Progetto di Controllo dei Robot

Closed Loop DDP for a generic manipulator: an iLQR implementation

A.A. 2021/2022

Prof. Antonio Bicchi

Studenti:
Alessio Tumminello
Marco Ferreri

Research Center E. Piaggio
University of Pisa

Tutors:
Prof. Manolo Garabini
Ing. Franco Angelini
Ing. Mathew Jose Pollayil

Sommario

1. Introduzione alla Programmazione Dinamica
2. iLQR – iterative Linear Quadratic Regulator
3. Programmazione dinamica Closed Loop per un manipolatore generico: un’ implementazione iLQR
4. Simulazione su Gazebo
5. Risultati ottenuti
6. Innovazione e generalità del tool
7. Conclusioni
8. Sviluppi futuri

1. Introduzione alla Programmazione Dinamica

- **Differential Dynamic Programming**

  Si tratta di un algoritmo di controllo ottimo all'interno della classe di ottimizzazione delle traiettorie. L'algoritmo utilizza modelli localmente quadratici della dinamica e delle funzioni di costo e presenta una convergenza quadratica. È strettamente legato al metodo di Newton.

- **iLQR – iterative Linear Quadratic Regulator**

  Il DDP classico richiede le derivate di secondo ordine della dinamica, che di solito sono la parte più costosa del calcolo. Se si mantengono solo i termini del primo ordine, si ottiene un'approssimazione Gauss-Newton nota come **iterative Linear Quadratic Regulator** (**iLQR**), che è simile alle iterazioni Riccati, ma tiene conto della regolarizzazione e del line-search necessari a gestire la non linearità.

2. iLQR – iterative Linear Quadratic Regulator

A partire dalla dinamica a Tempo Continuo:

$$\begin{cases} \dot{q} \\ \ddot{q} = M^{-1}[-C(q,\dot{q}) - G(q) - D\dot{q} - Fsgn(\dot{q}) + \tau] \end{cases}$$

$$x_1 = q$$
$$x_2 = \dot{q}$$

$$\begin{cases} \dot{x_1} = x_2 \\ \dot{x_2} = M^{-1}[-C(x_2) - G - D(x_2) - Fsgn(\dot{q}) + \tau] \end{cases}$$

Si è effettuata una procedura di discretizzazione usando Runge-Kutta del 4°ordine:

$$x_{k+1} = f(x_k, u_k, k)$$

E si sono valutati i jacobiani simbolici. Fatto ciò abbiamo sostituito lo stato e l’ingresso di controllo:

$$f_x = \frac{\partial f}{\partial x_k} \qquad f_u = \frac{\partial f}{\partial u_k}$$

2. iLQR – iterative Linear Quadratic Regulator

Dal **funzionale di costo** fornito dall'utente:

$$J_t(x, U_t) = \sum_{i=t}^{N-1} l(x_i, u_i) + l_f(x_N)$$

Sono stati valutati i ***gradienti*** ad x_k e u_k del cost stage «l» e del cost final «l_f» :

$$l_x = \frac{\partial l}{\partial x_k} \qquad l_{f_x} = \frac{\partial l_f}{\partial x_k}$$
$$l_u = \frac{\partial l}{\partial u_k} \qquad l_{f_u} = \frac{\partial l_f}{\partial u_k}$$

E gli ***hessiani*** ad x_k e u_k:

$$l_{xx} = \frac{\partial l_x}{\partial x_k} \qquad l_{ux} = \frac{\partial l_u}{\partial x_k} \qquad l_{f_{xx}} = \frac{\partial l_{f_x}}{\partial x_k}$$
$$l_{uu} = \frac{\partial l_u}{\partial u_k} \qquad l_{xu} = \frac{\partial l_x}{\partial u_k} \qquad l_{f_{uu}} = \frac{\partial l_{f_u}}{\partial u_k}$$

2. iLQR – iterative Linear Quadratic Regulator

La **Q-function** è l'analogo in Tempo Discreto dell'Hamiltoniana, nota come *pseudo-Hamiltoniana*.

L'espansione del primo ordine di Q è data da:

$$Q_{\mathbf{x}} = \ell_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^{\top} V'_{\mathbf{x}}$$
$$Q_{\mathbf{u}} = \ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^{\top} V'_{\mathbf{x}}$$

$$Q_{\mathbf{xx}} = \ell_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^{\top} V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}}$$
$$Q_{\mathbf{ux}} = \ell_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^{\top} V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}}$$
$$Q_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^{\top} V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{u}}$$

Dove V' è la **value function** allo step successivo:

$$V_{\mathbf{x}} = Q_{\mathbf{x}} - \mathbf{K}^{\top} Q_{\mathbf{uu}} \mathbf{k}$$
$$V_{\mathbf{xx}} = Q_{\mathbf{xx}} - \mathbf{K}^{\top} Q_{\mathbf{uu}} \mathbf{K}$$

