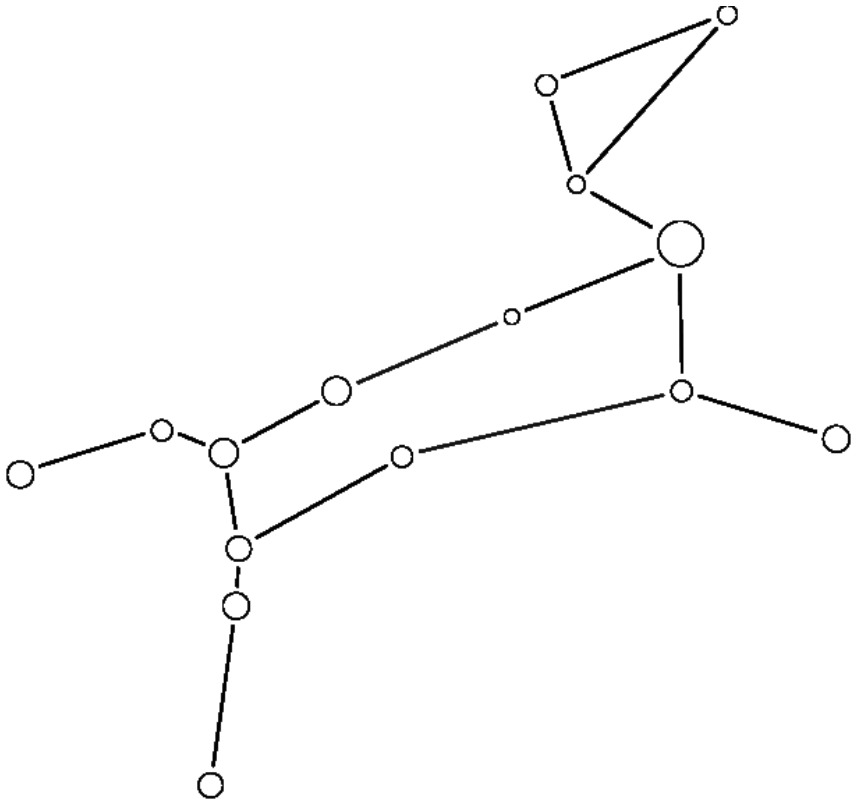
Control Systems Lab - Legged Team

Introductory task

Simulation Frameworks for prototype robotic leg

National Technical University of Athens
School of Mechanical Engineering



Author: Michail Papadakis

July 20, 2025

Contents

List of Figures

List of Tables

1. Introduction

The purpose of this introductory task is to create two simulation frameworks in ros using the gazebo simulator and in simscape. To test these frameworks, two control tasks are to be simulated: a static task in which the leg must exert a constant vertical force, and a dynamic task where the leg must follow an elliptical trajectory. The two frameworks are validated using a custom-made analytical model that simulates the same tasks and are compared with each other.

2. Creating a simplified 3D model

This section contains the geometric and dynamic properties of the leg and information regarding their calculations or estimations. The model must contain the following characteristics:

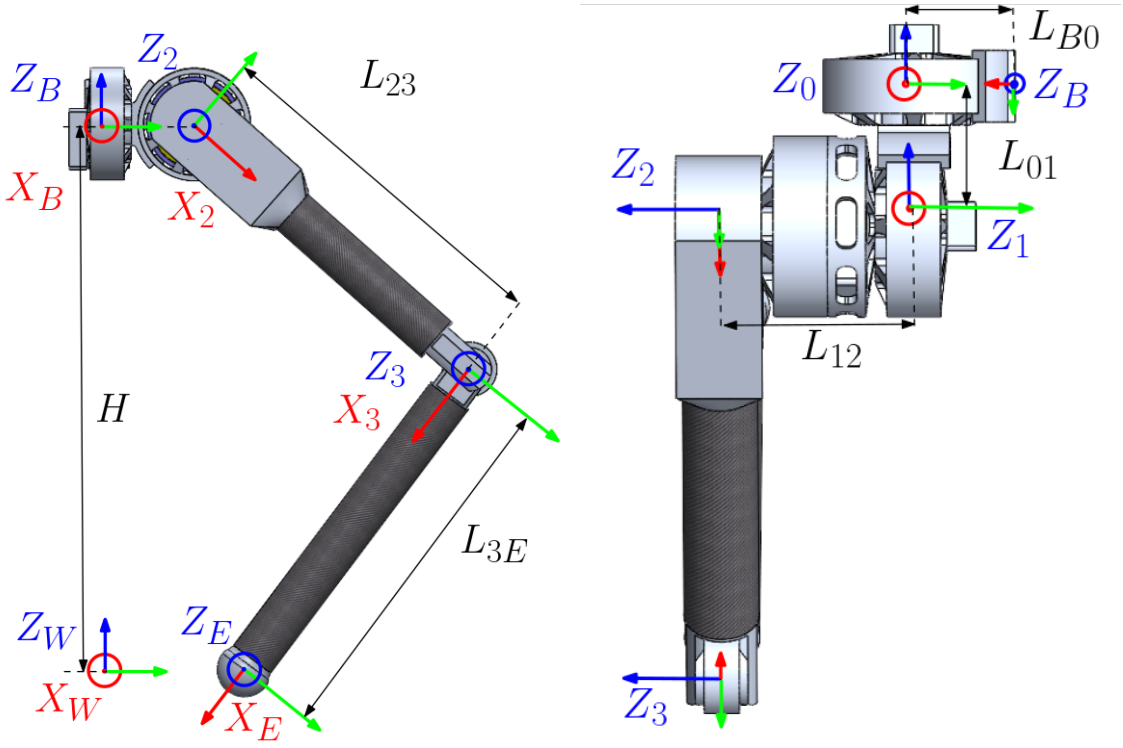
- Kinematic properties (geometric distances and joint limits)
- Dynamic properties (inertial quantities, friction coefficients)
- Collision properties.

2.1 Kinematic Properties

2.1.1 Geometric Quantities

A simplified 3D cad model was created using some data from J. Valvis's thesis [3] and **mainly** from measurements from the real leg. In the appendix (section 8.1) more information for the 3d modeling in solidworks is provided.

In figures 1a and 1b below, the final model is presented along with its coordinate systems[1]. The main kinematic parameters are also displayed and their arithmetic values, which are obtained from solidworks[2], are presented in table 1.



(a) Front view. (b) Top view

Figure 1: Coordinate frames of the leg.

Table 1: Table with the geometric quantities that affect the leg kinematics.

H	L_{B0}	L_{01}	L_{12}	L_{23}	L_{3E}
400	55.55	77.63	112.08	253.1	223

[1] In Iro's thesis [1], the z axis direction is such that the leg has to rotate in the negative direction in order to touch the ground. The same convention was used in the current model.

[2] Height was chosen arbitrarily.

The Denavit-Hartenberg (DH) parameters[3] in table 2, can be constructed using the above information.

Table 2: DH parameters.

Link i	**a**	**L**	**d**	**ϑ**
1	0	0	$-L_{01}$	q_1
2	$\pi/2$	0	L_{12}	$-\pi/2 + q_2$
3	0	L_{23}	0	q_3
E	0	L_{3E}	0	0

For the correct kinematic simulation, the lengths of table 1 must be measured exactly. Also, from the previous analysis, it is implied that the zero position of the robot is equivalent to a fully extended horizontal leg.

2.1.2 Kinematic Limits

Regarding the kinematic limits of each joint, it is observed that joints 1 and 2 are free to perform infinite rotations. In contrast, the third joint collides with the second link, and is thus constrained. However, it is logical to assume that joint 1 is also limited, as in reality it would collide with the body of the quadruped robot.

The calculation of the angle limit of the third link is done based on figure 2:

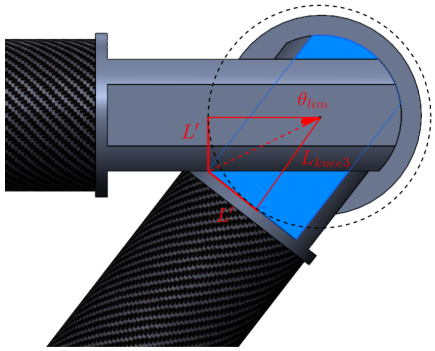


Figure 2: Calculation of joint 3 angle limit.

Where:

$$L' = 28.5mm,\ L_{knee3} = 27.95mm$$

$$\theta_{lim} = atan(\frac{L_{knee3}/2}{L'}) = 0.45589 \leq 0.456 \tag{1}$$

θ_{lim} is chosen a bit greater ($\theta_{lim} = 0.456$). Thus: $|\theta_2| \leq \pi - 2\theta_{lim} \approx 2.229$

The joint limits are summarised in table 3:

Table 3: Limits and characterization of each leg joint.

q_1	q_2	q_3
$\in\ [-\pi/2,\ \pi/2]$	$\in \mathbb{R}$	$\in\ [-2.229,\ 2.229]$
revolute joint	continuous joint	revolute joint

[3]It must be noted that the DH parameters are not directly used in the URDF. In the DH convention, the rotations take place around the local axis, while in the *URDF* rotations take place around the axis of the previous link.

2.2 Dynamic Properties

2.2.1 Inertial Quantities

Having defined the origin of the coordinate system of each link (in accordance with figures 1a and 1b) in the solidworks assemblies, it is easy to calculate the inertial properties of the links using the *Mass Properties* tool.

Solidworks provides the center of mass position C_o in the link coordinate system($\{L\}$):

$$C_o = x\hat{L_x} + y\hat{L_y} + z\hat{L_z}$$

and the inertial tensor in a coordinate system parallel to the link coordinate system, that has its origin in the center of mass (C_o). Using these quantities, the inertial properties can be inserted in the urdf under the `<inertial>` tag.

However, due to the different conventions regarding the products of inertia that are explained in the appendix (section 8.3), the products of inertia are inserted using the opposite sign.

At this point, it is important to state a problem regarding the inertial properties of the actuators, and in particular, the parts of the actuators that have a different velocity from the link that they actuate. While there is a `<mechanicalReduction>` sub-element in the `<transmission>` tag in the urdf, there is no way to insert the inertial properties of the actuator. In the appendix, in section 8.4, it is proved that the inertia of the actuator cannot be included in the link inertia in the general case. For this reason, **the actuator inertia**, while contributing in the dynamics of the leg in real life, **is neglected in the simulations.**

The inertia, as it is visualized in the gazebo simulation environment, is presented in figure 3. In the appendix, in section 8.2, the inertial properties of the parts, as calculated from solidworks, are included for the sake of thoroughness.

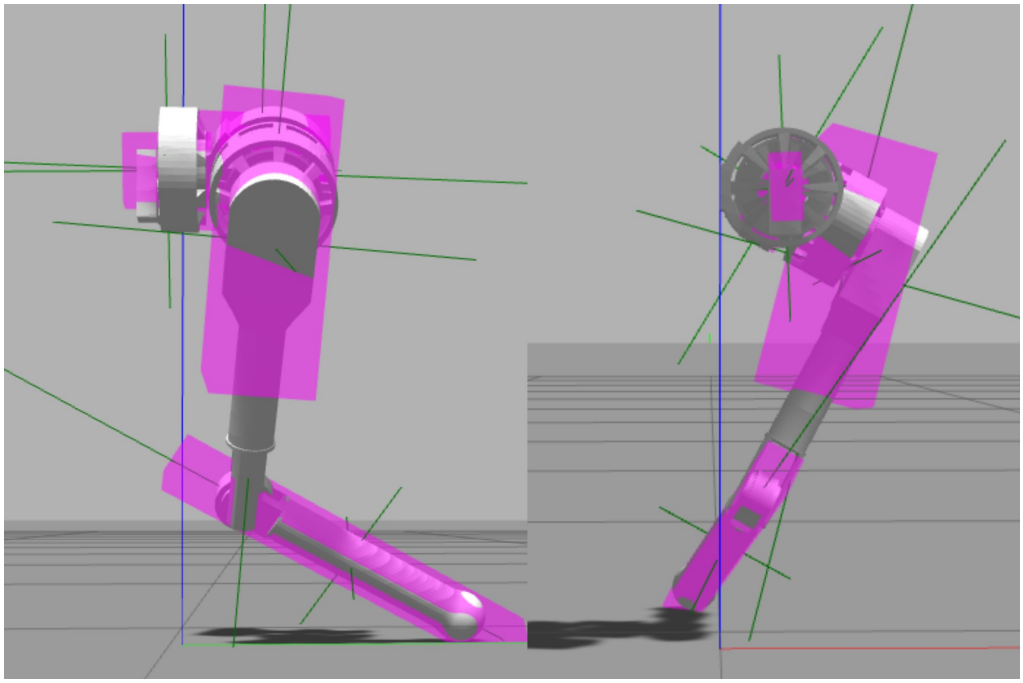


Figure 3: Inertial of the leg as visualized in gazebo.

2.2.2 Remaining joint properties

The remaining joint properties are the joint static friction and damping, which are set to zero, and the velocity and effort limits.

Joint velocity limit: From the dataset[4], the velocity limit is around 3709 rpm $= 388.4$ rad/s.

[4] `https://tmotor.en.made-in-china.com/product/wjOmskNcpHhu/China-T-Motor-U8-Lite-Kv100-Efficiency-BLDC-Aircraft-Uav-Quadcopter-Motor.html`

Joint torque limit: From J. Valvis's thesis [3], the maximum current that can flow through the motor in steady state is $i_{max} = 13.36A$. For the estimation of the motor constant, the torque-current characteristic was plotted using the motor manufacturer data, which is presented in figure 4:

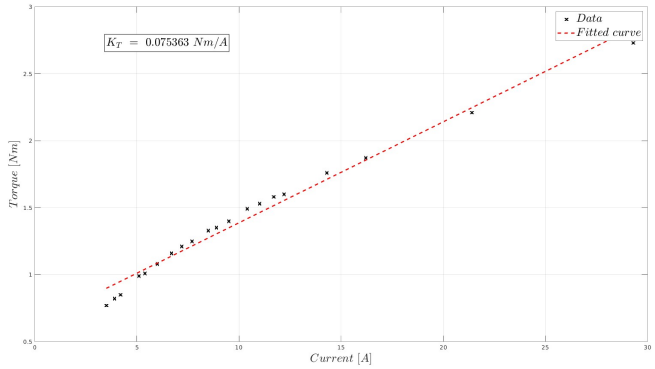


Figure 4: Motor constant estimation.

Using the above information, the maximum steady state[5] torque is:

$$\tau_{max} = K_t \cdot i_{max,continuous} = 0.0754 \cdot 13.36 =>$$

$$\boxed{\tau_{max,continuous} = 1.007344Nm}$$

Reductions of motors The actuator mechanical reductions are presented in table 4.

Table 4: Reductions of motors.

Ab/Ad (Joint 1)	**Hip (Joint 2)**	**Knee (Joint 3)**
1 : 7	1 : 10	1 : 10

In the URDF, the velocity and torque limits are included at the link level (as seen by the link). However, the manufacturer provides the aforementioned limits at the motor level. So, the limits are transformed using the equations (2) and (3).

$$\tau_{max,continuous,i} = k_{ir}\tau_{max} \tag{2}$$

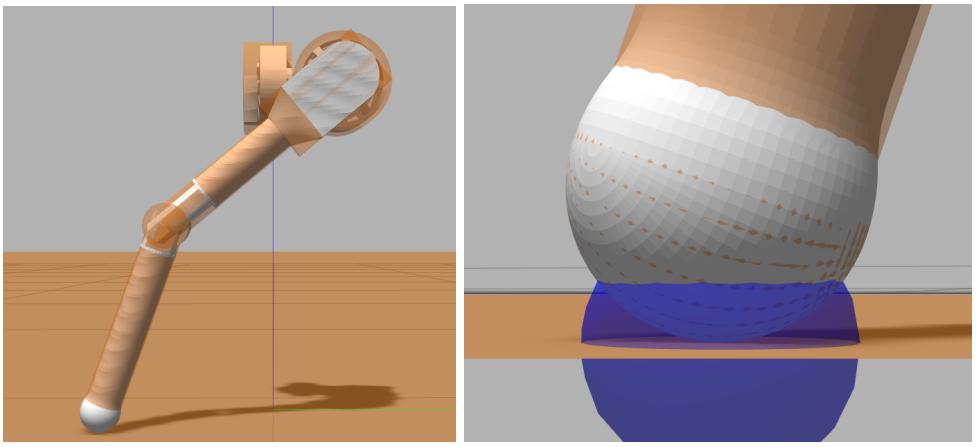
$$\omega_{max,i} = \frac{\omega_{max}}{k_{ir}} \tag{3}$$

where k_{ir} is the reduction of the i-th actuator.

[5]The maximum current is greater if the current flows in short bursts. $i_{peak} = 19.84A$ for $\Delta T \leq 720s$. This can be observed in Valvis's thesis [3], in figure 5.17.

2.3 Collision Properties

The collision geometry is comprised of geometric primitives that surround the whole robot. During the placement of each joint in the urdf file, the appropriate geometry was created. The results are presented in figures 5b and 5a.



(a) Front view of the whole leg. (b) Focused view on the foot.

Figure 5: Collision Geometry.

Apart from the collision geometry, the contact dynamical parameters must be set. These parameters depend on the contact model. Generally, a spring-damper ground interaction model was used with the addition of friction. In the simscape simulation this was achieved using a sphere to plane contact from the contact library[6]. In the gazebo simulation, the spring (K_p) and damping (K_d) parameters along with the static friction (μ_s) of the end effector were set in the `<collision>` element. The coefficient of restitution and torsional friction were set to zero. Thus, the contact parameters are presented in the table 5:

Table 5: Model collision properties.

K_p	1e4
K_d	1e2
μ_s	0.7

[6] `https://www.mathworks.com/matlabcentral/fileexchange/47417-simscape-multibody-contact-forces-library`

3. Leg Kinematics

This section contains the kinematic analysis of the leg.

