```

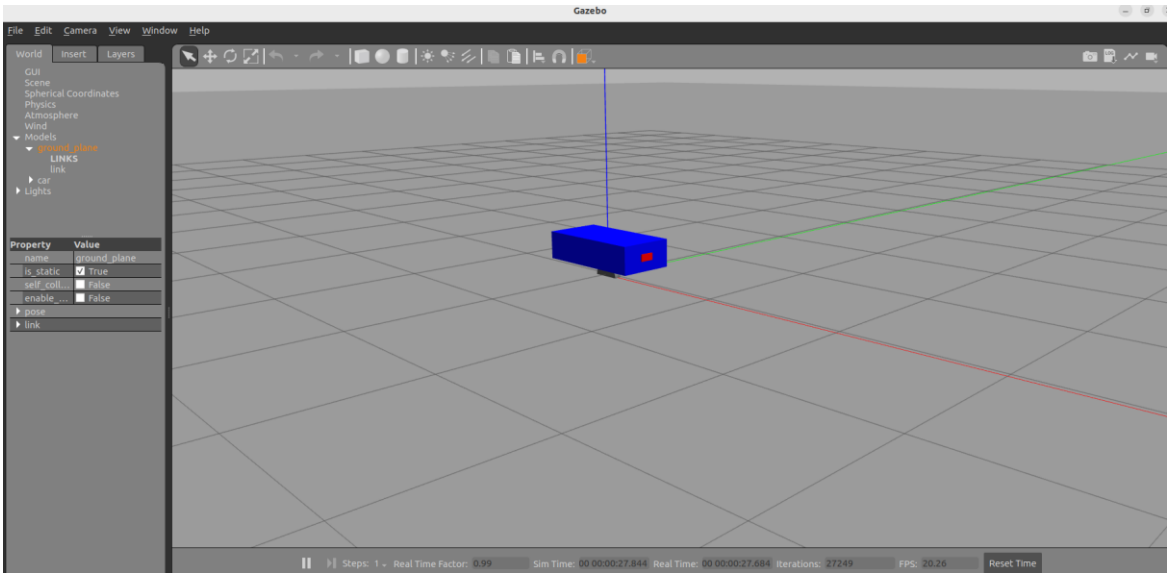


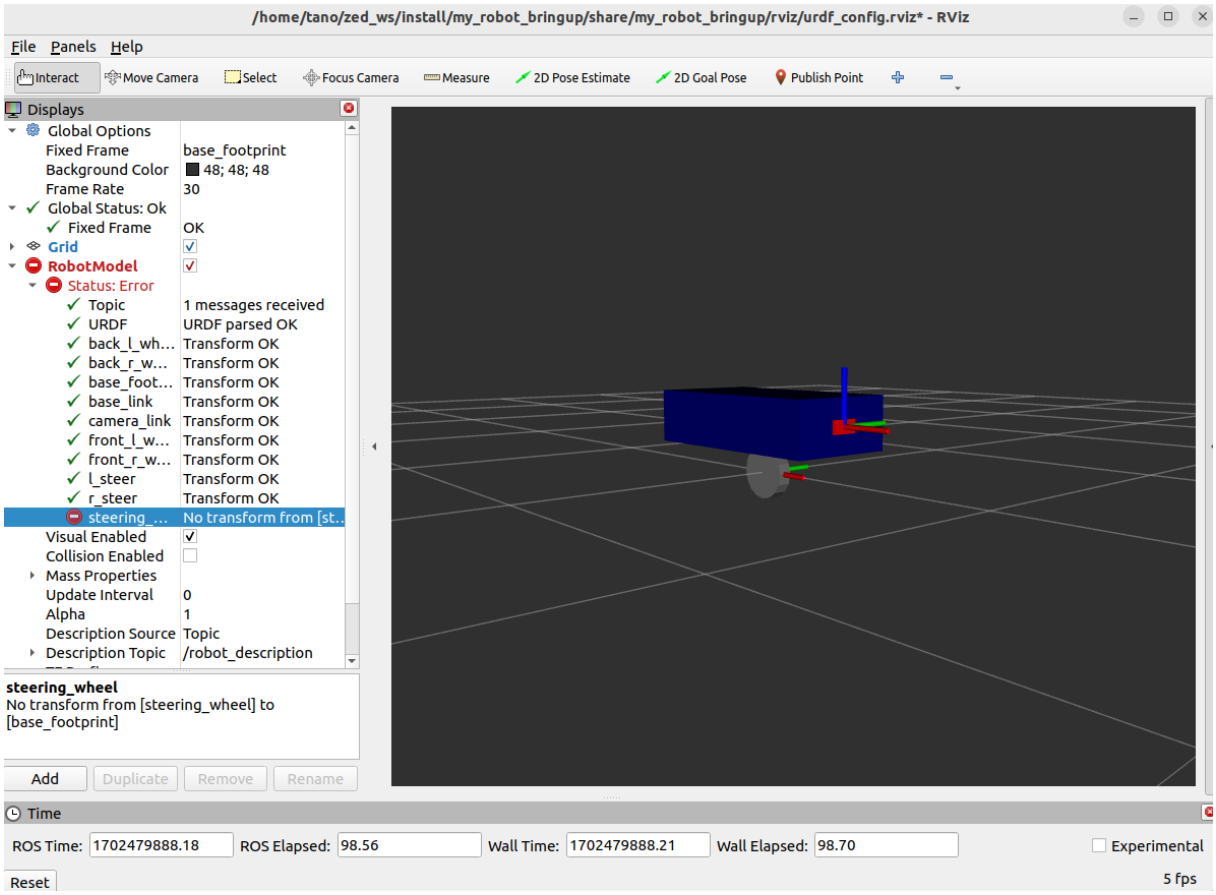


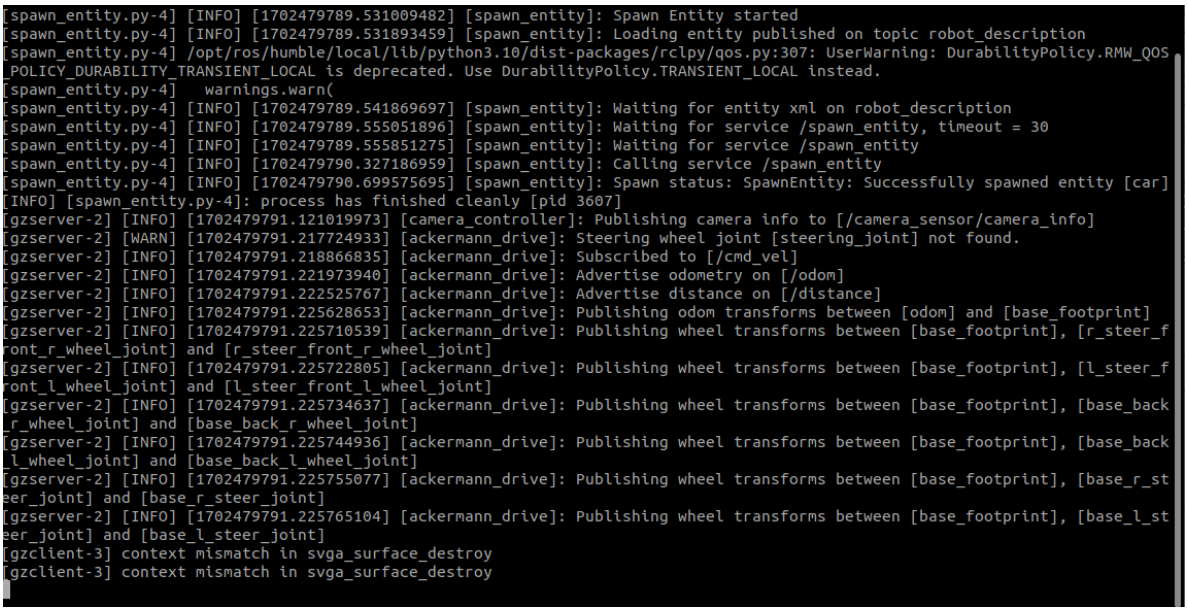
IMPLEMENTAZIONE SU GAZEBO

Purtroppo, allo stato attuale, nonostante non vi si riscontrino problemi nel file URDF, come testimonia il funzionamento del modello su RVIZ, il plugin Ackermann genera un errore durante la simulazione su Gazebo.

In particolar modo, si pensa che l'errore sia legato al plugin che non riesce in qualche modo a leggere correttamente i joint non fixed all'interno del file URDF, infatti, se si provano a sostituire i join di tipo revolute (con un movimento limitato -60/60 gradi) il modello viene letto e caricato correttamente assieme al plugin ma ovviamente non è possibile sfruttare le potenzialità del plugin Ackermann in quanto mancano le parti rotanti.