Mentre i guadagni di feed-back (della perturbazione dello stato) e feed-forward, i quali realizzano l' aggiornamento del controllo ottimo sono:

$$\mathbf{k} \triangleq -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} \qquad \mathbf{K} \triangleq -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}$$

$$\boxed{u_{k+1} = u_k + k + K(x_{k+1} - x_k)}$$

2. iLQR – **Pseudo codice**

Run iLQR algorithm

1. **Inizializzazione**
    1. Evoluzione della dinamica con un ingresso random e un initial guess per allocare la memoria:
$$u_0 = rand;\ x_0$$
$$x_{k+1} = F(x_k, u_k)$$
    2. Valutazione del funzionale di costo: $J(x_k, u_k)$

for *i* in range(*max_iteration*):

2. **Backward pass**
   Guadagni ottimi:
$$k,\ K = backward\_pass(x_k, u_k, regu)$$
3. **Forward pass**
   Coppie in ingresso ottime e stato predetto*:*
$$x_{k+1}, u_{k+1} = forward\_pass(x_k, u_k, k, K)$$
4. **Line Search**
   Se il costo si è ridotto, allora diminuisci il fattore di regolarizzazione
   Else aumentalo

3. Programmazione dinamica Closed Loop per un manipolatore generico

Nel codice è stata scritta una procedura di **setup** che viene **eseguita solo una volta**. All'interno di questa sezione del programma viene caricato l'URDF del robot e viene ottenuto il modello dinamico come descritto in precedenza.

Successivamente esso viene discretizzato e vengono ottenuti gli jacobiani e le hessiane in simbolico.
In questo specifico problema il seguente funzionale è stato considerato.

$$l(x_i, u_i) = (q_d - q)^T Q (q_d - q) + u^T R\, u$$

$$l_f(x_N) = 0$$

Dopo aver ottenuto **offline** **un'espressione simbolica delle matrici**, esse vengono utilizzate per ottenere delle **funzioni** chiamabili **online** che ricevono come argomenti lo stato e gli ingressi di controllo al passo precedente. Tali matrici vengono utilizzate per calcolare i guadagni di controllo come descritto precedentemente.

3. Programmazione dinamica Closed Loop per un manipolatore generico

Dato che l'algoritmo parte dall'URDF esso è generale per qualsiasi tipo di manipolatore seriale.

Per i calcoli in simbolico si è utilizzata la libreria **CasADi** mentre il package *urdf2casadi* ci ha permesso di ottenere la dinamica in forma matriciale.

URDF


```xml
<?xml version="1.0" ?>
<!-- =================================================================================== -->
<!-- |    This document was autogenerated by xacro from rrbot.xacro                    | -->
<!-- |    EDITING THIS FILE BY HAND IS NOT RECOMMENDED                                  | -->
<!-- =================================================================================== -->
<!-- Revolute-Revolute Manipulator -->
<robot name="rrbot">
  <!-- Space btw top of beam and the each joint -->
  <!-- ros_control plugin -->
  <gazebo>
    <plugin filename="libgazebo_ros_control.so" name="gazebo_ros_control">
      <robotNamespace>/rrbot</robotNamespace>
      <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
    </plugin>
  </gazebo>
  <!-- Link1 -->
  <gazebo reference="link1">
    <material>Gazebo/Orange</material>
  </gazebo>
  <!-- Link2 -->
  <gazebo reference="link2">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
    <material>Gazebo/Black</material>
  </gazebo>
  <!-- Link3 -->
  <gazebo reference="link3">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
    <material>Gazebo/Orange</material>
  </gazebo>
  <!-- Used for fixing robot to Gazebo 'base_link' -->
  <link name="world"/>
  <joint name="fixed" type="fixed">
    <parent link="world"/>
    <child link="link1"/>
  </joint>
  <!-- Base Link -->
  <link name="link1">
    <collision>
      <origin rpy="0 0.0 0" xyz="0 0 1.0"/>
      <geometry>
        <box size="0.1 0.1 2"/>
      </geometry>
    </collision>
    <visual>
      <origin rpy="0 0.0 0" xyz="0 0 1.0"/>
      <geometry>
        <box size="0.1 0.1 2"/>
      </geometry>
      <material name="orange"/>
    </visual>
```

URDF2 CASADI

CasADi

Dinamica tempo continuo in simbolico

$$\begin{cases} \dot{q} \\ \ddot{q} = M^{-1}[-C(q,\dot{q}) - G(q) - D\dot{q} - Fsgn(\dot{q}) + \tau] \end{cases}$$