3.1 Direct Kinematics

Using the DH parameters (Table 2), the homogeneous transformation matrices ${}^{i-1}\mathbf{T}_i$ are given by[7]:

$$ {}^{i-1}\mathbf{T}_i = \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & L_{i-1} \\ c(a_{i-1})s(\theta_i) & c(a_{i-1})c(\theta_i) & -s(a_{i-1}) & -s(a_{i-1})d_i \\ s(a_{i-1})s(\theta_i) & s(a_{i-1})c(\theta_i) & c(a_{i-1}) & c(a_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

The resultant matrices are presented below:

$$ {}^{W}\mathbf{T}_0 = \begin{bmatrix} 0 & -1 & 0 & L_{B0} \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & H \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5} $$

$$ {}^{0}\mathbf{T}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & -L_{01} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

$$ {}^{1}\mathbf{T}_2 = \begin{bmatrix} cos(\theta_2 - \pi/2) & -sin(\theta_2 - \pi/2) & 0 & 0 \\ 0 & 0 & -1 & -L_{12} \\ sin(\theta_2 - \pi/2) & cos(\theta_2 - \pi/2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_2 & c_2 & 0 & 0 \\ 0 & 0 & -1 & -L_{12} \\ -c_2 & s_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

$$ {}^{2}\mathbf{T}_3 = \begin{bmatrix} c_3 & -s_3 & 0 & L_{23} \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8} $$

$$ {}^{3}\mathbf{T}_E = \begin{bmatrix} 1 & 0 & 0 & L_{3E} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9} $$

The homogeneous transformation matrix ${}^{W}\mathbf{T}_E$ from the world frame to the end effector frame is:

$$ {}^{W}\mathbf{T}_E = {}^{W}\mathbf{T}_0 \cdot \underbrace{{}^{0}\mathbf{T}_1(q_1) \cdot {}^{1}\mathbf{T}_2(q_2) \cdot {}^{2}\mathbf{T}_3(q_3)}_{{}^{0}\mathbf{T}_3(q_1,q_2,q_3)} \cdot {}^{3}\mathbf{T}_E \tag{10} $$

3.2 Inverse Kinematics

Problem statement:

Let us consider the desired ${}^{W}\mathbf{P}_D = [{}^{W}x, {}^{W}y, {}^{W}z]^T$ expressed in the world frame. The desired orientation is random ($\tilde{\mathbf{R}}$) as it is not possible to set both the position and the orientation of the end effector. This will become apparent in the following analysis. Thus:

$$ {}^{W}\mathbf{T}_D = \begin{bmatrix} \tilde{\mathbf{R}} & {}^{W}\mathbf{P}_D \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} $$

The purpose of the Inverse Kinematics (IK) analysis is to find the joint angles that result in the given pose/configuration. Thus:

$$ {}^{W}\mathbf{T}_D = {}^{W}\mathbf{T}_0 \cdot {}^{0}\mathbf{T}_3(q_1, q_2, q_3) \cdot {}^{3}\mathbf{T}_E => $$

[7] Here $c(\theta) = cos(\theta)$ and $s(\theta) = sin(\theta)$.

$$ {}^0\mathbf{T}_3(q_1,q_2,q_3)\,{}^3\mathbf{T}_E = ({}^W\mathbf{T}_0)^{-1}\,.^W\,\mathbf{T}_D \tag{11} $$

where

$$ ({}^W\mathbf{T}_0)^{-1} = \begin{bmatrix} ({}^W\mathbf{R}_0)^T & -({}^W\mathbf{R}_0)^T\,.^W\,\mathbf{P}_0 \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & -H \\ -1 & 0 & 0 & L_{B0} \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

and

$$ ({}^W\mathbf{T}_0)^{-1}.{}^W\mathbf{T}_D = \begin{bmatrix} ({}^W\mathbf{R}_0)^T\tilde{\mathbf{R}} & ({}^W\mathbf{R}_0)^T\,.^W\,\mathbf{P}_D - ({}^W\mathbf{R}_0)^T\,.^W\,\mathbf{P}_0 \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} ({}^W\mathbf{R}_0)^T\tilde{\mathbf{R}} & \begin{bmatrix} {}^W z - H \\ L_{B0} - {}^W x \\ -{}^W y \end{bmatrix} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} $$

The desired position expressed in the 0 frame is:

$$ {}^0\mathbf{P}_D = [{}^0x_D,\ {}^0y_D,\ {}^0z_D]^T = [{}^W z - H,\ L_{B0} - {}^W x,\ -{}^W y]^T \tag{12} $$

${}^0\mathbf{T}_3$ is given by:

$$ {}^0\mathbf{T}_3 = \begin{bmatrix} c_1s_{2+3} & c_1c_{2+3} & s_1 & L_{12}s_1 + L_{23}c_1s_2 \\ s_1s_{2+3} & s_1c_{2+3} & -c_1 & -L_{12}c_1 + L_{23}s_1s_2 \\ -c_{2+3} & s_{2+3} & 0 & -L_{01} - L_{23}c_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13} $$

The ${}^0\mathbf{T}_E$ matrix is:

$$ {}^0\mathbf{T}_3 = \begin{bmatrix} c_1s_{2+3} & c_1c_{2+3} & s_1 & L_{12}s_1 + L_{23}c_1s_2 + c_1s_{2+3}L_{3E} \\ s_1s_{2+3} & s_1c_{2+3} & -c_1 & -L_{12}c_1 + L_{23}s_1s_2 + s_1s_{2+3}L_{3E} \\ -c_{2+3} & s_{2+3} & 0 & -L_{01} - L_{23}c_2 - c_{2+3}L_{3E} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14} $$

From equations (12) and (14) the following system { (15),(16),(17) } is obtained :

$$ L_{12}s_1 + L_{23}c_1s_2 + c_1s_{2+3}L_{3E} =^0 x_D \tag{15} $$

$$ -L_{12}c_1 + L_{23}s_1s_2 + s_1s_{2+3}L_{3E} =^0 y_D \tag{16} $$

$$ -L_{01} - L_{23}c_2 - c_{2+3}L_{3E} =^0 z_D \tag{17} $$

The following analysis will present the procedure to find the solutions to the system and the constraints that ${}^0P_D$ must satisfy.

Finding θ_1 :

$$ (15)s_1 - (\ 16)c_1 \Rightarrow L_{12} =^0 x_D s_1 -^0 y_D c_1 \tag{18} $$

Using the identity $c_1 = \sqrt{1 - s_1^2}$ the equation (18) becomes:

$$ ({}^0x_D^2 +^0 y_D^2)s_1^2 - 2L_{12}({}^0x_D)s_1 + L_{12}^2 -^0 y_D^2 = 0 \tag{19} $$

$$ \Delta = 4L_{12}^2({}^0x_D)^2 - 4(L_{12}^2 -^0 y_D^2)({}^0x_D^2 +^0 y_D^2) $$

For a solution to exist $\Delta \geq 0 \Rightarrow$

$$ \boxed{{}^0x_D^2 +^0 y_D^2 \geq L_{12}^2} \tag{20} $$

And the solution of (19) is

$$s_1 = \frac{2L_{12}({}^0x_D) \pm \sqrt{\Delta}}{2({}^0x_D^2 + {}^0y_D^2)}$$

and thus:

$$\boxed{\theta_1 = asin(\frac{2L_{12}({}^0x_D) \pm \sqrt{\Delta}}{2({}^0x_D^2 + {}^0y_D^2)})} \tag{21}$$

As $\theta_1 \in [-\pi/2, \pi/2]$, the $asin()$ function has a unique solution. However, due to the presence of "$\pm$", there are two solutions. In the programs, it is checked that both solutions satisfy equation (18).

Finding $\theta_2,\ \theta_3$:

For each of the solutions of (21), the following steps are taken:

$$(\ 16\ )s_1 + (\ 15\ )c_1 \Rightarrow L_{23}s_2 + L_{3E}s_{2+3} = {}^0x_Dc_1 + {}^0y_Ds_1 \triangleq K_1 \tag{22}$$

Rewriting the equation (17):

$$L_{23}c_2 + c_{2+3}L_{3E} = -{}^0z_D - L_{01} \triangleq K_2 \tag{23}$$

Equations (22), (23) are the classic IK system for a two link manipulator with the following solutions:

$$\boxed{\theta_3 = acos(\frac{K_1^2 + K_2^2 - L_{23}^2 - L_{3E}^2}{2L_{23}L_{3E}})} \tag{24}$$

Which has solutions when:

$$cos(2.229) \leq \frac{K_1^2 + K_2^2 - L_{23}^2 - L_{3E}^2}{2L_{23}L_{3E}} \leq 1$$

$$\boxed{L_{23}^2 + L_{3E}^2 + 2L_{23}L_{3E}cos(2.229) \leq K_1^2 + K_2^2 \leq (L_{23} + L_{3E})^2} \tag{25}$$

There are two solutions from the equation (24) for θ_3 that correspond to:

$$\left\{ \begin{array}{c} (c_3, s_3 = \sqrt{1 - c_3^2}) \\ \\ (c_3, s_3^* = -\sqrt{1 - c_3^2}) \end{array} \right\}$$

Thus, equations (22) and (23) are rewritten using the first θ_3 solution as:

$$\begin{bmatrix} L_{3E}s_3 & L_{3E}c_3 + L_{23} \\ L_{3E}c_3 + L_{23} & -L_{3E}s_3 \end{bmatrix} \begin{Bmatrix} c_2 \\ s_2 \end{Bmatrix} = \begin{bmatrix} A & B \\ B & -A \end{bmatrix} \begin{Bmatrix} c_2 \\ s_2 \end{Bmatrix} = \begin{Bmatrix} K_1 \\ K_2 \end{Bmatrix}$$

The determinant is :

$$det = -A^2 - B^2 = -L_{23}s_3^2 - L_{3E}^2c_3^2 - L_{23}^2 - 2L_{23}L_{3E}c_3 + = -(L_{23}^2 + L_{3E}^2 + 2L_{23}L_{3E}c_3) < 0$$

As $det \neq 0$, there is always a solutions to the system. So:

$$c_2 = -\frac{AK_1 + BK_2}{det}$$

$$s_2 = \frac{-BK_1 + AK_2}{det}$$

$$\boxed{\theta_2 = atan2(s_2, c_2)} \tag{26}$$

While the second solution is:

$$c_2^* = -\frac{-AK_1 + BK_2}{det}$$
$$s_2^* = \frac{-BK_1 - AK_2}{det}$$

$$\boxed{\theta_2^* = atan2(s_2^*, c_2^*)} \tag{27}$$

Using only the ${}^W P_D$, without defining the end - effector orientation, all the joint angles are fully defined.

3.3 Direct Differential Kinematics - Geometric Jacobian

The geometric jacobian can be computed by the following formula:

$$\mathbf{J}_V = \begin{bmatrix} {}^0\mathbf{J}_{V,linear} \\ {}^0\mathbf{J}_{V,angular} \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{z}_1 \times {}^0\mathbf{P}_E^1 & {}^0\mathbf{z}_2 & {}^0\mathbf{z}_n \times {}^0\mathbf{P}_E^n \\ {}^0\mathbf{z}_1 & 0 & {}^0\mathbf{z}_n \end{bmatrix} \tag{28}$$

where:

$${}^0\mathbf{z}_i = {}^0\mathbf{R}_i \; {}^i\mathbf{z}_i = {}^0\mathbf{R}_i [0,\ 0,\ 1]^T \tag{29}$$

and:

$${}^0\mathbf{P}_E^i = {}^0\mathbf{P}_E - {}^0\mathbf{P}_i \tag{30}$$

The rate of change of the orientation is of no interest[8], so only ${}^0\mathbf{J}_{V,linear}$ will be computed. From now on, ${}^0\mathbf{J}_{V,linear} = \mathbf{J}_V$. Using the equation (28), the geometric jacobian is:

$${}^0\mathbf{J}_V = \begin{bmatrix} L_{12}c_1 - L_{23}s_1s_2 - L_{3E}s_1s_{23} & (L_{3E}c_{23} + L_{23}c_2)c_1 & L_{3E}c_1c_{23} \\ L_{12}s_1 + L_{23}c_1s_2 + L_{3E}c_1s_{23} & (L_{3E}c_{23} + L_{23}c_2)s_1 & L_{3E}s_1c_{23} \\ 0 & L_{23}s_2 + L_{3E}s_{23} & L_{3E}s_{23} \end{bmatrix} \tag{31}$$

To change the reference frame of the jacobian, the equation (32) is used:

$${}^W\mathbf{J}_V = {}^W\mathbf{R}_0 \cdot {}^0\mathbf{J}_V \tag{32}$$

The following is also true:

$$det({}^W\mathbf{J}_V) = det({}^W\mathbf{R}_0) det({}^0\mathbf{J}_V) = det({}^0\mathbf{J}_V)$$

So the singularities of the jacobian do not change.

3.3.1 Singularities

To find singularities in the Jacobian, the determinant is computed:

$$det(\mathbf{J}_V) = L_{23}L_{3E}^2(c_2c_3^2 - c_2 - c_3s_2s_3) - L_{23}^2L_{3E}s_2s_3 \tag{33}$$

Singularities occur when:

- $s_3 = 0$. Due to angle limits, this results in $q_3 = 0$. This is the known singularity resulting from an RR manipulator being straight[9].
- When the following equation is satisfied:

$$-L_{3E}s_{3+2} - L_{23}s_2 = 0 \tag{34}$$

There are infinately many solutions to this equation, that satisfy:

$$q_3 = -q_2 - sin^{-1}\left(\frac{L_{23}}{L_{3E}}sin(q_2)\right) \tag{35}$$

$$q_3 = -q_2 + sin^{-1}\left(\frac{L_{23}}{L_{3E}}sin(q_2)\right) + \pi \tag{36}$$

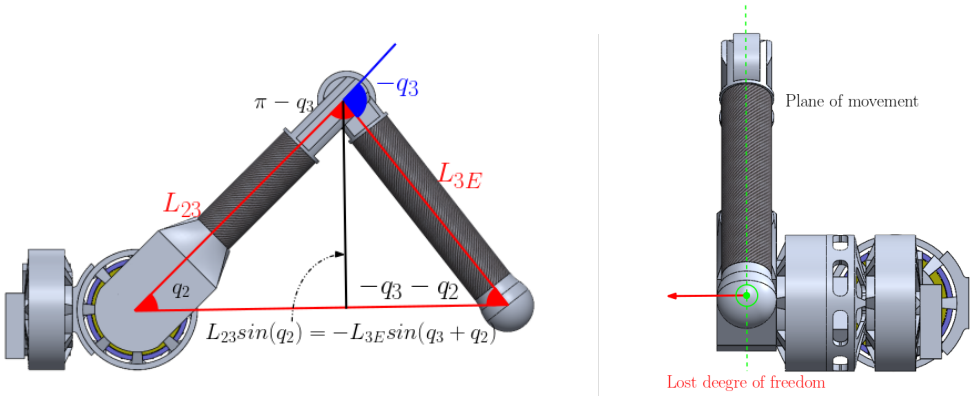
This singularity expresses the law of sines[10] and has the following geometric meaning:

[8] To set the 6DOFs of the end effector, 6 actuators are needed at least.

[9] If the 2nd and 3rd degree of freedom are seen as an RR manipulator

[10] The law of sines is the following:

$$\frac{s_{3-2}}{L_{23}} = \frac{s_2}{L_{3E}}$$



(a) Geometric meaning of equation (34).

(b) Lost degree of freedom due to second singularity.

Figure 6: Geometric meaning of second singularity.

4. Gazebo Simulation Framework

To create the simulation framework, two packages were created: `leg_description` and `leg_control`. The former contains the leg urdf file and a static library (`Leg`) that contains a class responsible for logging the states of the robot and provides methods for the robot kinematics. The latter contains the controller implementations.

4.1 `leg_description` package and `Leg` library

The package contains a `.xarco` with the leg description. There is a launch file called `visual.launch` in order to verify the resulting urdf in *rviz*.

The `Leg` library contains the state (joint angles) `q`, joint velocities `qt`, the end effector position `Xe`, the Geometric Jacobian `Jv` and finally the number and the solutions from the last inverse kinematics. There are a callback method (`PoseCallback`) to log the state of the robot, methods to get the protected attributes (e.g. `getSols`, `getState`), methods for kinematic calculations (eg `IK`, `DK`) and methods to calculate the true distance between poses[11].

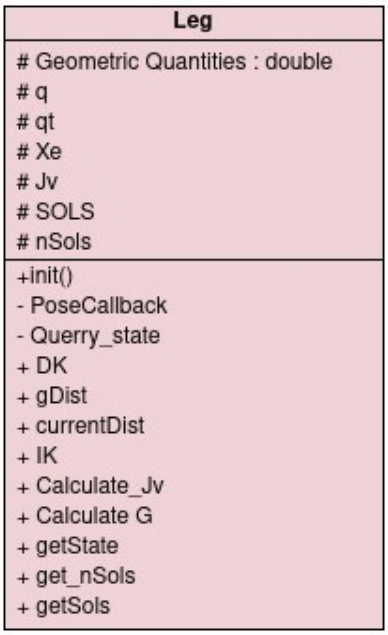


Figure 7: UML diagramm of `Leg` class.

4.2 `Leg_control` package

This package contains the leg task controllers (Position Controller, Trajectory Controller, Effort Controller) and a high level controller (HLC) in order to control the leg.

This package introduces the following nodes:

- *legHLC*: This node acts as a high-level controller, switching between task controllers. The source code is in `legHLC.cpp`.
- *legContact*: This node subscribes to the *end_effector_collision* topic and gets the foot contact forces. Then, it transforms these forces in the world frame[12] and publishes the to *contact_force* topic. The source code is in `legContact.cpp`.

This package introduces the following services for the *legHLC* node:

[11] As the second joint is continuous there are two distances between two angles. A naive subtraction $\Delta\theta = \theta_{to} - \theta_{from}$ may give large angular distances.

[12] Normally, it is possible to specify the frame of reference in the gazebo_ros_bumper plugin, but it didn't seem to work. So this node is a work around.

- *ControllerSelector*: Select which low level controller to activate at a given moment. There is a Custom *.srv* file, named `ControllerSelector.srv` defining the service.

  Usage: The user selects which task controller to activate by sending a number between 1 and 3. In the current implementation, the following correspondence takes place: 1: Position Controller; 2: Trajectory Controller; 3: Effort Controller.

- *GoalSet*: Set a target for the position controller. There is a Custom *.srv* file, named `pos.srv` defining the service. .

  Usage: The first input to the service call is a Boolean, to specify whether the target is in joint-angle space (False) or in Cartesian space (True). The other three inputs are either the desired joint angles $[q_{1,d},\ q_{2,d},\ q_{3,d}]$ or the coordinates of the desired position in the world frame $[x_d,\ y_d,\ z_d]$ depending on the first input.

- *SetEllipse*: Set ellipse parameters for the trajectory controller. There is a Custom *.srv* file, named `ellipse.srv` defining the service.

  Usage: The inputs are the ellipse parameters ($a,\ b,\ DX,\ DY,\ d\theta$) for an ellipse at the YZ plane, which is centered at ${}^{W}P = [LB0 + L12,\ DY,\ DZ]$. The parameterization used is the polar form of the ellipse, where the radius $r(\theta)$ is given by the equation (37).

$$r(\theta) = \frac{ab}{\sqrt{a^2 sin^2(\theta - d\theta) + b^2 cos^2(\theta - d\theta)}} \tag{37}$$

  Along with the ellipse parameters, the user specifies the period of the ellipse and the times that the trajectory will be repeated.

- *SetEffort*: Set desired end-effector wrench for the effort controller. There is a Custom *.srv* file, named `wrench.srv` defining the service.

  Usage: The inputs are the desired exerted force components of the end effector specified in the world frame of reference: $[{}^{W}F_{x,d},\ {}^{W}F_{y,d},\ {}^{W}F_{z,d}]$.

The launch file `LegFramework.launch` is the default launch file of the package and opens up the *rqt_gui* node (with a custom perspective), the gazebo simulator and the custom-made nodes that act as the leg controller.