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
ground_plane
LINKS
link
car
Lights
Property Value
name ground_plane
is_static True
self_coll... False
enable_... False
pose
link
Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:00:27.844 Real Time: 00 00:00:27.684 Iterations: 27249 FPS: 20.26
Reset Time


/home/tano/zed_ws/install/my_robot_bringup/share/my_robot_bringup/rviz/urdf_config.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Goal Pose Publish Point
Displays
Global Options
Fixed Frame base_footprint
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid
RobotModel
Status: Error
Topic 1 messages received
URDF URDF parsed OK
back_l_wh... Transform OK
back_r_w... Transform OK
base_foot... Transform OK
base_link Transform OK
camera_link Transform OK
front_l_w... Transform OK
front_r_w... Transform OK
l_steer Transform OK
r_steer Transform OK
steering_... No transform from [st...
Visual Enabled
Collision Enabled
Mass Properties
Update Interval 0
Alpha 1
Description Source Topic
Description Topic /robot_description
steering_wheel
No transform from [steering_wheel] to [base_footprint]
Add Duplicate Remove Rename
Time
ROS Time: 1702479888.18 ROS Elapsed: 98.56 Wall Time: 1702479888.21 Wall Elapsed: 98.70
Experimental
Reset
5 fps


[spawn_entity.py-4] [INFO] [1702479789.531009482] [spawn_entity]: Spawn Entity started
[spawn_entity.py-4] [INFO] [1702479789.531893459] [spawn_entity]: Loading entity published on topic robot_description
[spawn_entity.py-4] /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/qos.py:307: UserWarning: DurabilityPolicy.RMW_QOS
_POLICY_DURABILITY_TRANSIENT_LOCAL is deprecated. Use DurabilityPolicy.TRANSIENT_LOCAL instead.
[spawn_entity.py-4]   warnings.warn(
[spawn_entity.py-4] [INFO] [1702479789.541869697] [spawn_entity]: Waiting for entity xml on robot_description
[spawn_entity.py-4] [INFO] [1702479789.555051896] [spawn_entity]: Waiting for service /spawn_entity, timeout = 30
[spawn_entity.py-4] [INFO] [1702479789.555851275] [spawn_entity]: Waiting for service /spawn_entity
[spawn_entity.py-4] [INFO] [1702479790.327186959] [spawn_entity]: Calling service /spawn_entity
[spawn_entity.py-4] [INFO] [1702479790.699575695] [spawn_entity]: Spawn status: SpawnEntity: Successfully spawned entity [car]
[INFO] [spawn_entity.py-4]: process has finished cleanly [pid 3607]
[gzserver-2] [INFO] [1702479791.121019973] [camera_controller]: Publishing camera info to [/camera_sensor/camera_info]
[gzserver-2] [WARN] [1702479791.217724933] [ackermann_drive]: Steering wheel joint [steering_joint] not found.
[gzserver-2] [INFO] [1702479791.218866835] [ackermann_drive]: Subscribed to [/cmd_vel]
[gzserver-2] [INFO] [1702479791.221973940] [ackermann_drive]: Advertise odometry on [/odom]
[gzserver-2] [INFO] [1702479791.222525767] [ackermann_drive]: Advertise distance on [/distance]
[gzserver-2] [INFO] [1702479791.225628653] [ackermann_drive]: Publishing odom transforms between [odom] and [base_footprint]
[gzserver-2] [INFO] [1702479791.225710539] [ackermann_drive]: Publishing wheel transforms between [base_footprint], [r_steer_f
ront_r_wheel_joint] and [r_steer_front_r_wheel_joint]
[gzserver-2] [INFO] [1702479791.225722805] [ackermann_drive]: Publishing wheel transforms between [base_footprint], [l_steer_f
ront_l_wheel_joint] and [l_steer_front_l_wheel_joint]
[gzserver-2] [INFO] [1702479791.225734637] [ackermann_drive]: Publishing wheel transforms between [base_footprint], [base_back
_r_wheel_joint] and [base_back_r_wheel_joint]
[gzserver-2] [INFO] [1702479791.225744936] [ackermann_drive]: Publishing wheel transforms between [base_footprint], [base_back
_l_wheel_joint] and [base_back_l_wheel_joint]
[gzserver-2] [INFO] [1702479791.225755077] [ackermann_drive]: Publishing wheel transforms between [base_footprint], [base_r_st
eer_joint] and [base_r_steer_joint]
[gzserver-2] [INFO] [1702479791.225765104] [ackermann_drive]: Publishing wheel transforms between [base_footprint], [base_l_st
eer_joint] and [base_l_steer_joint]
[gzclient-3] context mismatch in svga_surface_destroy
[gzclient-3] context mismatch in svga_surface_destroy