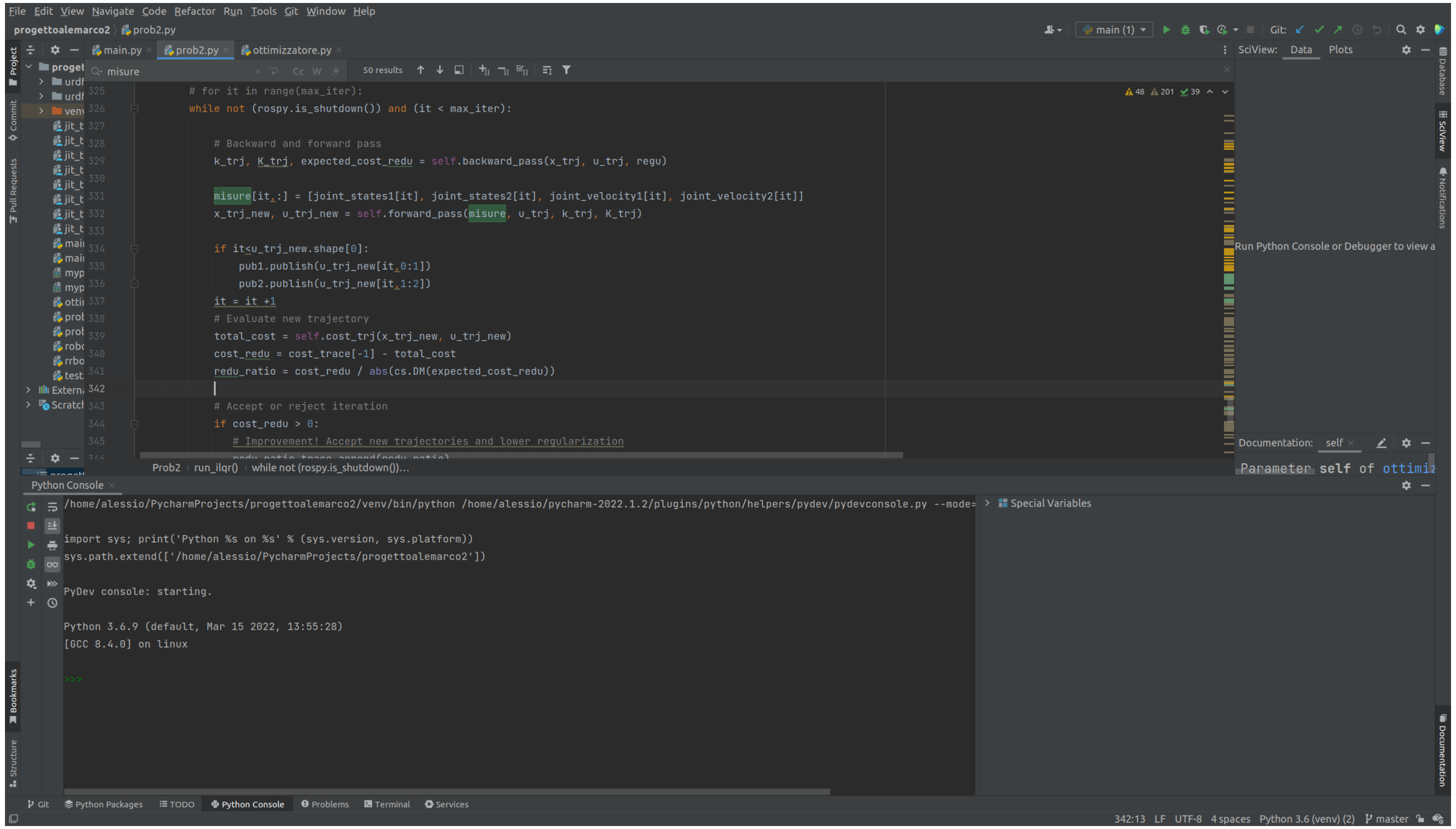
Dinamica tempo discreto in simbolico
Utilizzata per calcolare jacobiani ed hessiani