Figure 8 is a concise UML diagram of the implemented controllers in order to showcase the structure of the framework.

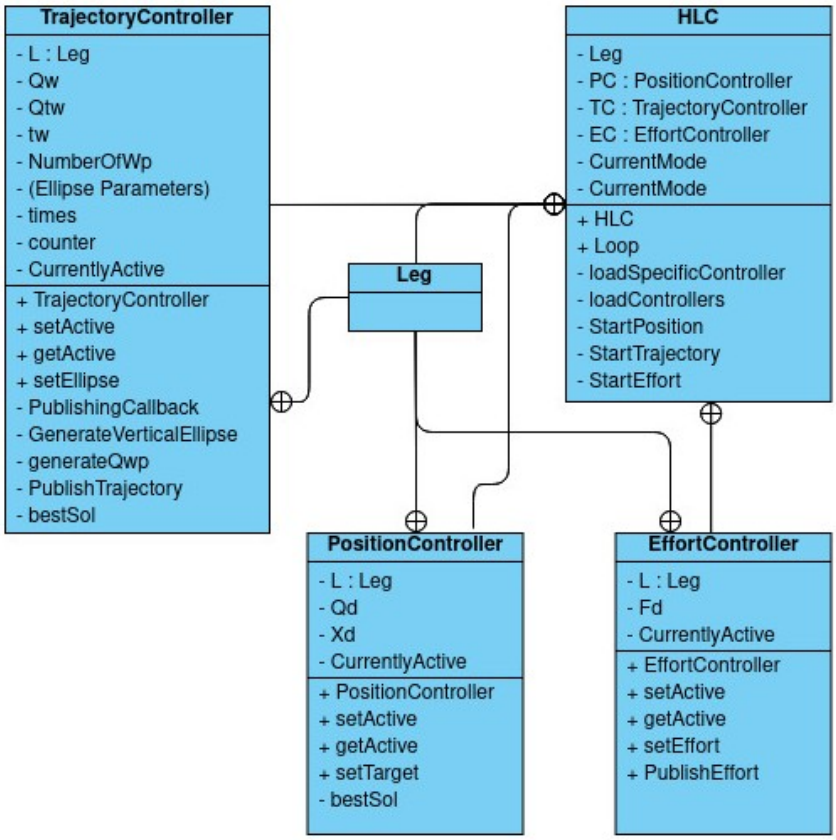


Figure 8: UML diagrams of the controller classes. (The task controllers have a reference to the same instance of `Leg`).

4.2.1 Framework GUI and Documentation

For the GUI, *rqt_gui*[13] is used. There are two *Service Caller* windows, as calling services is frequent. Also, there is *Dynamic Reconfigure* window, to change both the parameters of the low level controllers and the physics simulation parameters. Along with these functionalities, there is a *rviz* window, to visualize the leg, a *multiplot* window and an *rqt_plot* window for user specified plots.

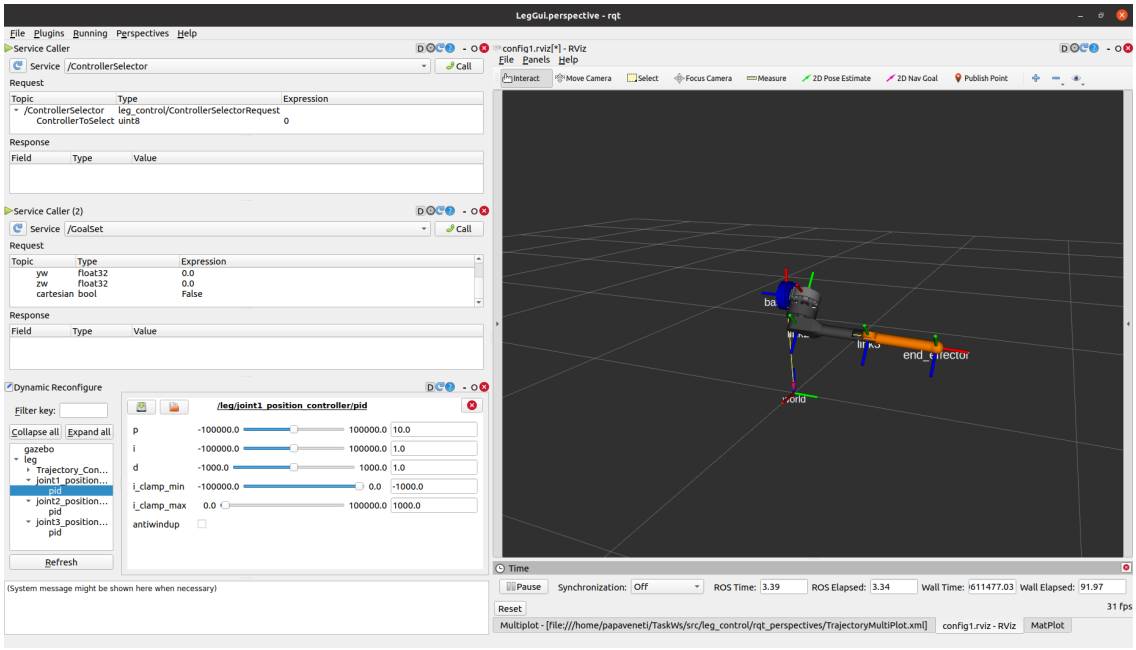


Figure 9: Gazebo Simulation Framework GUI.

Also, it must be noted that the whole framework has been documented[14]. In figure 10 the documentation for the method `generateQwp` of the *TrajectoryController* is presented as an example.

```
/** @brief Generate angular position waypoints from cartesian waypoints
 *
 * @note Checks if a point is reachable (solution to IK exists), then solution with the shortest distance in joint angle
 * space. This is **NOT OPTIMAL**. As for velocities, the mean velocity for each segment is calculated. Then for each setpoint,
 * if the velocity of the previous and next segment are in the same direction, their mean value is given. If the velocities point
 * to the opposite direction, the desired velocity is set to 0.The generation can fail because: 1) a waypoint is unreachable,
 * 2) timestamp is unreasonable
 *
 * @param Xwp A 3xN `Eigen::matrix` with all the cartesian waypoints. Each column is a point in 3D space.
 * @param twp A Nx1 `Eigen::Vector` with the timestamps for each waypoint.
 *
 * @returns true if the waypoint generation is successful, false otherwise.
 */
bool generateQwp( Eigen::MatrixXd Xwp, Eigen::VectorXd twp){…
```

Figure 10: Ros framework documentation example.

[13] `http://wiki.ros.org/rqt_gui`
[14] The documentation is *Doxygen* compatible.

5. Simscape Simulation Framework

This section gives information about the structure of the matlab - simscape simulation framework.

The matlab - simscape simulation framework is made up of a collection of classes that model the controllers and the robot, a class that handles the simulation and the simscape simulation.

The user can set up a series of tasks for the leg and then simulate them. This is done by using methods from the controller classes directly. It is possible to run only the custom-made simulation (using the analytical equations that are analysed in section 6), only the simscape simulation, or both.

A basic matlab script to set-up the simulation is included in the appendix, in section 8.5.

All the code has been documented, so the user can understand how it works using the command `doc`. For example, the documentation for the position controller

```
1 doc Pos
```

gives the following output:

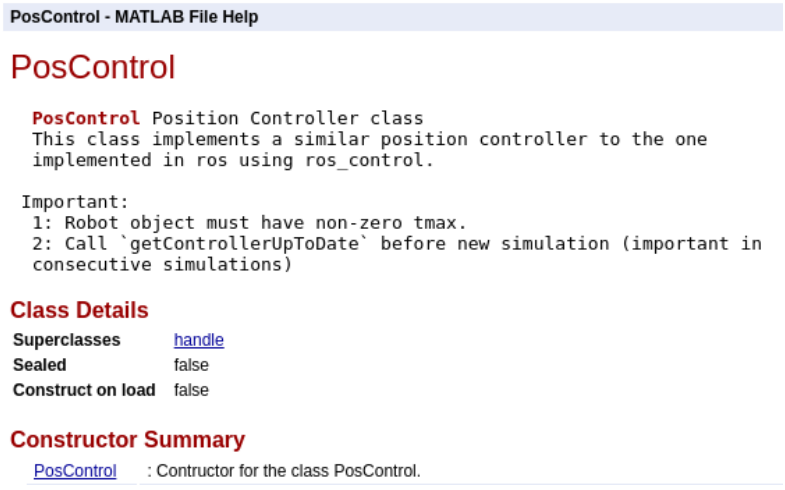


Figure 11: Simscape framework documentation.

5.1 Framework structure

The structure of the framework is presented in the UML diagram in figure 12.

Firstly, the `robot` abstract class is the parent class for all robots to be modeled[15] and it mainly contains abstract methods for the robot kinematics. `robot` is inherited by the `legRobot` class, that models the Argos's leg. The `robot` class has the `sim` method, that implements the robot forward dynamics in order to run simulations using the matlab ode solvers.

There are also controller classes that implement similar controllers to the *ros_control* ones in order to compare the two frameworks. The controller classes have methods to set up new targets (reference points), methods to set the state of the controller (current errors, internal timers) that are useful for the simulation handler class, and the `control_sim` method in order to run simulations using the ode solvers. All the controller classes share the same instance of the `legRobot` object, so the state of the robot is synchronized[16]. Also, the trajectory controller class (`trajControl`) has a trajectory generation class (`trajectoryGen`) that creates polynomial trajectories in a manner similar to the *joint_trajectory_controller* of *ros_control*.

The simulation handler class is the `simulation_preparation` class. This class contains all the controllers and the same instance of the `legRobot` object. It contains methods to set up the simulation parameters and run the simulations.

[15]To set up the right simulations, a model for a two-link manipulator was developed, so the robot parent class made the transition from the two-link robot to the leg easier.

[16]The same result is achieved using references in C++. In matlab classes, the `robot` parent class inherits the `handle` class.

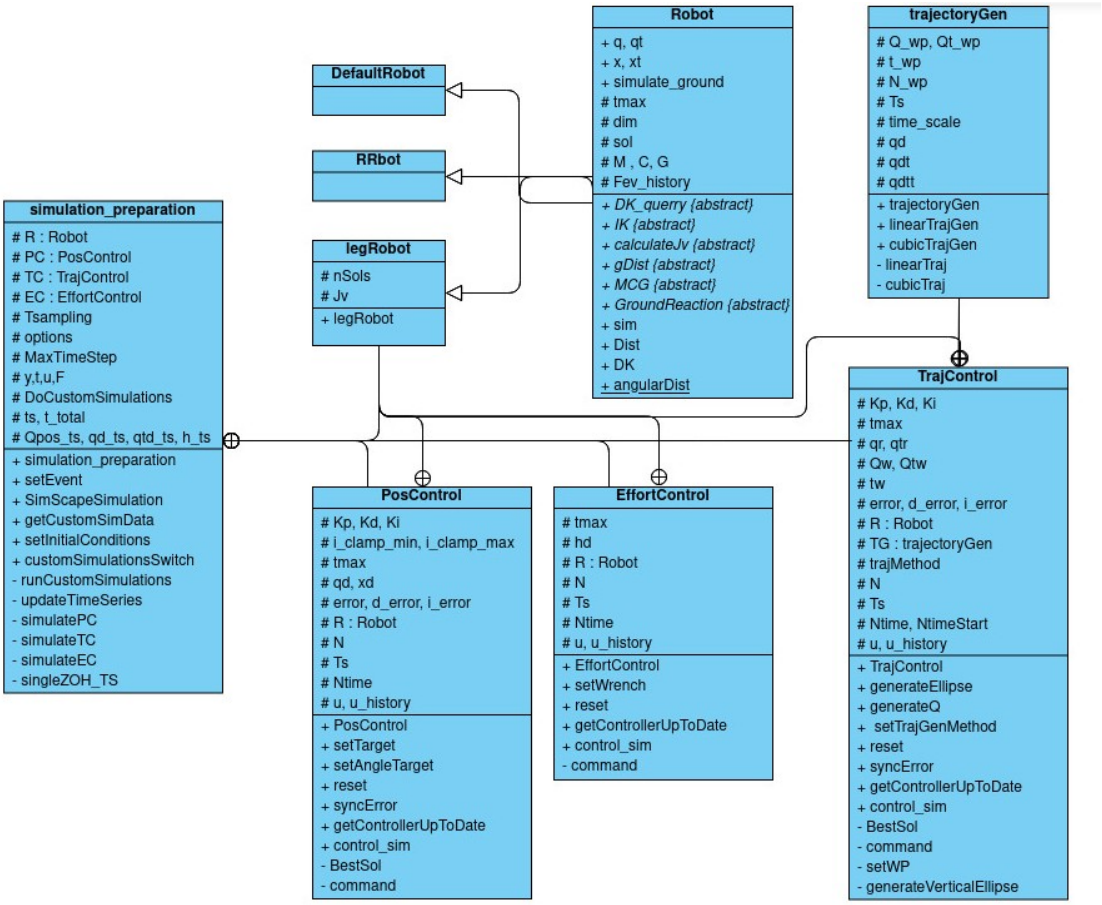


Figure 12: UML diagram for the matlab-Simscape simulation framework.

5.2 Simscape simulation

The simscape simulation is presented in figure 13. The references and the parameters of the controllers are imported from the matlab workspace. That is why, **running a script similar to 1 is a prerequisite for running a simulation using this framework.**

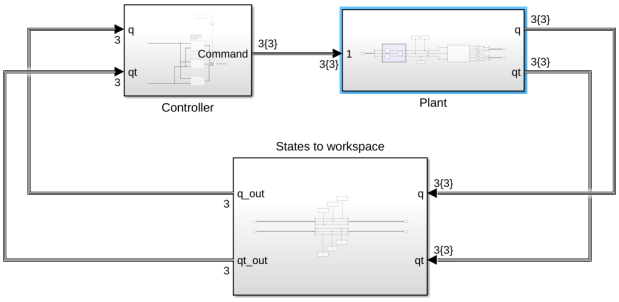
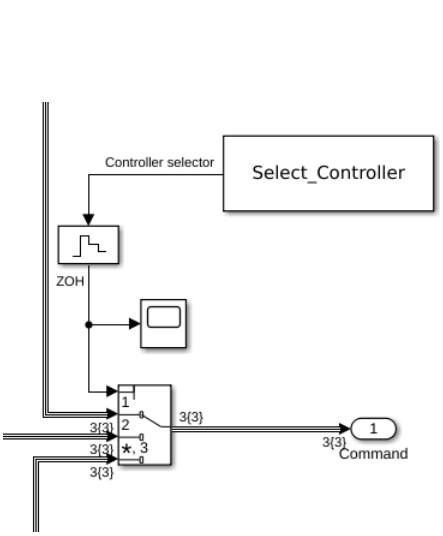


Figure 13: Simscape simulation: Overview.

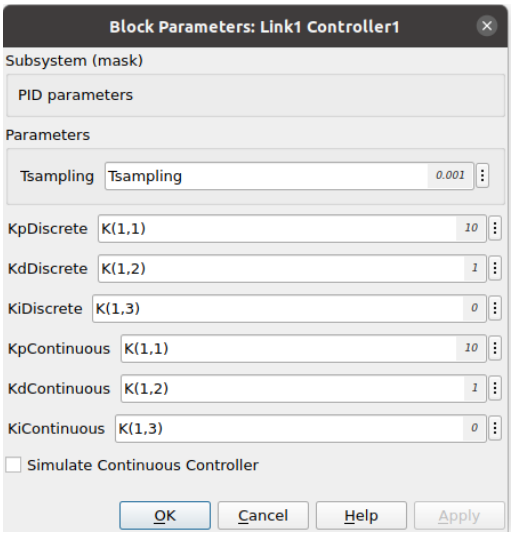
The *plant* was created by importing the urdf, setting the motor limits and the ground interaction model. All the states and contact forces are logged and exported to the workspace after the simulation. This is done in the *States to workspace* block.

The *controller* block, which is presented in figure 14a, has three separate controllers and a multiport switch that enables the command of the currently selected controller to drive the system. Also, three separate subsystem, which are included using the subsystem reference block[17], were created. An example is presented in figure 14b. The custom subsystems are:

- *PositionLinkController*,
- *TrajectoryLinkController*,
- *shortest_angle_diff*,



(a) Controller block.



(b) Custom Subsystem.

Figure 14: Simscape simulation components.

[17] https://www.mathworks.com/help/simulink/ug/referenced-subsystem-1.html

6. Analytical Modeling

6.1 General Methodology

For the derivation of the analytical dynamical equations, the Euler-Lagrange method was used. The process is as follows:

1. Calculate the Kinetic energy (T) of the system.
2. Calculate the Potential energy (U) of the system.
3. Calculate the Lagrangian: $\mathcal{L} = T - U$. (`leg_eom.m`)
4. Find the following derivatives (`EL_derivatives.m`):

$$\frac{\partial}{\partial t}\left(\frac{\partial(\mathcal{L})}{\partial \dot{\mathbf{q}}}\right)$$
$$\frac{\partial(\mathcal{L})}{\partial \mathbf{q}}$$

where $\mathbf{q}$ is the vector of generalized coordinates. Here $\mathbf{q}$ is the joint angle vector. The equations of motion are given by:

$$\frac{\partial}{\partial t}\left(\frac{\partial(\mathcal{L})}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial(\mathcal{L})}{\partial \mathbf{q}} = \mathbf{f} \tag{38}$$

where $\mathbf{f}$ is the generalized force. It contains:

- the coulomb friction: $\mathbf{F_s}\ sign(\dot{\mathbf{q}}) = diag\{F_{s,1}, ..., F_{s,i}, ..\} \cdot sign(\dot{\mathbf{q}})$
- damping forces: $\mathbf{D}\ \dot{\mathbf{q}} = diag\{D_1, ..., D_i, ..\} \cdot \dot{\mathbf{q}}$
- forces due to interactions with the environment: $\mathbf{J}^T(\mathbf{q})\mathbf{h}_e$, where $\mathbf{J}^T$ is the geometric Jacobian and $\mathbf{h}_e$ is the wrench vector from the end-effector to the environment
- motor inputs: $\boldsymbol{\tau}$

Usually, these equations are more useful in a matrix form, that takes the following form:

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{D}\dot{\mathbf{q}} + \mathbf{F}_s sign(\dot{\mathbf{q}}) + \mathbf{J}^T(\mathbf{q})\mathbf{h}_e = \boldsymbol{\tau} \tag{39}$$

5. Collect the terms to get the matrix form of the equations. (`EL_collect.m`)

Having the dynamics in the form of equation (39), one can easily simulate the system using equation (40). (This form is compatible with the matlab solvers):

$$\frac{d}{dt}\begin{bmatrix}\mathbf{q} \\ \dot{\mathbf{q}}\end{bmatrix} = \begin{bmatrix}\dot{\mathbf{q}} \\ \mathbf{B}(\mathbf{q})^{-1} \cdot (\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) - \mathbf{D}\dot{\mathbf{q}} - \mathbf{F}_s sign(\dot{\mathbf{q}}) - \mathbf{J}^T(\mathbf{q})\mathbf{h}_e)\end{bmatrix} \tag{40}$$

Regarding the friction and damping forces, usually only the joint static and viscous friction[18] is taken in account in robotics, and thus these forces can be calculated if the matrices $\mathbf{F}_s$ and $\mathbf{D}$ are defined. Both in gazebo and simscape, one can directly define these matrices. These forces will be ignored, except if stated otherwise. The interaction force is zero as long as there are no interactions, such as when the robot is moving in free space. In the static task, more information will be presented. Thus, initially, one has to find the $\mathbf{B}, \mathbf{C}, \mathbf{G}$ matrices.

[18] Friction forces that have to do with the end effector and the environment are modelled as an external wrench $\mathbf{h}_e$.

6.2 Calculating the Lagrangian

The first step is the calculation of the Lagrangian. It was done using the program `leg_eom.m`. This script is responsible for:

- Defining the transformation matrices ${}^{i-1}\mathbf{T}_i$ (calling `Tsimplify.m`).
- Load the geometric and inertial quantities of the leg.
- Calculate the positions of the coordinate frame i with respect to the j frame[19].

$$ {}^{j}\mathbf{P}_i = {}^{j}\mathbf{J}_i(1:3,\ 4) = \begin{bmatrix}\mathbf{I}_{3x3} & \mathbf{0}_{3x1}\end{bmatrix} {}^{j}\mathbf{T}_i \begin{bmatrix}\mathbf{0}_{3x1} \\ 1\end{bmatrix} \tag{41} $$

- Calculate the orientation of the coordinate frame i with respect to another j frame

$$ {}^{j}\mathbf{R}_i = {}^{j}\mathbf{T}_i(1:3,\ 1:3) = \begin{bmatrix}\mathbf{I}_{3x3} & \mathbf{0}_{3x1}\end{bmatrix} {}^{j}\mathbf{T}_i \begin{bmatrix}\mathbf{I}_{3x3} \\ \mathbf{0}_{1x3}\end{bmatrix} \tag{42} $$

- Calculate the positions of the center of mass of link i with respect to the 0 frame. From solidworks, we have ${}^{i}\mathbf{r}_{c,i}$ (the position of the center of mass of link i with respect to the coordinate frame of the link i).

$$ {}^{0}\mathbf{x}_{c,i} = {}^{0}\mathbf{P}_i + {}^{0}\mathbf{R}_i\ {}^{i}\mathbf{r}_{c,i} \tag{43} $$

- Calculate the angular velocities of the coordinate frame i with respect to the 0 frame. These are the same as the angular velocities of the center of mass of the link i with respect to the 0 frame.

$$ {}^{0}\boldsymbol{\omega}_i = {}^{0}\boldsymbol{\omega}_{i-1} + {}^{0}\mathbf{R}_i\ [0,\ 0,\ \dot{q}_i]^T \tag{44} $$

- Calculate the linear velocities of the coordinate frame i with respect to the 0 frame.

$$ {}^{0}\mathbf{u}_i = {}^{0}\mathbf{u}_{i-1} + \underbrace{{}^{0}\mathbf{u}_i^{i-1}}_{=\mathbf{0}} + [{}^{0}\boldsymbol{\omega}_{i-1}]^x\ {}^{0}\mathbf{R}_{i-1}\ {}^{i-1}\mathbf{P}_i \tag{45} $$

- Calculate the velocities of the center of mass of link i with respect to the 0 frame.

$$ {}^{0}\mathbf{u}_{c,i} = {}^{0}\mathbf{u}_i + \underbrace{{}^{0}\mathbf{u}_{c,i}^{i}}_{=0} + [{}^{0}\boldsymbol{\omega}_i]^x\ {}^{0}\mathbf{R}_i\ {}^{i}\mathbf{r}_{c,i} \tag{46} $$

Having calculated these quantities, it is easy to write the Kinetic energy as:

$$ T = \sum_{i=1}^{3} \left[\frac{1}{2} m_i\ {}^{0}\mathbf{u}_{c,i}^T\ {}^{0}\mathbf{u}_{c,i} + \frac{1}{2}\ {}^{0}\boldsymbol{\omega}_i^T\ {}^{0}\mathbf{R}_i\ \mathbf{I}_i\ ({}^{0}\mathbf{R}_i^T)\ {}^{0}\boldsymbol{\omega}_i \right] \tag{47} $$

The potential energy is :

$$ U = -\sum_{i=1}^{3} m_i \mathbf{g}_0^T\ {}^{0}\mathbf{x}_{c,i} \tag{48} $$

where $\mathbf{g}_0 = [-g,\ 0,\ 0]$. Looking at 1b, gravity is indeed looking in the negative x-axis of the 0 frame.

The Lagrangian is the $\mathcal{L} = T - U$

[19] Indexing via matrix multiplication was done because `symfun` objects cannot be indexed using parenthesis indexing

6.3 Getting the B,C,G matrices

Getting the derivatives of the Lagrangian can be done using the differentiation functions of the `symbolic` package of matlab and takes place in the `EL_derivatives` script. `EL_collect` is responsible for collecting the terms in returning the B, C, G matrices required for the direct dynamics simulation using equation (40).

Each equation concerning the i-th DOF has the following form:

$$eq_i = \sum_j b_{i,j}(\mathbf{q})\ddot{q}_j + \sum_j c_{i,j}(\mathbf{q}, \dot{\mathbf{q}})\dot{q}_j + G_i(\mathbf{q}) \tag{49}$$

To get the matrices, the following process is followed for each degree of freedom i:

1. Get $b_{i,j}$ from the coefficients of $\ddot{q}_j$ in the equation of the i-th degree of freedom. This can be achieved as the polynomial of $\ddot{q}_j$, $p(\ddot{q}_j)$ has degree $deg(p(\ddot{q}_j)) \leq 1$. The latter fact can be observed from (49).

2. Update the equation:
$$eq_i^{new,1} = eq_i - \sum_j b_{i,j}(\mathbf{q})\ddot{q}_j$$

3. Get $c_{i,j}$ from the coefficients of $\dot{q}_j$ in the **new** equation of the i-th degree of freedom $eq_i^{new,1}$. The polynomial of $\dot{q}_j$, $p(\dot{q}_j)$ has degree $deg(p(\dot{q}_j)) \leq 2$. This fact can be observed from (49). The polynomial is $p(\dot{q}_j) = p_0 + p_1\dot{q}_j + p_2\dot{q}_j^2$. So $c_{i,j}$ is obtained by :
$$c_{i,j} = p_1 + p_2\dot{q}_j$$
Generally, there are infinite[20] choices for the C matrix.

4. Update the equation:
$$eq_i^{new,2} = eq_i^{new,1} - \sum_j C_{i,j}(\mathbf{q}, \dot{\mathbf{q}})\dot{q}_j$$

5. The rest are the gravitational terms: $G_i(\mathbf{q}) = eq_i^{new,2}$

Getting the coefficients is done by the `coeffs` command of the `symbolic toolbox`. Getting the coefficients of the full polynomial for each degree of freedom is done by providing the *'All'* argument.

6.3.1 Validating the methodology

The scripts that were developed carry out all the hard calculations and factorizations, and abstract the methodology. To debug and validate them, the scripts were used to derive the equations of motions of simple 1 or 2 degree of freedom manipulators that were studied in the robotics course[21]. This was done in the `EL_validation.m` script.

6.4 Validating the Euler Lagrange model

Having ensured that the above programs behave as intended (concerning the calculation of the relevant matrices), the $B(\mathbf{q})$, $C(\mathbf{q}, \dot{\mathbf{q}})$, $G(\mathbf{q})$ matrices for the leg were extracted, using the same geometric and inertial properties that were passed to the urdf. To validate the whole process, some basic simulations were conducted. An indicative simulation is presented in figure 15 below. In this simulation, gravity was neglected, and a sinusoidal input was commanded in every joint that is given by $u_i = 0.01sin(6\pi t)$ $[Nm]$. The purpose was to **check that matrix extraction**

[20] Siciliano [2], section 7.2.1. For example, a term $k\dot{q}_1\dot{q}_2$ in the i-th equation can be distributed in the **C** matrix as follows:

$$c_{i,1} += \lambda\dot{q}_2$$
$$c_{i,2} += (k - \lambda)\dot{q}_1$$

where $\lambda \in \mathbb{R}$.

[21] `http://nereus.mech.ntua.gr/courses/robotics/robotics_pdf/kk/Dynamics.pdf`

process. The solver settings are presented in table 6. It can be observed that the simulations using the derived equations of motion are identical with the simscape simulations. This is because similar solver settings were used.

Table 6: Leg initial simulations solver settings for matlab and simscape.

Simulation	**Abs Tol**	**Rel Tol**	**solver**	**max step**
matlab simulation	1e-6	1e-8	ode45	$5e-2$
simscape	1e-8	1e-6	ode45	$5e-2$

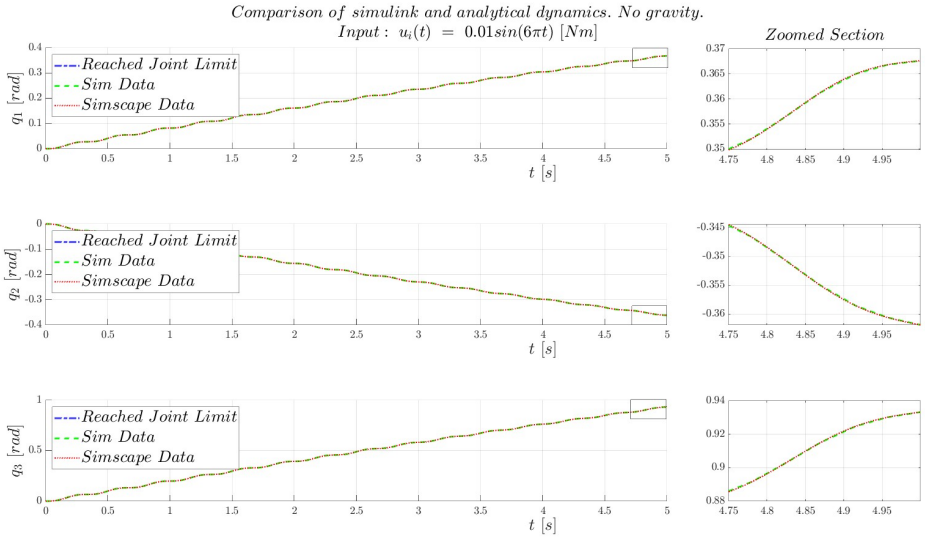
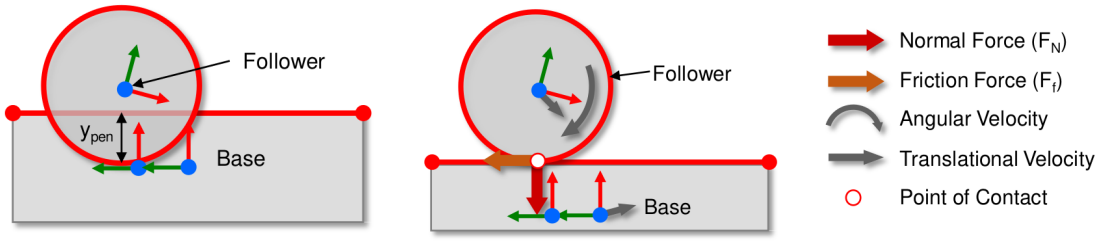


Figure 15: Comparison of angle position histories for the three joints for the simulation of an actuated leg without the presence of gravity with initial conditions $\mathbf{q} = [0, 0, 0]$ and $\dot{\mathbf{q}} = [0, 0, 0]$ in matlab and simscape. In the right side, there is a focus plot in order to better showcase the deviation.

6.5 Contact Force Modeling

For the analytical modeling, it was attempted to create a contact model similar[22] to the one used previously in Iro's thesis [1].



(a) Normal force.

(b) Friction force.

Figure 16: Simscape contact model.

The previously used model is a "sphere to plane" contact model[23] that comes in the simscape contact library. In this model, the contact normal force is given by equation (50):

$$F_n = \begin{cases} K_p y_{pen} + K_d \dot{y}_{pen} & y_{pen} > 0, \dot{y}_{pen} > 0 \\ K_p y_{pen} & y_{pen} > 0, \dot{y}_{pen} \leq 0 \\ 0 & y_{pen} < 0 \end{cases} \tag{50}$$

where K_p and K_d are the collision properties defined in 5 and y_{pen} is the penetration of the point of contact (poc), as seen in figure 16a. To model the contact friction, a stick-slip continuous friction model was used, as in Iro's thesis [1]. The norm of the contact friction force is given by the equation (51):

$$F_f = \mu F_n \tag{51}$$

where:

$$\mu = \begin{cases} v_{poc} \cdot \frac{\mu_s}{v_{th}} & v_{poc} < v_{th} \\ \mu_s - (v_{poc} - v_{th}) \cdot \frac{\mu_s - \mu_k}{0.5 v_{th}} & v_{th} \leq v_{poc} < 1.5 v_{th} \\ \mu_k & v_{poc} \geq 1.5 v_{th} \end{cases} \tag{52}$$

The direction opposes the velocity of the point of contact. As the ground is the plane for $z = 0$:

$$\vec{v}_{dir} = \frac{[\,{}^W v_x,\ {}^W v_y]}{\sqrt{{}^W v_x^2 + {}^W v_y^2}}$$

and thus the friction force is:

$$\vec{F}_f = F_f \cdot \begin{bmatrix} -\vec{v}_{dir} \\ 0 \end{bmatrix}$$

The graphical interpretation of (52) is shown in figure 17.

Note: In the current implementation, the point of contact coincides with the end effector. This creates an error in the modeling, as the true point of contact is in the radius of the foot. This can be seen in figure 18.

[22] The model is similar and not exactly the same because the simulation results were deemed acceptable using the current-inexact model. The difference is that in simulink the velocity of the point of contact (poc) used in the contact model is the true poc velocity while the current implementation uses the end-effector velocity. This is explained in detail in the current section.

[23] `https://www.mathworks.com/matlabcentral/fileexchange/47417-simscape-multibody-contact-forces-library`

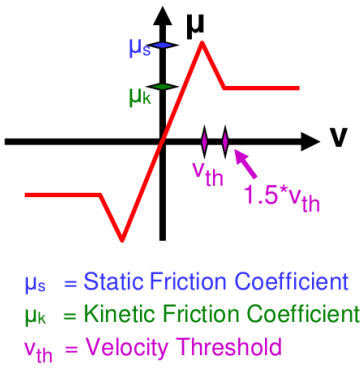


Figure 17: Stick-slip friction model.

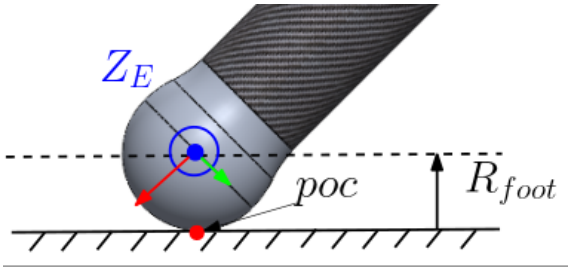


Figure 18: Difference between true point of contact of foot and end effector coordinate frame origin.

6.6 Controller Details

6.6.1 PID controller

Both PID controllers from `ros_control`[24] and simulink [25] calculate the command using the following algorithm:

1. $e_{position} = q_{desired} - q$
2. $\dot{e}_{position} = (e_{position} - e_{position,last})/T_s$, where T_s is the sampling rate.
3. $\int e_{position} = e_{position} \cdot T_s + \left(\int e_{position}\right)_{last}$
4. $u = K_P e_{position} + K_D \dot{e}_{position} + K_I \int e_{position}$

Indeed, the *z-transform* of the above torque command is[26]:

$$\frac{U(z)}{E(z)} = K_P + K_D \frac{1}{T_s} \frac{z-1}{z} + K_I T_s \frac{1}{z-1} \tag{53}$$

6.6.2 Angular Distance

The PID controller calculated the joint angular error ($e_{position} = q_{desired} - q$), which is the angular distance between the desired joint angle and the current one. This distance[27] is calculated as follows (as $q_i \notin \mathbb{R}$ but $q_i \in \mathcal{S}$):

$$q_{temp} = q_{final} - q_{start} + \pi \tag{54}$$

$$q_{distance} = mod(q_{temp}, 2\pi) - \pi \tag{55}$$

The correctness of this calculation is evident throught the following example, which is presented in figure 19. If the initial angular position of a joint is $q_{start} \approx \pi - d\theta_1$ and the desired angular position is $q_{goal} \approx \pi + d\theta_2$, then the numeric difference is $e_{position} = q_{goal} - q_{start} = 2\pi - d\theta_1 - d\theta_2$. This, assuming $d\theta_i$ is small, results in big error value and thus a big torque command. The joint will be rotated as the blue path in figure 19 indicates. However, it is evident that the optimal path is the black path of the same figure. Indeed, using (54) and (55):

$$q_{temp} = 3\pi - d\theta_1 - d\theta_2$$

$$q_{distance} = -d\theta_1 - d\theta_2$$

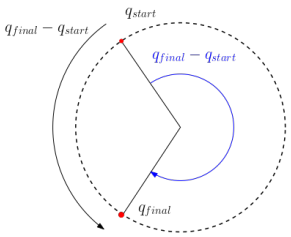


Figure 19: Angle distance calculation example.

That way, large values of errors to describe angular distances are avoided, resulting in **smoother movement** and **smaller torque commands**.

[24] `http://docs.ros.org/en/indigo/api/control_toolbox/html/classcontrol__toolbox_1_1Pid.html`

[25] `https://www.mathworks.com/help/simulink/slref/pidcontroller.html`, using an unfiltered derivative and forward euler integration method

[26] Which is the exact same formula in simscape when using an unfiltered derivative and forward euler integration method

[27] This is important for the second joint.

6.6.3 Trajectory Generation

The trajectory generation implements the following algorithm (`generateQwp` method in the ros framework, `generateQ` in the matlab framework). The input is a set of cartesian waypoints (a $3 \times N$ vector containing positions in the world frame) and the timestamp corresponding to each waypoint.

1. Starting from the first waypoint:
   (a) Find the solutions to the inverse kinematic (IK) problem. **If there are none, the trajectory fails.**
   (b) Select the best solution from IK, using equation (56).

$$\mathbf{q}_i = argmin(|| \ f_{dist}(\mathbf{q}_{sol}, \mathbf{q}_{i-1}) \ ||) \tag{56}$$

   where $\mathbf{q}_i$ is the best solution for the i-th waypoint, $\mathbf{q}_{sol}$ are the current solutions of the IK problem and f_{dist} is the joint angular distance function[28] as described in section 6.6.2. For the first waypoint, the algorithm the "*previous*" solution is the current state of the robot.

   After doing the above for all the waypoints, a $3 \times N$ matrix with joint angle waypoints has been created.