$$x_{k+1} = f(x_k, u_k, k)$$

Runge kutta 4° order

3. Programmazione dinamica Closed Loop per un manipolatore generico

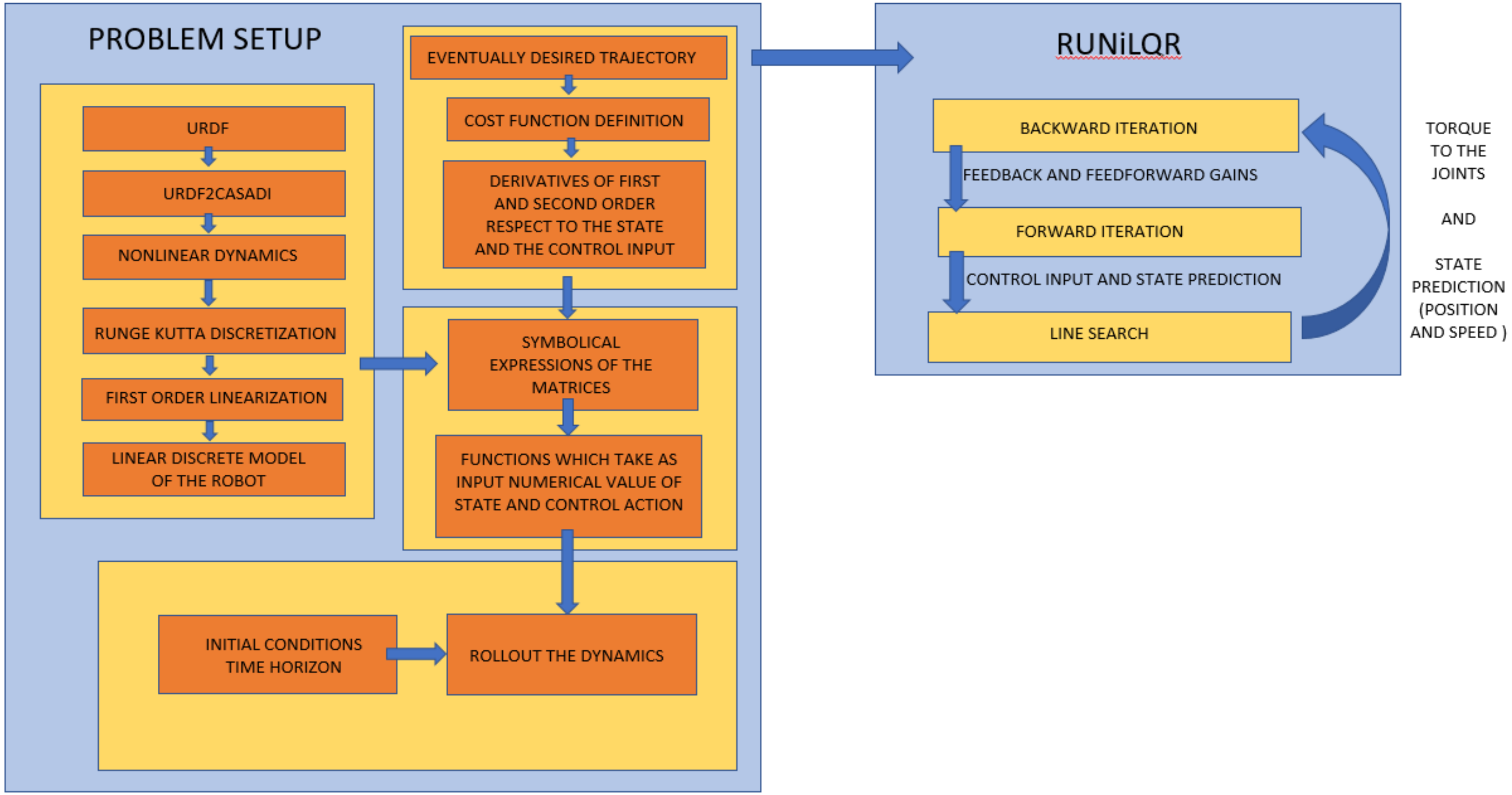
In una prima fase di debug dell'algoritmo si è **retroazionato lo stato stimato ottenuto dalla forward** all'istante precedente. Ciò ci ha permesso di testare l'algoritmo iterativo **senza** utilizzare **simulatori esterni**.
In questa fase di debug delll'algoritmo ci si è avvalsi del software *pycharm* che fornisce un comodo debugger.



Per la struttura del codice basata sulla programmazione a classi si è presi spunto dalla libreria Flex sviluppata dai colleghi M.Biasizzo e A.Camiletto*

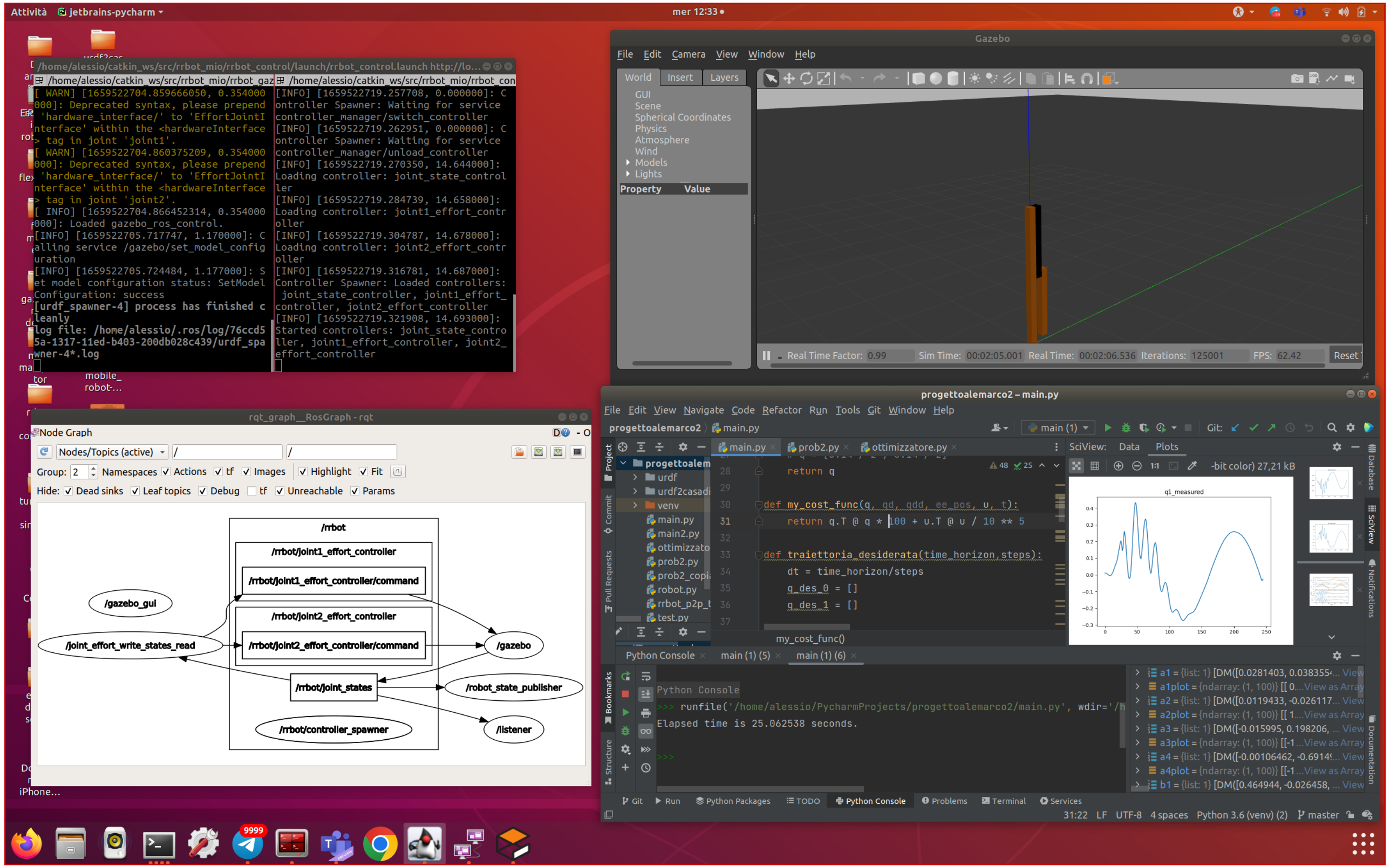
3. Programmazione dinamica Closed Loop per un manipolatore generico

Debug dell'algoritmo con retroazione dello stato stimato dalla forward (senza misure ottenute dalla simulazione su Gazebo)