2. The mean velocity of the i segment (from q_{i-1} to q_i) is calculated:

$$\bar{\omega}_i = \frac{q_i - q_{i-1}}{dt_i}$$

3. If the mean velocity of two consecutive segments has the same sign (the same direction), then the algorithm sets the desired velocity of the middle waypoint as the mean of the mean velocities:

$$\dot{q}_{d,i} = (\bar{\omega}_i + \bar{\omega}_{i+1})/2$$

   Otherwise, the desired velocity of the waypoint is set to zero.

6.6.4 Trajectory Controller

Having the generated trajectories $\mathbf{q}_d(t)$, $\dot{\mathbf{q}}_d(t)$ the controller is (in a discrete form) :

$$\mathbf{u} = \mathbf{K}_P(\mathbf{q}_d(t) - \mathbf{q}) + \mathbf{K}_D(\dot{\mathbf{q}}_d(t) - \dot{\mathbf{q}}) + \mathbf{K}_I \int (\mathbf{q}_d(t) - \mathbf{q})dt \tag{57}$$

6.6.5 Effort Controller

The effort controller is:

$$\boldsymbol{\tau} = \mathbf{J}_v^T \mathbf{h}_d + \mathbf{G}(\mathbf{q}) \tag{58}$$

and thus, the system in steady state , where $\dot{\mathbf{q}} = 0$ and $\ddot{\mathbf{q}} = 0$ will compensate the gravitational force and exert the desired wrench in the environment.

[28] It returns a vector of distances for each joint.

7. Comparing the Frameworks

In this section, the frameworks are compared in four tasks:

- Position Control task
- Trajectory Control task (dynamic task)
- Effort Control task (static task)
- Combined task

The system geometric, collision properties are the ones presented in section 2 in tables 1, 3,5. The starting position of the leg for all[29] the simulations is the zero position of the leg, which is for $\mathbf{q} = [0, 0, 0]^T$, and is shown in figure 20:

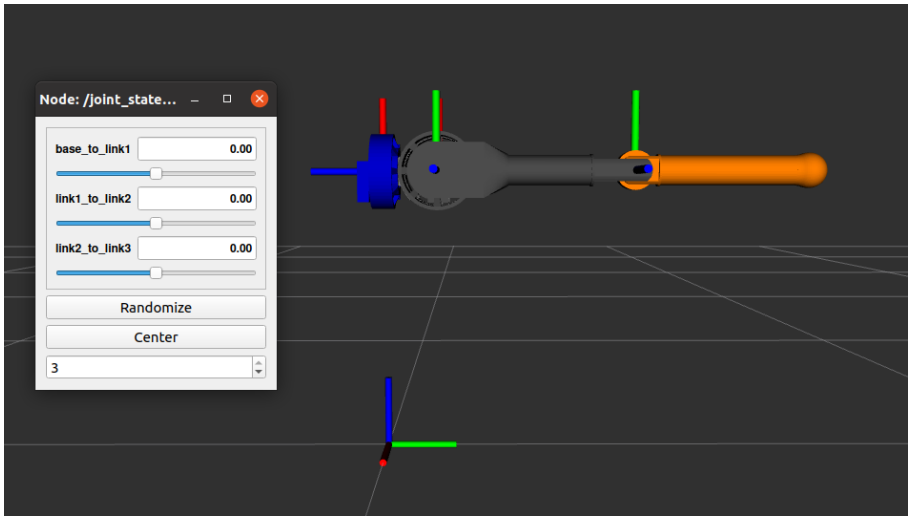


Figure 20: Zero position of the leg.

The solver settings[30] are presented in tables 7 and 8. A more refined *max_step* was chosen for the custom simulation because the simulation timestamps must be close to the moment that a new commands is issued by the controller.[31]

Table 7: Leg simulations solver settings for matlab and simscape.

Simulation	**Abs Tol**	**Rel Tol**	**solver**	**max step**
matlab simulation	1e-6	1e-8	ode45	$1e-4$
simscape	1e-8	1e-6	ode45	$2e-4$

Table 8: Leg simulations solver settings for gazebo.

Simulation	**solver**	**max_step**	**erp**
gazebo	world	0.001	Default value: 0.2

[29] Apart from the effort control simulation.

[30] The erp solver setting for ODE is presented here: `https://ode.org/ode-latest-userguide.html#sec_3_7_0`

[31] A controller with an update rate of $1ms$, should update the commands at $t_{update} = 0,\ 1,\ 2,\ \ldots\ [ms]$. If the simulation max step is $1ms$, then the solution timesteps could be $t = 0,\ 0.5,\ 1.5\ \ldots\ [ms]$, due to the internal workings of matlab ode solvers, that refine the timestep to achieve the desired tolerances. In the current implementation, this shift would cause the controller to update half a millisecond later. This delay results in large errors. By refining the solution *max_step* parameter, these errors decrease.

7.1 Position Control Comparison

For this simulation, the initial conditions are: $\mathbf{q} = [0, 0, 0]^T$, $\dot{\mathbf{q}} = [0, 0, 0]^T$ and desired positions: $\mathbf{q}_d = [0, 0, 0]^T$. In the following figures (21, 22, 23, 24,25, 26) the results from the three simulations are compared.

Figure 21 shows that the simulations are very close to each other. However, big steady state errors are observed. This is because the gains are set randomly at $K_p = 10,\ K_d = 1$.

Figure 22, is a deviation plot. The initial difference is due to synchronization errors between the gazebo and matlab simulations and ros intrinsic[32] delays[33]. The maximum simulation error percentage[34] is of the errors is around 2.5%, for the 3rd joint. It can be observed, that matlab and simulink simulation errors are less significant.

In figure 23 the joint velocities are compared for completeness. Figure 24 shows the torque histories of the actuators for the position control task and figure 25 focuses on the transient response. Figure 26 shows the deviation plot of the torque commands. Again, there is an initial greater deviation, which exponentially decays. The error percentage between the simulations is less than 4%.

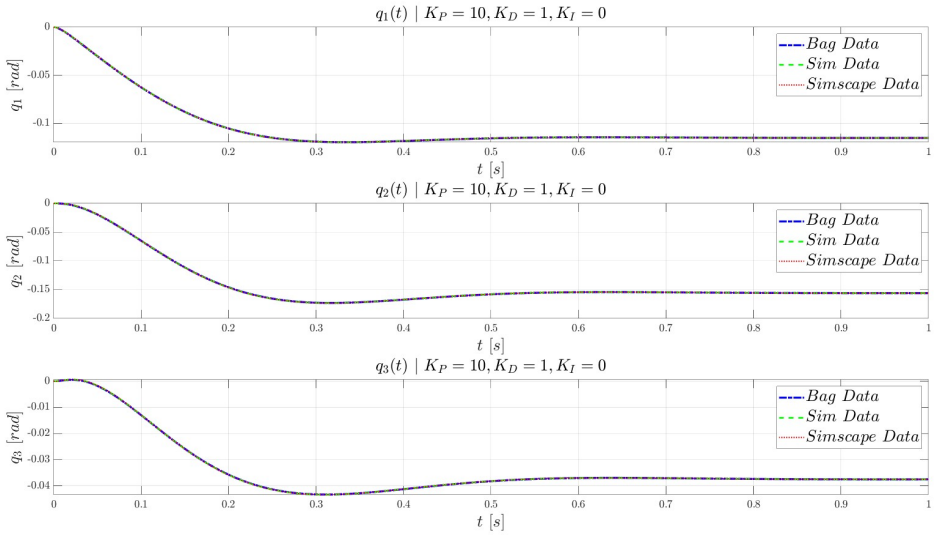


Figure 21: Comparison of joint position histories for the three joints for the simulation of the Position Controller in matlab, simscape and gazebo.

[32] E.G. service call delays. A delay in a controller command in the start of the simulation creates an error that propagates in the joints. Joint 1 is affected more notably, as it is the first joint of the leg.

[33] The exponential decay of the error indeed supports this explanation. As seen when simulating observers in control systems, small differences in initial conditions decay exponentially. The same is true for synchronization errors.

[34] Where the simulation error percentage for each joint is:

$$\%\ simulation\ error = \left| \frac{(max\ error_i)}{(max\ amplitude_i)} \right|$$

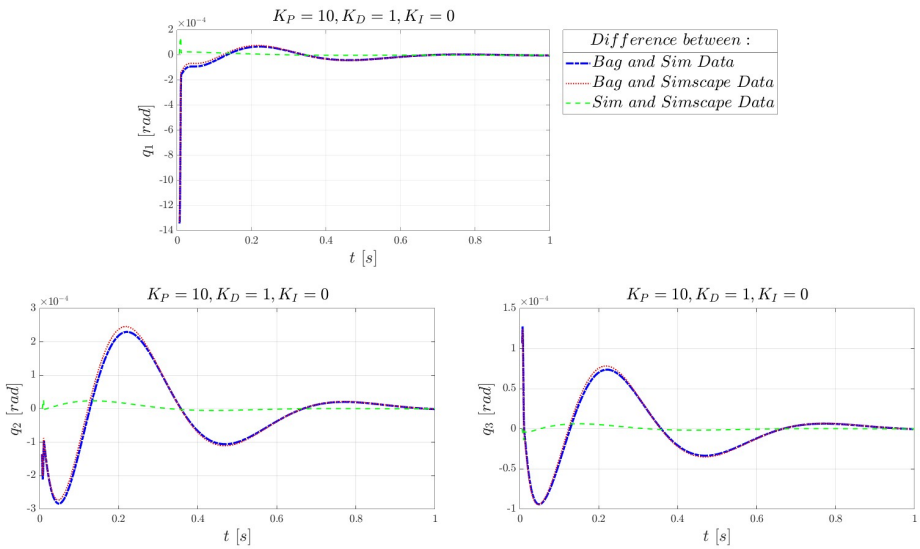


Figure 22: Deviation plots of joint position histories for the three joints for the simulation of the Position Controller in matlab, simscape and gazebo.

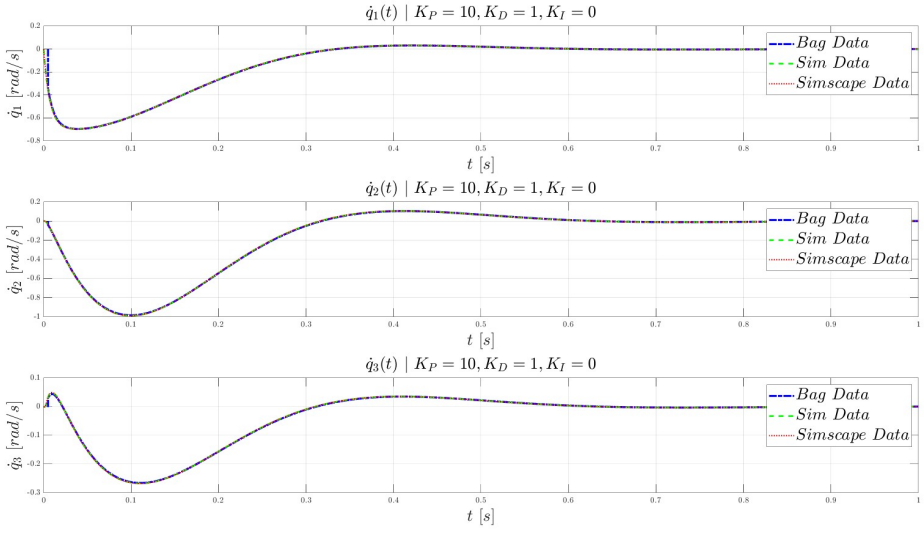


Figure 23: Comparison of joint velocities histories for the three joints for the simulation of the Position Controller in matlab, simscape and gazebo.

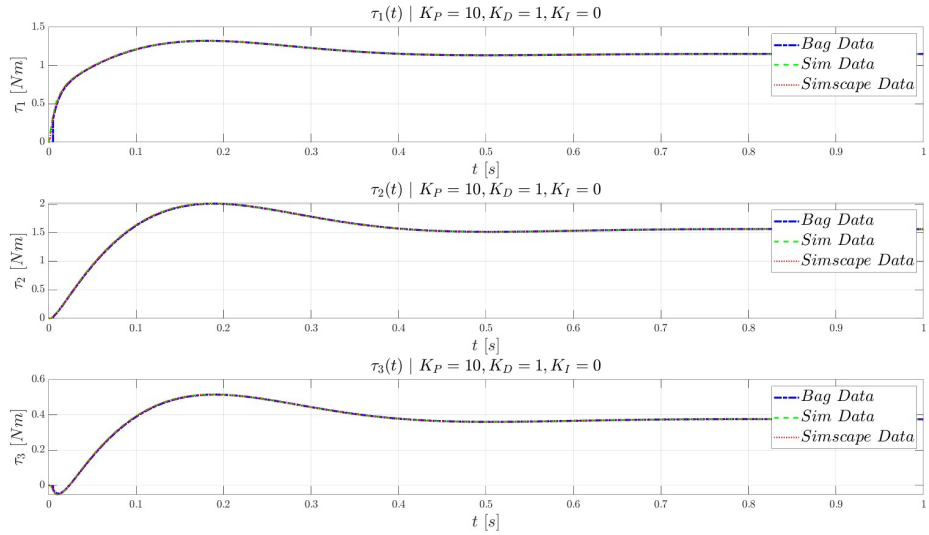


Figure 24: Comparison of actuator torque histories for the three joints for the simulation of the Position Controller in matlab, simscape and gazebo.

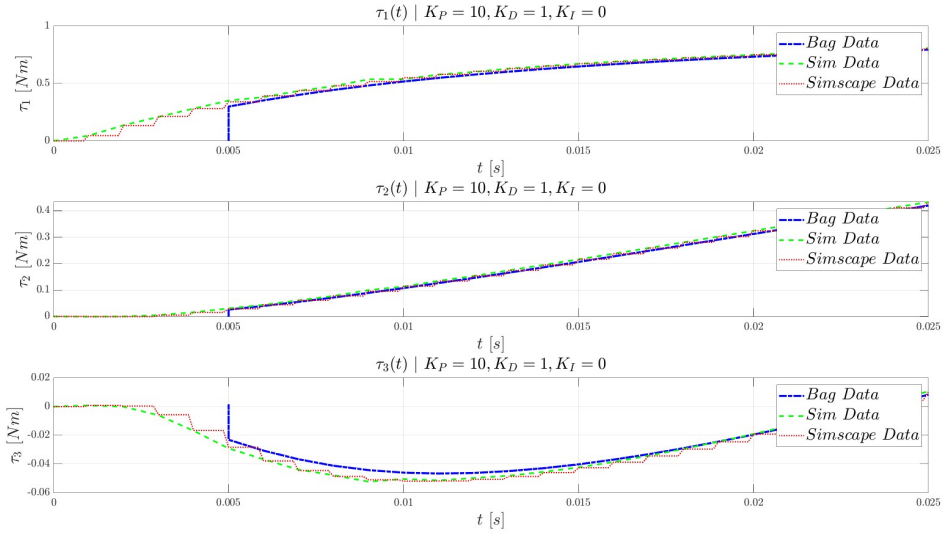


Figure 25: Focused plot, for comparison of actuator torque histories for the three joints for the simulation of the Position Controller in matlab, simscape and gazebo.

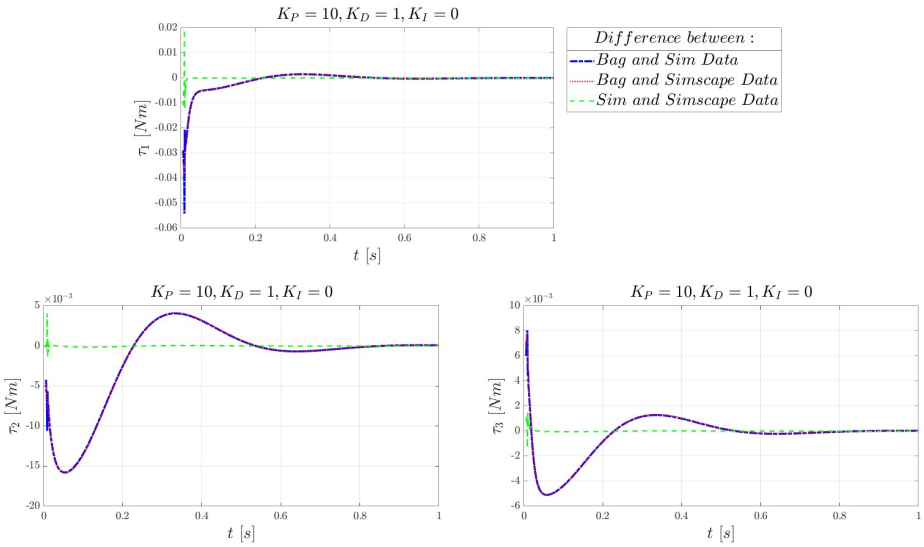


Figure 26: Deviation plots of actuator torque histories for the three joints for the simulation of the Position Controller in matlab, simscape and gazebo.

7.2 Trajectory Control Comparison

For this simulation, the initial conditions are: $\mathbf{q} = [0, 0, 0]$, $\dot{\mathbf{q}} = [0, 0, 0]$ and the desired ellipse is described by the following equations as already discussed:

$$r(\theta) = \frac{ab}{\sqrt{a^2 sin^2(\theta - d\theta) + b^2 cos^2(\theta - d\theta)}}$$

$$^{W}P = r(\theta) \begin{bmatrix} 0 \\ cos(\theta) \\ sin(\theta) \end{bmatrix} + \begin{bmatrix} LB0 + L12 \\ DY \\ DZ \end{bmatrix}$$

and the parameter values are presented in the table 9.

Table 9: Ellipse parameters.

a	0.15	**b**	0.06
DY	0.11208	**DZ**	0.105
dθ	0	**T**	5

Firstly, the reference trajectories are compared in figure 27 and it can be observed that two trajectories are the same[35].

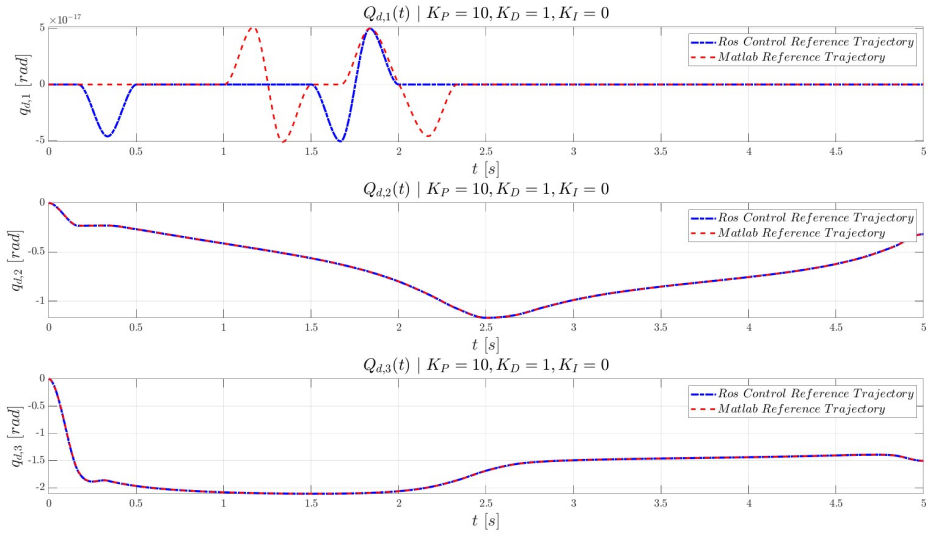


Figure 27: Comparison between the generated reference trajectories in matlab/simscape and gazebo.

In the following figures (28,29, 30, 31,32) the results from the three simulations are compared. The errors in figures 29 and 32 again show a big initial difference between the gazebo and matlab simulations. This deviation stems from ros intrinsic delays and from the non perfect synchronization of the data. Indeed, after some time the errors drop significantly, even though trajectory tracking is a dynamic task. The position simulation error percentage **after the initial difference** ($t > 1s$) is less than 0.2% across all joints while the torque simulation error percentage is is less 0.5% across all joints .

[35] In figure 27, the reference trajectory between matlab and gazebo seem to differ. But, the units are $1e-17rad$. So $q_{1,d}$ is practically zero.

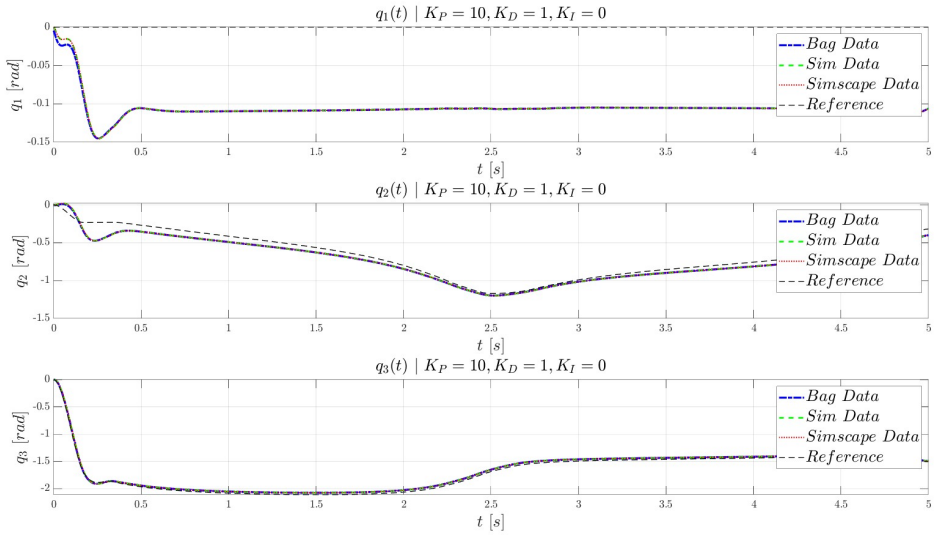


Figure 28: Comparison of joint position histories for the three joints for the simulation of the Trajectory Controller in matlab, simscape and gazebo.

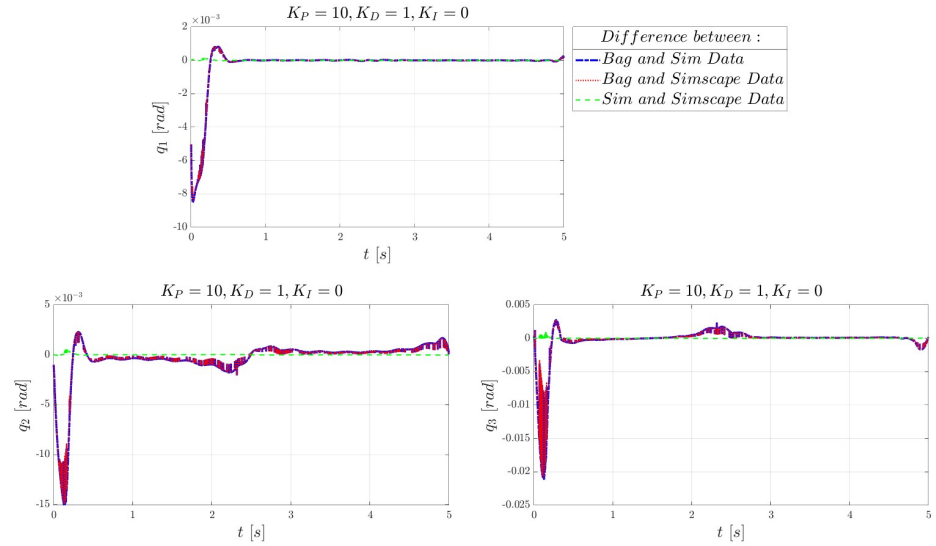


Figure 29: Deviation plots of joint position histories for the three joints for the simulation of the Trajectory Controller in matlab, simscape and gazebo.

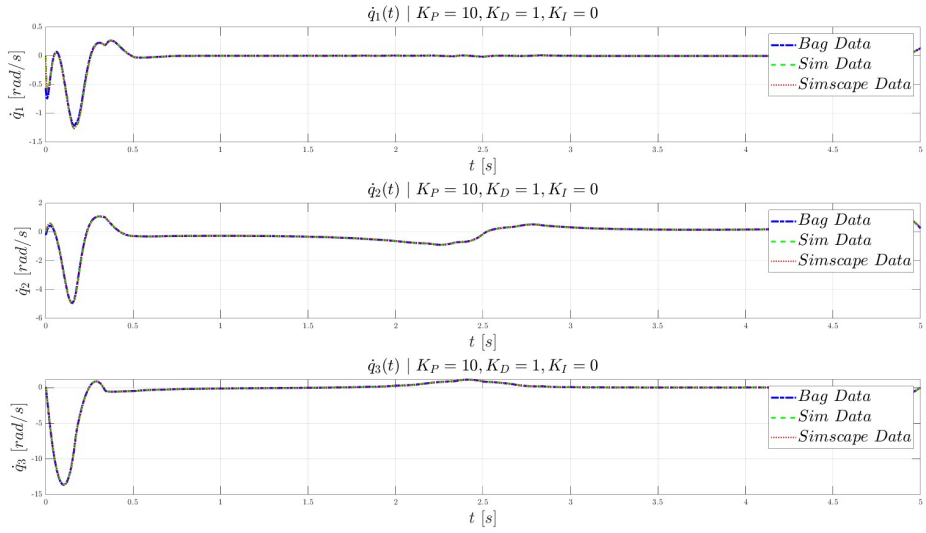


Figure 30: Comparison of joint velocities histories for the three joints for the simulation of the Trajectory Controller in matlab, simscape and gazebo.

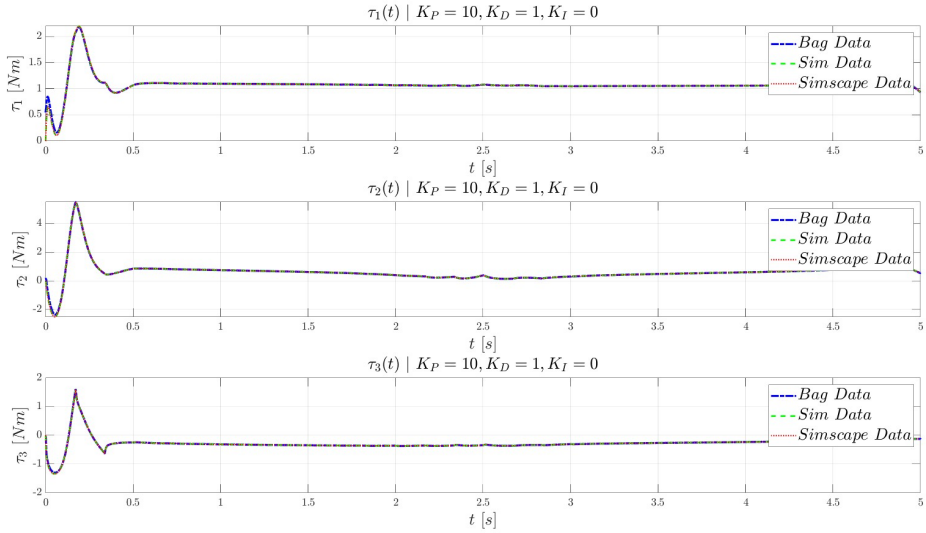


Figure 31: Comparison of joint torque histories for the three joints for the simulation of the Trajectory Controller in matlab, simscape and gazebo.

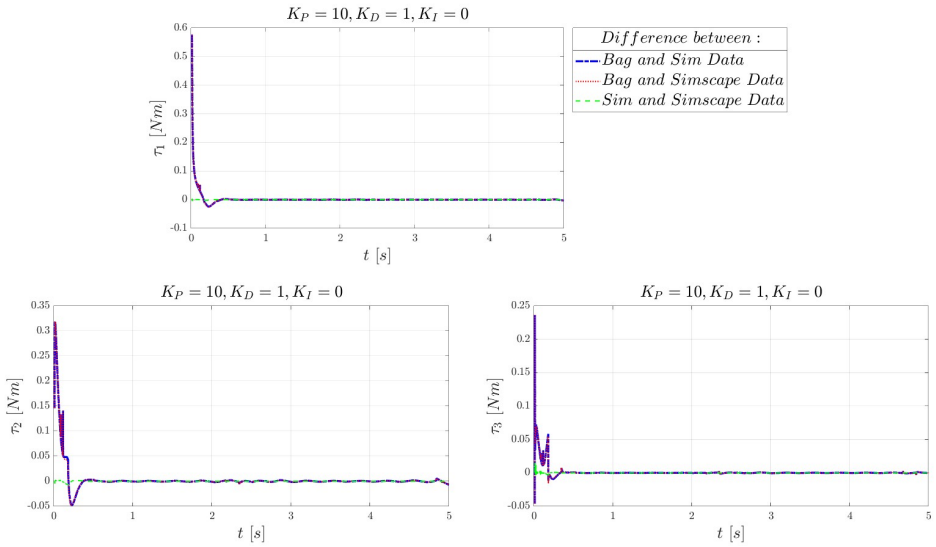


Figure 32: Deviation plots of joint torque histories for the three joints for the simulation of the Trajectory Controller in matlab, simscape and gazebo.

7.3 Effort Control Comparison

For this simulation, the initial conditions are: $\mathbf{q} = [-0.0025, -1.1899, -1.2599]^T$, $\dot{\mathbf{q}} = [0, 0, 0]^T$ and desired wrench is: $\mathbf{h}_d = [0, 0, -100]^T$. The initial stance of the leg is shown in figure 33.

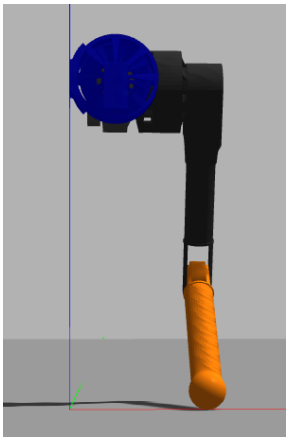


Figure 33: Leg starting position for effort control test.

In the following figures (34, 35, 36, 37, 38, 39, 40) the results from the three simulations are compared. The position error is greater, as seen in figure 35. That is because there is no position controller to drive the errors to zero. Thus, the initial difference, which is a result of the ros service delay, does not decay. In addition, some errors between the matlab simulations and gazebo seem to increase over time. That is because **the stick slip model**, which is used in simscape and in the custom simulation, **doesn't make the velocity zero**, as can be seen in figure 17.

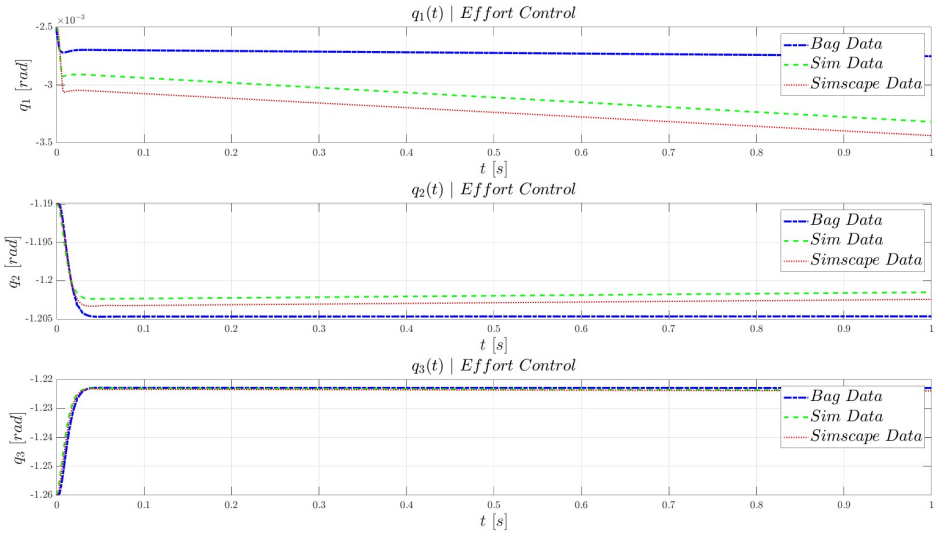


Figure 34: Comparison of joint position histories for the three joints for the simulation of the Effort Controller in matlab, simscape and gazebo.

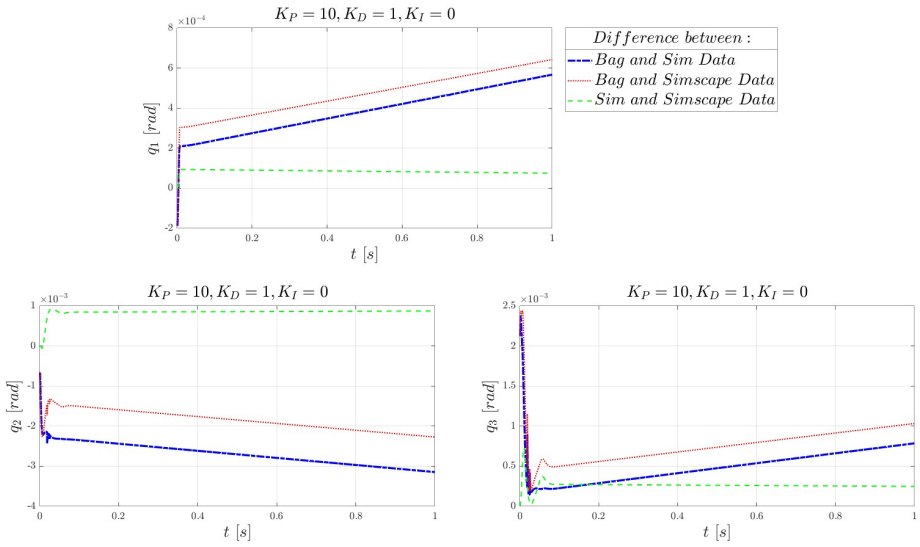


Figure 35: Deviation plots of joint position histories for the three joints for the simulation of the Effort Controller in matlab, simscape and gazebo.

Figures 36 and 37 show the velocity histories of the joints. The transient response (figure 36) is similar across all frameworks. However, in gazebo, the steady state of joint velocities of joints 2 and 3 are ten times higher (shown in figure 37). The numerical values are presented in table 10.

Important: The velocities that are extracted from the `legjoint_states` topic, which is the ros topic that gazebo publishes the leg states, do not seem to be the correct velocities.

Firstly, the angular joint positions in gazebo have a much smaller inclination (almost zero) compared to the joint velocities in matlab and simscape, even though the reported joint velocities in gazebo are ten times higher. This can be observed from a focused joint position plot in figure 38.

Indeed, doing the math, from the velocity plots (figure 37 or table 10), the joint angular displacements are:

$$\begin{aligned}
\bar{q}_{t,1} = -2.26 \cdot 10^{-5}\ rad/s &\rightarrow \Delta q_1 \approx (-2.26 \cdot 10^{-5}) \cdot 0.8 = 1.81 \cdot 10^{-5}\ rad \\
\bar{q}_{t,2} = 3.59 \cdot 10^{-3}\ rad/s &\rightarrow \Delta q_2 \approx (3.59 \cdot 10^{-3}) \cdot 0.8 = 2.9 \cdot 10^{-3}\ rad \\
\bar{q}_{t,3} = -8.62 \cdot 10^{-3}\ rad/s &\rightarrow \Delta q_3 \approx (-8.62 \cdot 10^{-3}) \cdot 0.8 = -6.9 \cdot 10^{-3}\ rad
\end{aligned}$$

However, from the focused position plots (figure 38), the measured joint displacement in joints are :

$$\begin{aligned}
\Delta q_1 &\approx -0.00275114 + 0.00270626 = -4.488 \cdot 10^{-5}\ rad \\
\Delta q_2 &\approx -1.20465 + 1.20468 = 3 \cdot 10^{-5}\ rad \\
\Delta q_3 &\approx -1.22289 + 1.22285 = -4 \cdot 10^{-5}\ rad
\end{aligned}$$

This is because gazebo uses a coulomb friction model[36]. According to this model, if:

$$|F_{tangential}| < \mu |F_{normal}| \tag{59}$$

the simulator prevents the contacting surfaces from sliding. In the current simulation, $\mu = 0.7$, $F_n \approx 50N$ and thus $\mu |F_{normal}| \approx 35N$ while the friction forces are less than $10N$.

Conclusion: The true $\dot{\mathbf{q}}$ in gazebo is closer to zero, as the contact point is not moving. However, the simulator does not publish this joint velocity.

---

[36] `http://www.ode.org/ode-latest-userguide.html#sec_3_11_1`

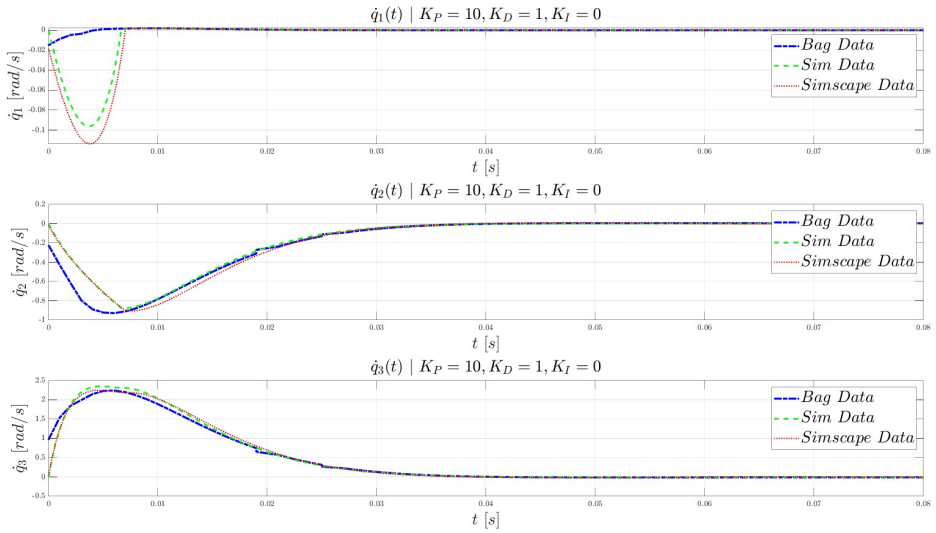


Figure 36: Comparison of transient response of joint velocities for the three joints for the simulation of the Effort Controller in matlab, simscape and gazebo.