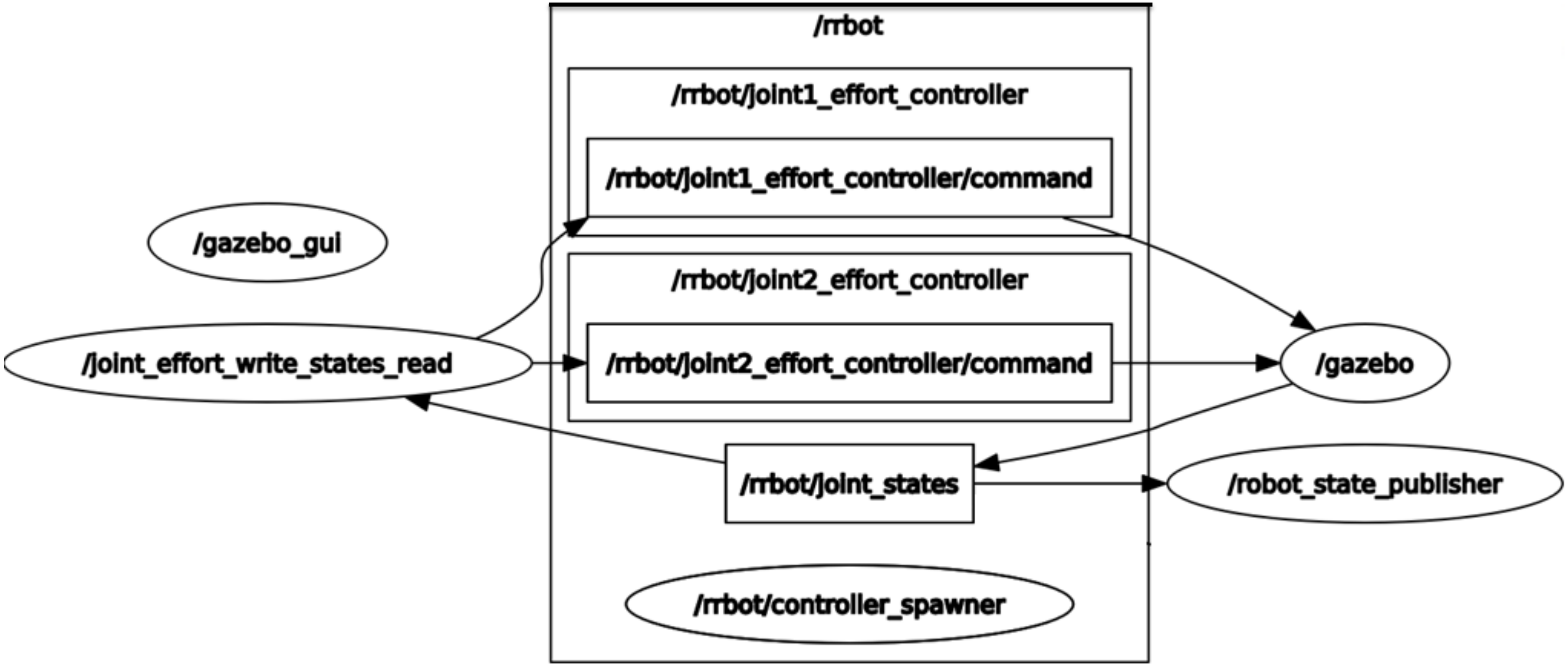
4. Simulazione su Gazebo

In seguito, per la fase di simulazione si sono utilizzati **ROS** e **Gazebo** per simulare il robot ed acquisire le misure. Nel nostro caso le misure di gazebo coincidono con lo stato (**C = identità**).