Table 10: Steady state velocity values for static effort control test.

	$q_{t,1}$	$q_{t,2}$	$q_{t,3}$
Gazebo	2.3e-5	36e-4	86e-4
Simscape	40e-5	8.7e-4	6.4e-4
Custom sim	40e-5	9e-4	6.4e-4

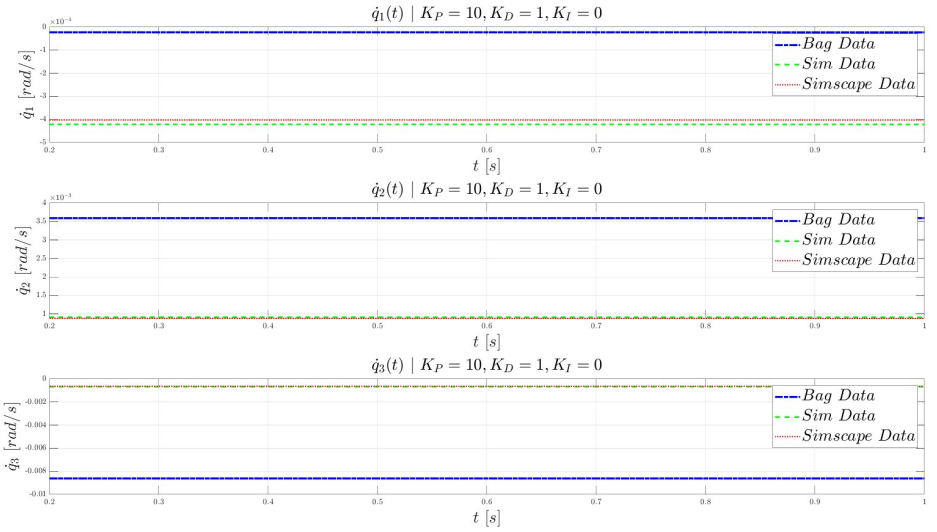


Figure 37: Comparison of steady state of joint velocities for the three joints for the simulation of the Effort Controller in matlab, simscape and gazebo.

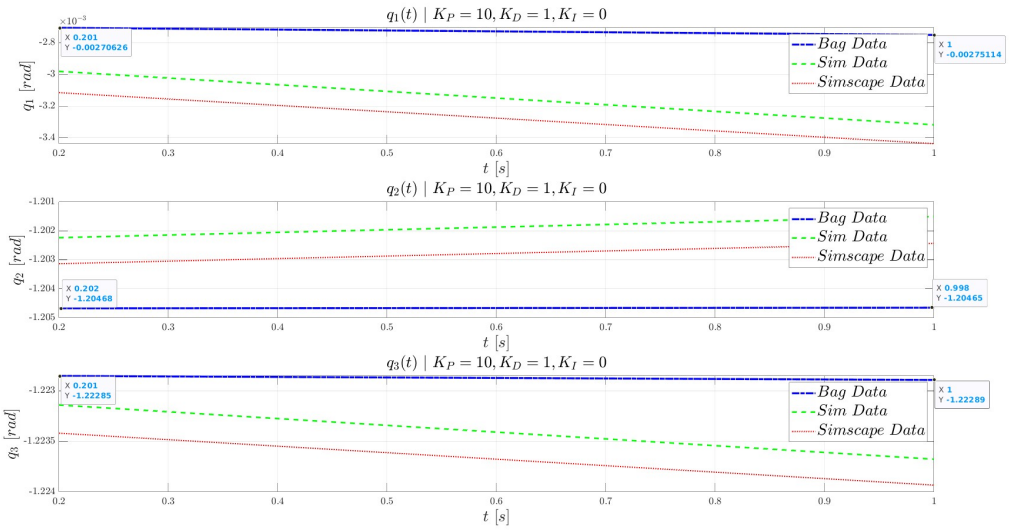


Figure 38: Focused comparison of joint position histories for the three joints for the simulation of the Effort Controller in matlab, simscape and gazebo.

Figure 39 shows the torque histories of the joints. There is a 0.6% difference in the torque of the second joint between the frameworks, which is the only joint that hasn't reached its torque limit. This error is a result of the slightly different positions in every simulation.

Figure 40 shows the contact forces histories. The transient responses of the friction forces have some differences that are to be expected due to the different friction model used by gazebo. However, the magnitude of the friction forces are comparable. The normal force has a similar transient response in all frameworks. The steady state values of the contact normal force are presented in table 11. The steady state values[37] of the normal force have a difference less than 1.5%, while the frictional forces, which use different models, have a difference less than 2.2%.

The normal forces exhibit a difference of about $1.65N$ between the custom simulation and gazebo. In the custom simulation, the point-of-contact is the end-effector origin, so the reported value must be compensated. This is discussed in section 7.3.1.

The difference between Gazebo and simscape is around $0.65N$. This may be a results of the slightly different positions between the simulations, as well as some numerical errors. To validate the gazebo simulator, the states of the leg $\mathbf{q}$ and $\dot{\mathbf{q}}$ as well as the input torque vector $\boldsymbol{\tau}$ were taken from the simulator, and the theoretical interaction force was calculated, and is reported in the **Analytical verification of Gazebo** entry, in table 11. There is still a difference of around $0.75N$.

Table 11: Steady state contact forces values for static effort control test.

	F_n $[N]$	F_x $[N]$	F_y $[N]$
Gazebo	51.8005	6.1125	-9.0825
Simscape	52.4594	5.9817	-9.0575
Custom sim	53.4656	6.0364	-9.3415
Analytical verification of Gazebo	52.3048	5.9949	-9.1750
Compensated Custom sim	52.5660	5.9471	-8.9943

[37] Regarding the custom simulation, the compensated value was used for comparison.

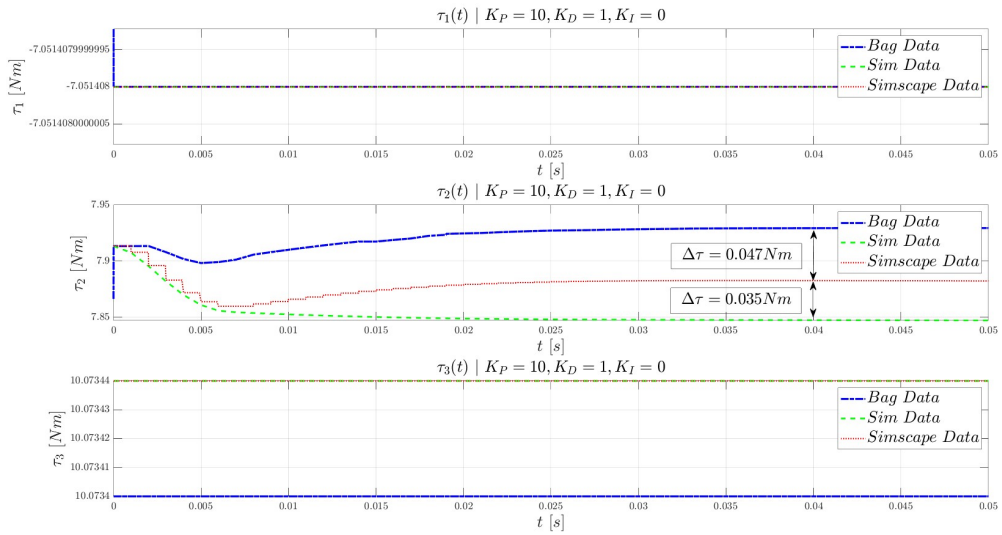


Figure 39: Comparison of joint torque histories for the three joints for the simulation of the Effort Controller in matlab, simscape and gazebo.

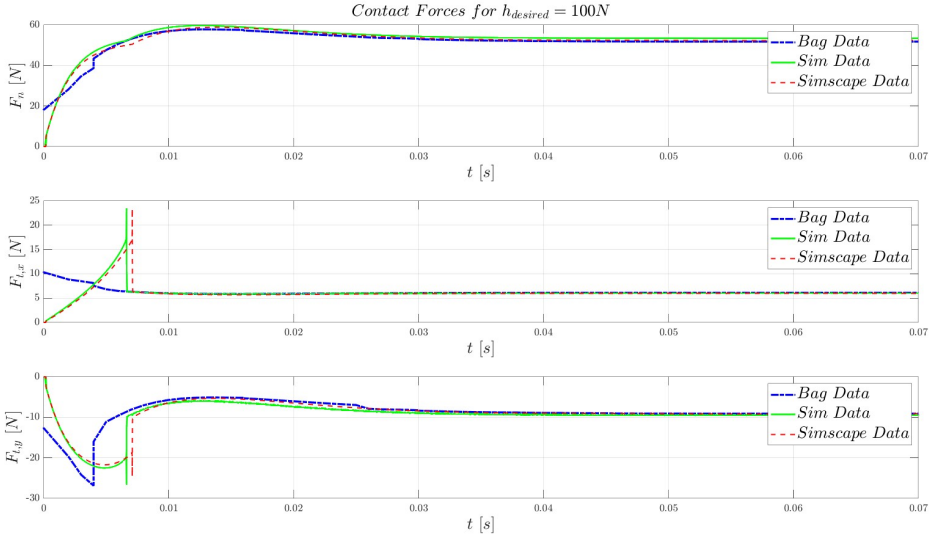


Figure 40: Comparison of the end effector contact force histories for the simulation of the Effort Controller in matlab, simscape and gazebo. (The plot doesn't show the whole experiment, as at around $5ms$, the forces reach a steady state.)

7.3.1 Explain contact force difference between simscape and custom contact model

The real contact point is not the end effector origin. The real contact point is given by the following translation from the end effector, as seen in figure 18:

$$^{W}\mathbf{P}_{poc} = [0,\ 0,\ -R_{foot} - y_d]$$

where $y_d = F_n/K_p$ is the depth of the contact. The orientation of the end effector is given by $^{W}\mathbf{R}_3 = {}^{W}\mathbf{R}_E$. Thus:

$$^{E}\mathbf{P}_{poc} = {}^{W}\mathbf{R}_E^T\ {}^{W}\mathbf{P}_{poc}$$

So the transformation matrix from the end effector to the contact point is:

$$^{E}\mathbf{T}_{poc} = \begin{bmatrix} \mathbf{I}_{3\times3} & {}^{E}\mathbf{P}_{poc} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}$$

With the above information, the geometric jacobian of this particular point of contact can be found using the procedure described in section 3.3.

So, using the joint torques from the custom simulation and the formula (60), the foot - ground interaction force can be calculated.

$$\mathbf{h} = (\mathbf{J}_{poc}^T)^{-1}(\boldsymbol{\tau} - \mathbf{G}) \tag{60}$$

Using the methodology above, the true contract force of the point of contact in the custom simulation is very close to the simscape contact force. Thus, the difference between the two models is indeed a result of the point of contact not coinciding with the end effector point in the center of the sphere of the foot.

7.4 Combined Task Comparison

For this simulation, the following task was simulated:

1. Cartesian target for the position controller ${}^{W}P_1 = [0.1667, 0.05, 0.08]$. The gains for the position controller are $K_p = 10,\ K_d = 1$ for all the joints.
2. Cartesian target for the position controller ${}^{W}P_2 = [0.1667, 0, 0.0225]$ after $1s$. After this step, **the foot of the leg is touching**[38] **the ground**.
3. Vertical force target of $100N$ for the effort controller after $1s$.

The initial conditions for the matlab and simscape simulation are: $\mathbf{q} = [0, 0, 0]$, $\dot{\mathbf{q}} = [0, 0, 0]$. In gazebo, the foot was initially given the default position control target $\mathbf{q}_d = [0, 0, 0]$. This difference is apparent in figures 43 and 44, as there are notable differences for $t < 0.5s$. However, the first task is a position control task with no contact, and thus the simulations will converge at $t = 1s$. The comparisons are concerned with the results after $t = 1s$ and the simulation is deemed acceptable[39].

In the following figures (41, 42, 43, 44) the results of the three simulations are compared. Figures 41 and 42 show that there are slight position errors. These are mainly due to the contacts[40] and thus they do not converge, as discussed in the previous experiment. Figure 43 shows the joint torque histories during the experiment. There seems to be a delay at $t = 2s$, which is discussed in section 7.4.1. Figure 44 shows the contact histories and table 12 shows the steady state values of the normal forces, including the compenstated value from the custom simulation.

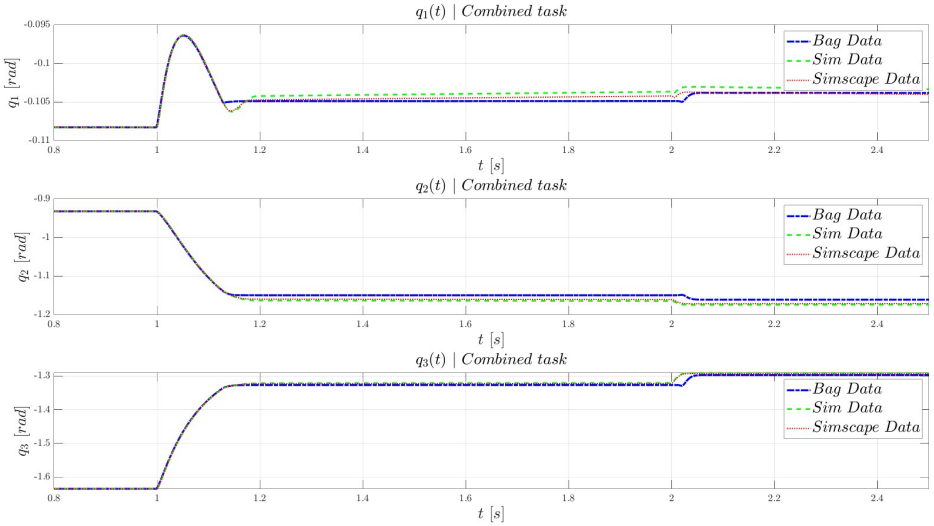


Figure 41: Comparison of joint position histories for the three joints for the simulation for a combined position and effort control in matlab and simscape.

[38] Without gravity, the sphere of the foot would be tangential to the ground.

[39] The synchronization of the joint data was done in such a way that the simulations match at $t = 1s$.

[40] One contact happens at around $t = 1.29s$ and the second is at around $t = 2s$, when the effort controller is activated.

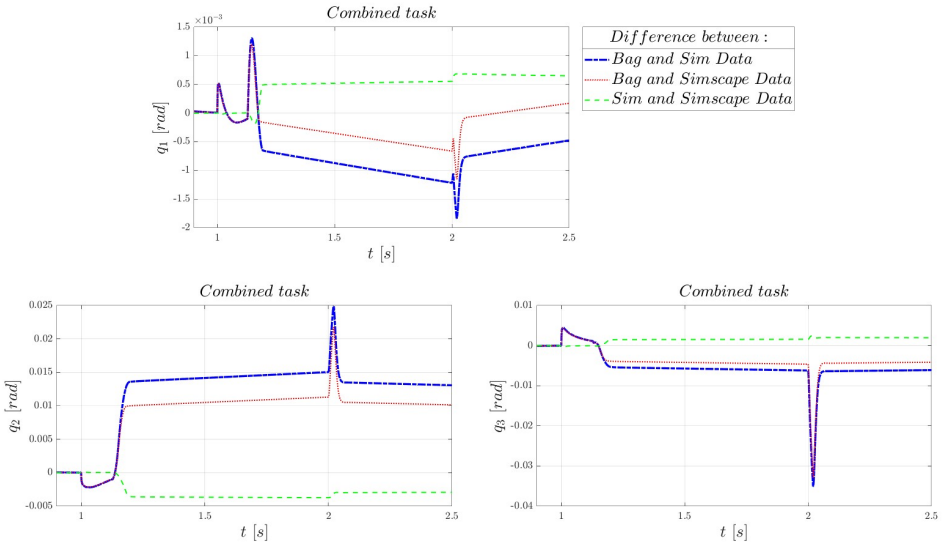


Figure 42: Deviation of joint position histories for the three joints for the simulation for a combined position and effort control in matlab and simscape.

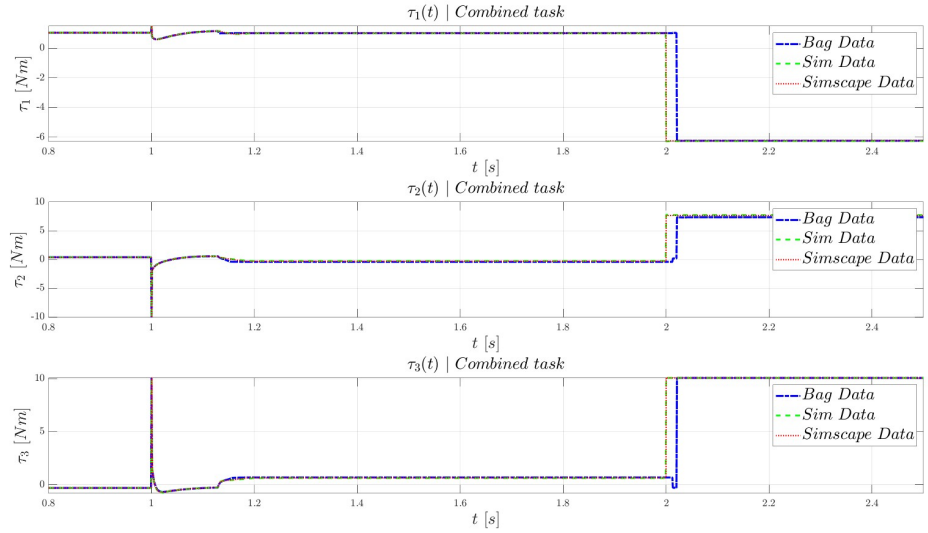


Figure 43: Comparison of joint torque histories for the three joints for the simulation for a combined position and effort control in matlab and simscape.