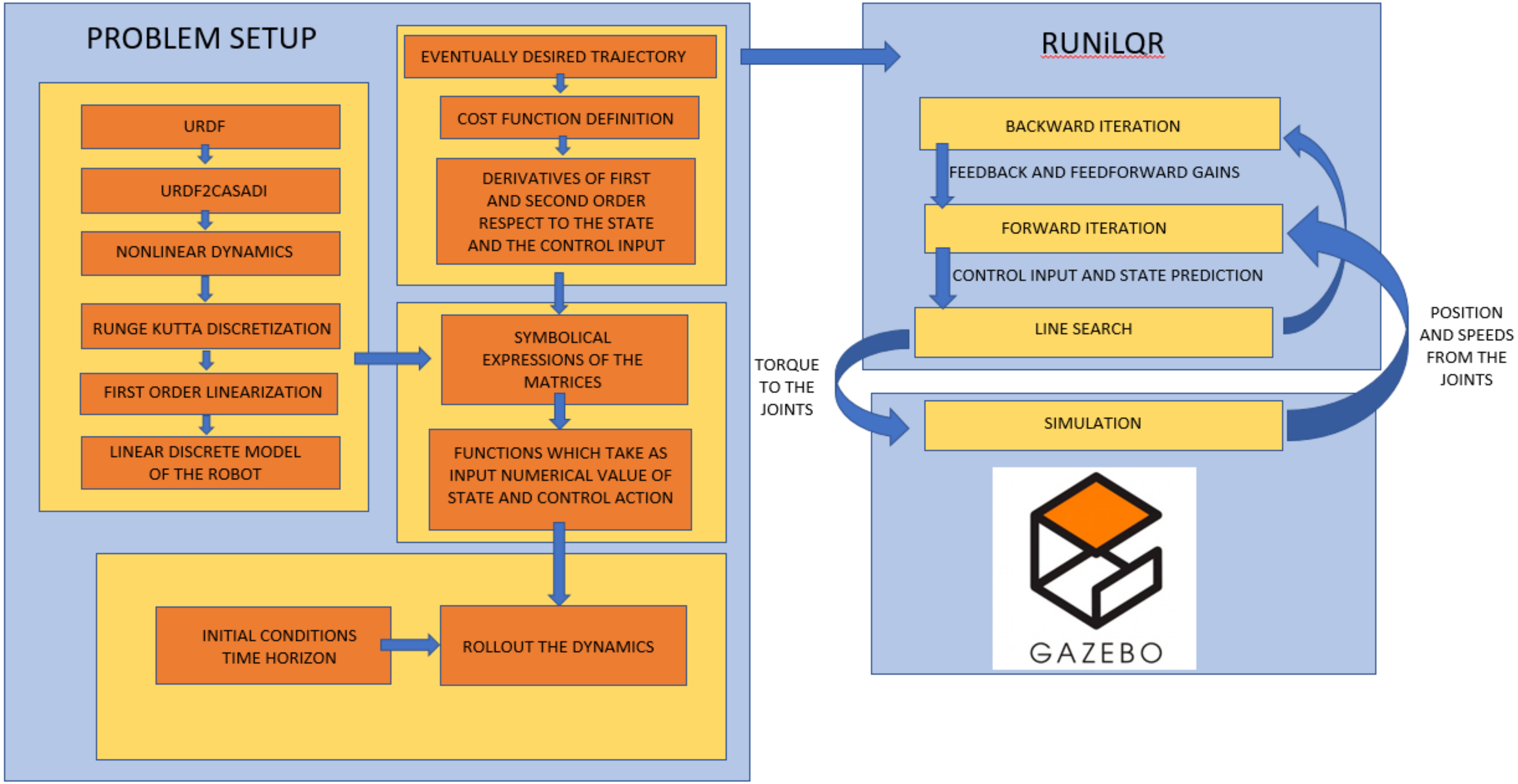
4. Simulazione su Gazebo

Per acquisire le misure dal simulatore e per applicare le coppie ottime ottenute dal controllo, che ci permettono di inseguire gli angoli di giunto desiderati, si sono utilizzati i seguenti nodi e topic:



4. Simulazione su Gazebo

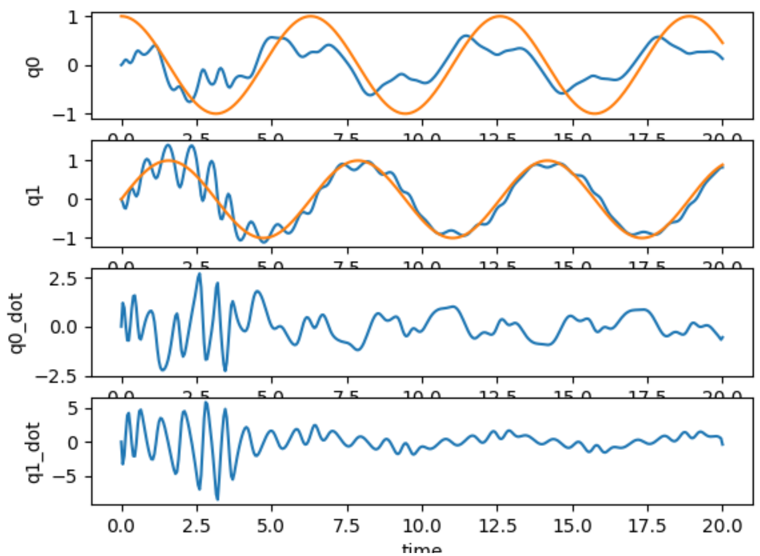
Closed Loop DDP for a generic robot



5. Risultati Ottenuti

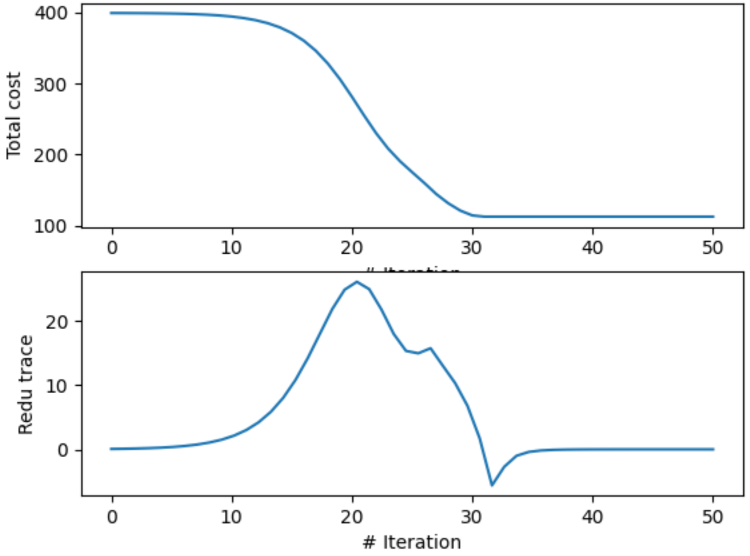
Qui si possono vedere gli andamenti delle variabili di stato rispetto ai valori desiderati. È possibile effettuare un tuning cambiando i pesi del funzionale di costo, così come visto nella teoria dell’LQR.

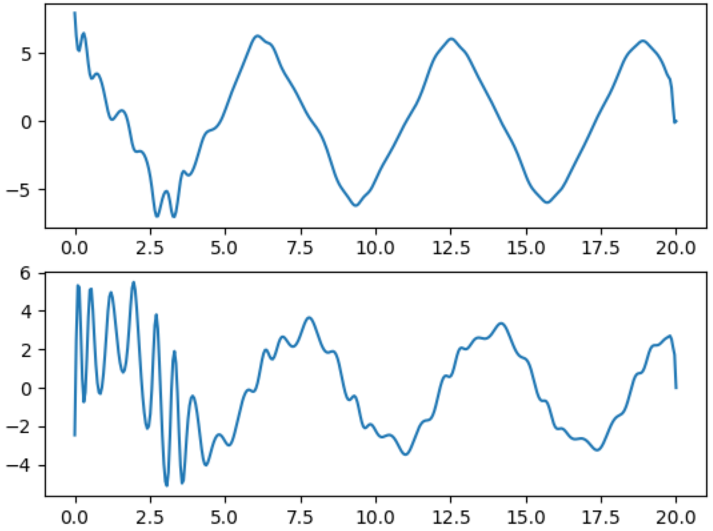
$q_{0_{des}}, q_{1_{des}}$



Minimizzazione del funzionale

Fattore di regolarizzazione (se >0 allora il valore del funzionale sta diminuendo)





$u_0 = \tau_0$

$u_1 = \tau_1$

6. Innovazione e generalità del Tool

L'***innovazione*** del nostro tool risiede nell'inseguimento degli angoli di giunto desiderati attraverso un controllo ottimo in coppia del robot, i cui guadagni sono ottenuti tramite l'iterazione degli algoritmi di Forward e Backward introdotti durante il corso di Sistemi Robotici Distribuiti.

L'ottenimento della dinamica attraverso il package *urdf2casadi* ci ha permesso di ***generalizzare*** il nostro tool a qualsiasi manipolatore seriale, avendo a disposizione semplicemente il file URDF descrittivo del manipolatore.

Inoltre la possibilità da parte dell'utente di definire **funzionali di costo specifici** consente di utilizzare il tool per realizzare task diversi (trajectory tracking, path following, pick and place).

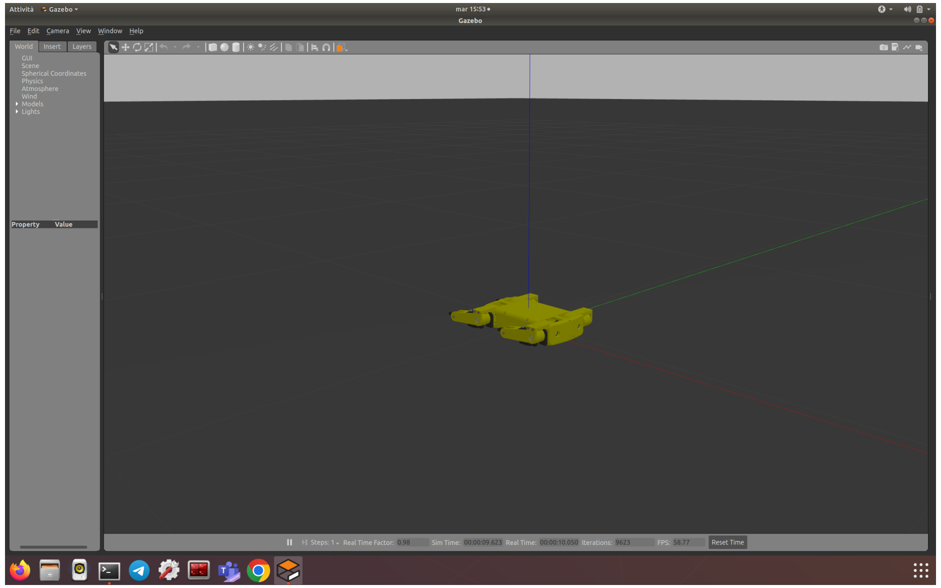
7. Conclusioni

Considerato che Python è un linguaggio di alto livello interpretato e non compilato, esso **non è real-time** per definizione. Abbiamo sperimentato che l'unico modo per eseguire una simulazione che non sfori i tempi di calcolo è quello di selezionare un relativamente **piccolo numero di steps** all'interno dell'orizzonte di simulazione.

Ciò si traduce in chiare difficoltà implementative nel sistema reale senza la traduzione dell'algoritmo in un linguaggio più efficiente come C++.

8. Sviluppi Futuri

Con l’obiettivo di estendere la generalità del tool si è pensato di applicarlo ad un quadrupede, ma leggendo la documentazione del package *urdf2casadi* ci si è resi conto che esso è in grado di analizzare ed ottenere la dinamica **solamente** a partire dal file URDF di **catene cinematiche seriali** e non di alberi cinematici.



Gli autori della libreria hanno menzionato tra gli **sviluppi futuri** a lungo termine di tale libreria **l’estensione all’analisi di alberi cinematici**, ciò consentirebbe un ulteriore generalità del nostro tool.

Robot Dynamics with URDF & CasADi

Lill Maria Gjerde Johannessen[1], Mathias Hauan Arbo[1], and Jan Tommy Gravdahl[1]

7.2.1 Generalization

The URDF represents robots with the structure of a branched kinematic tree. However, urdf_parser_py only supports parsing of kinematic chains, thus limiting u2c to only obtaining kinematics and dynamics of kinematic chains. Hence, u2c should, in the long run, be modified to support branched kinematic trees. This

104