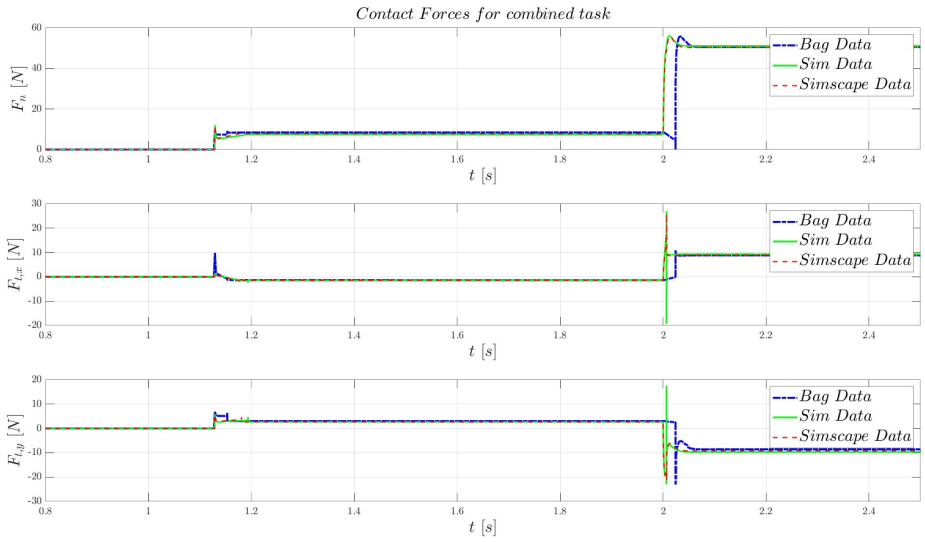


Figure 44: Comparison of the end effector contact force histories for a combined position and effort control in matlab and simscape.

Table 12: Steady state contact forces values for combined task control test.

	Simscape Contact Force	**Custom simulation Contact Force**	**Contact Force using point of contact Jacobian**	**Gazebo Simulation**
Normal Force	50.4603 N	51.1818 N	50.4555 N	50.6283

7.4.1 Explain delays between the matlab simulations and gazebo.

A c++ program was developed (`legSim.cpp`) to call the required services in order to simulate the effort task. All the wait time of $1s$ is hard-coded, using the command:

```
ros::Duration(1).sleep();
```

After each service call, an information message is displayed in the terminal to inform the user about the evolution of the task. The terminal output is presented in figure 45. In figure 46 the results of the the simulation are presented.

From figure 45 it can be observed that there is a $13ms$ delay[41] regarding the controller switching (loading the effort controllers and unloading the position controllers) and a $7ms$ delay from the deployment of the effort controller to issuing the effort command.

```
INFO] [1682413949.862237092, 4.135000000]: [Sim]: New position goal is [0.166700,0.000000,0.022500]
INFO] [1682413950.876347287, 5.148000000]: [Sim]: Select Effort Control
INFO] [1682413950.883939152, 5.155000000]: [Sim]: New effort vertical goal is -100.000000
```

Figure 45: Commands line output from the `LegSim` node.

In figure 46, there is a difference in the transient response of the torque commands. However, this difference is not due to mistakes in the modeling, but rather **a result of normal delays** in the ros ecosystem. During the first delay of $12ms$ the torque commands keep their previous values. Then, there is a slight jump because the effort controller is activated, with $F_{desired} = \mathbf{0}$. The torque commands is :

$$\boldsymbol{\tau} = \mathbf{J}^T \mathbf{F}_{desired} + \mathbf{G}$$

The torque vector is $\tau = [1.04247,\ 0.186749,\ -0.30101]$. Indeed, the gravity vector from the custom simulation (which has a slightly different position) is: $G = [1.0426,\ 0.1631,\ -0.2984]$. The duration of this *"jump"* is $7ms$. Thus, this jump is the second delay that was observed from the command line (the delay from the deployment of the effort controller to issuing the effort command).

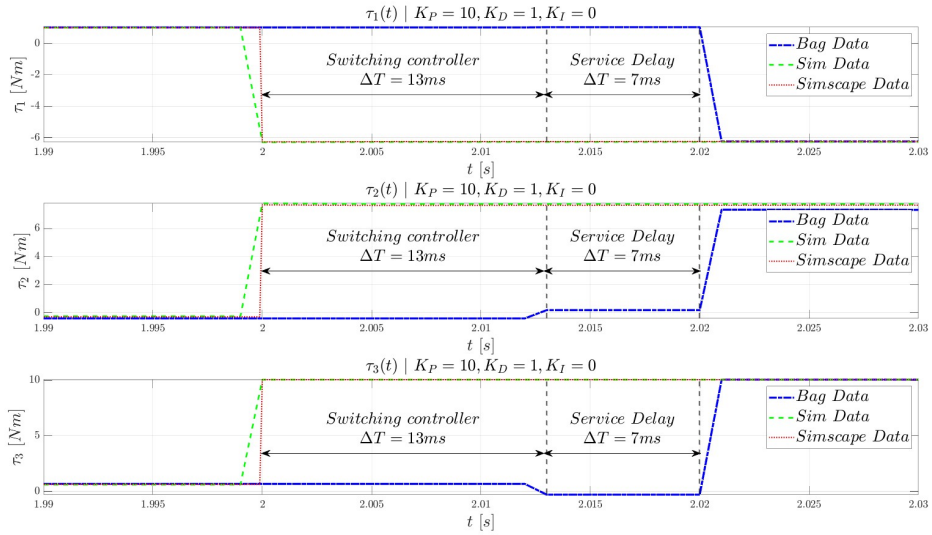


Figure 46: Detailed comparison of joint torque histories for the three joints for the simulation for a combined position and effort control in matlab, simscape and gazebo.

[41] The time difference between the position command and the effort controller selection is $\Delta T = 5.148 - 4.135 = 1.013s$. But the one second is the task wait time, so the *"unexpected"* delay is only $t_{delay} = 13ms$.

8. Appendix

8.1 Creating the cad model

The design was based on J.Valvis diploma thesis [3]. The goal of this model is to :

- visualize the leg during simulations,
- exactly specify the lengths that affect the leg kinematics,
- estimate the inertial quantities.

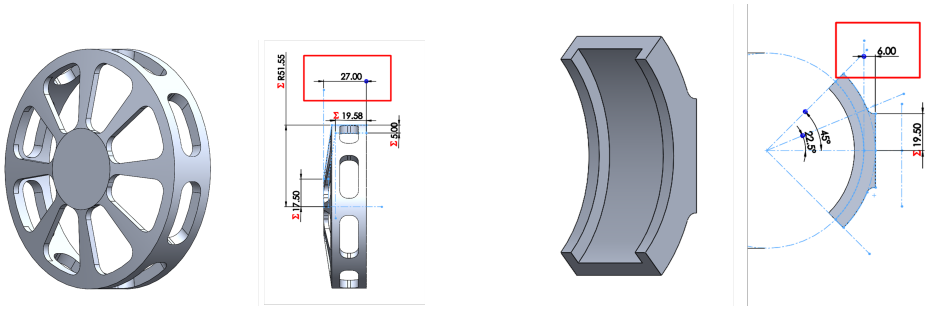
Thus, special care was taken in the cad modeling to keep the dimensions that affect the kinematics of the leg the same as the measured ones and include the main volumes of the leg in order to get a satisfactory estimate of the inertial properties.

According to the aforementioned thesis [3], the material used for the actuators is aluminum 7075. The cylindrical part of the legs are made of carbon fiber.

Table 13: Materials used.

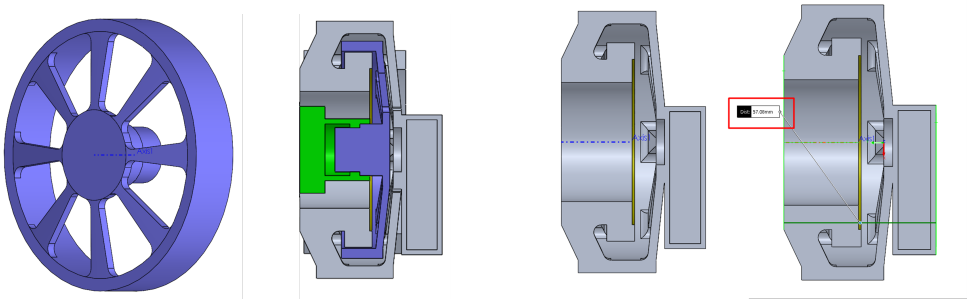
material	solidworks material	density $[kg/m^3]$
Aluminum 7075	7075-O (SS)	2810
Carbon fiber	thronerl mat vma	2000

The modeled parts are presented in the figures below (Figures 47, 48, 49, 50) The dimensions that must match the ones on the real leg to achieve the same kinematic behaviour have been marked.



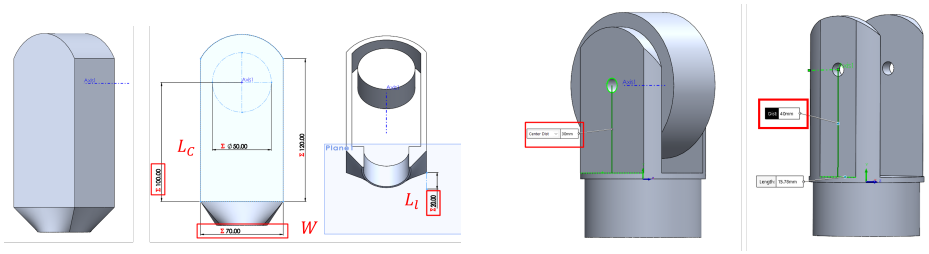
(a) Connector 1. (b) Connector 2.

Figure 47: Connector parts.



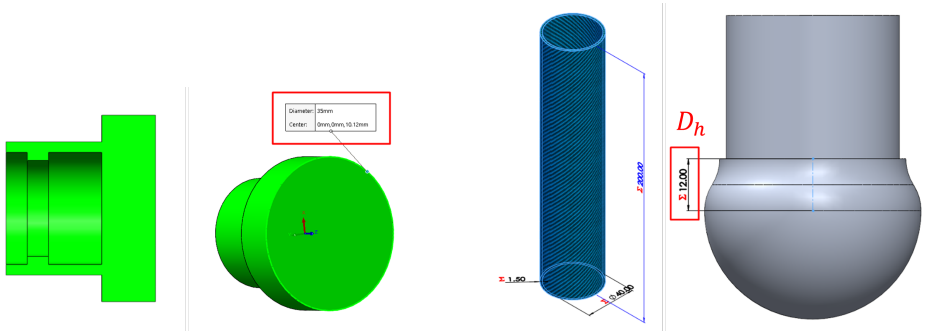
(a) Motor rotor. (b) Motor stator.

Figure 48: Motor parts.



(a) Hip. (b) Knee.

Figure 49: Link 2 parts.



(a) Planetary gearbox carrier. (b) Leg and foot.

Figure 50: Rest leg parts.

The measurements from existing leg are presented in table 14. (Lengths are in mm).

Table 14: Measured quantities for the cad modeling of the leg.

Connector 1	4	**Knee (link 3)**	28.5 (+3?)
Connector 2	30	**Leg** (D)	38
Hip (W)	45	**Leg Length (link 2)**	105
Hip (L_c)	-	**Leg Length (link 3)**	171
Hip (L_λ)	31	**Foot** (D)	45
Knee (link 2)	53.5	**Foot** (D_h)	20.5

Finally, it is desirable for the origin of the coordinate system of the STL models to match the origin that was specified in 1b and 1a. Thus, after defining a custom origin in the cad model, the assemblies are saved as a whole part and then the part is exported (saved as) as an STL. Before exporting the part, some settings must be tweaked according to the figure 51.

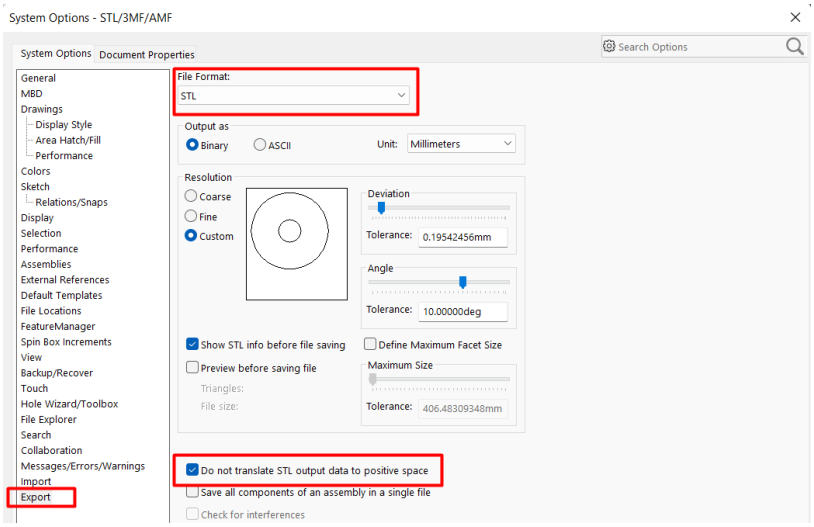


Figure 51: STL export settings.

In the section 8.2, the inertial properties of the parts, as calculated from solidworks, are included for the sake of thoroughness.

8.2 Cad model inertial properties from solidworks

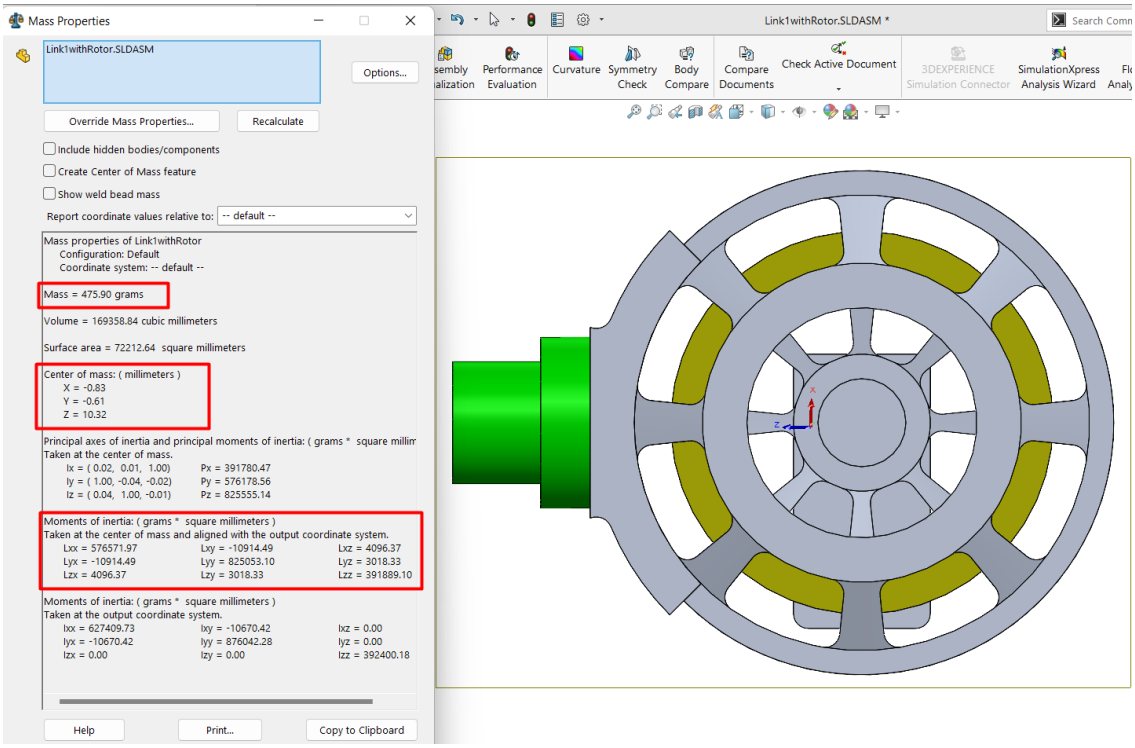


Figure 52: Link 1 inertial properties.

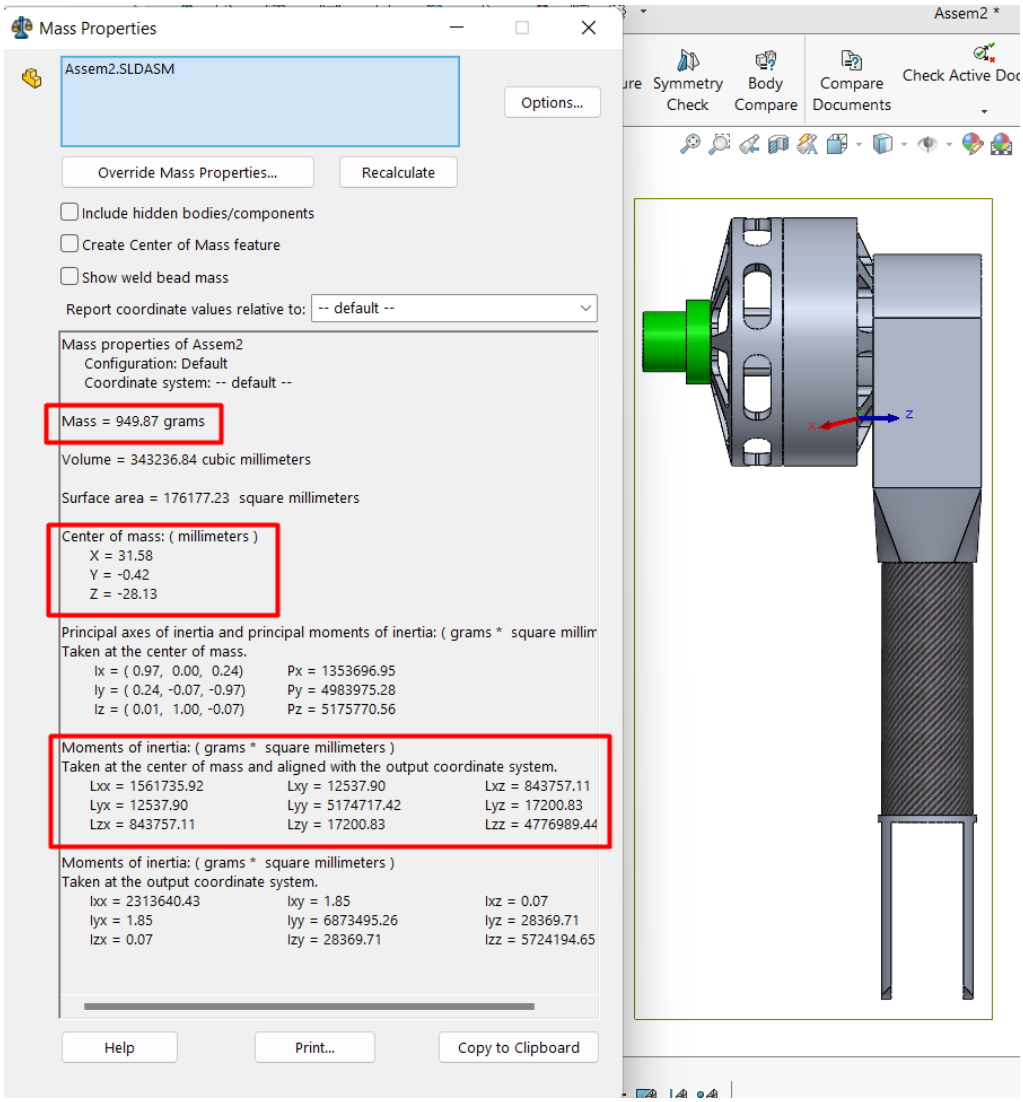


Figure 53: Link 2 inertial properties.

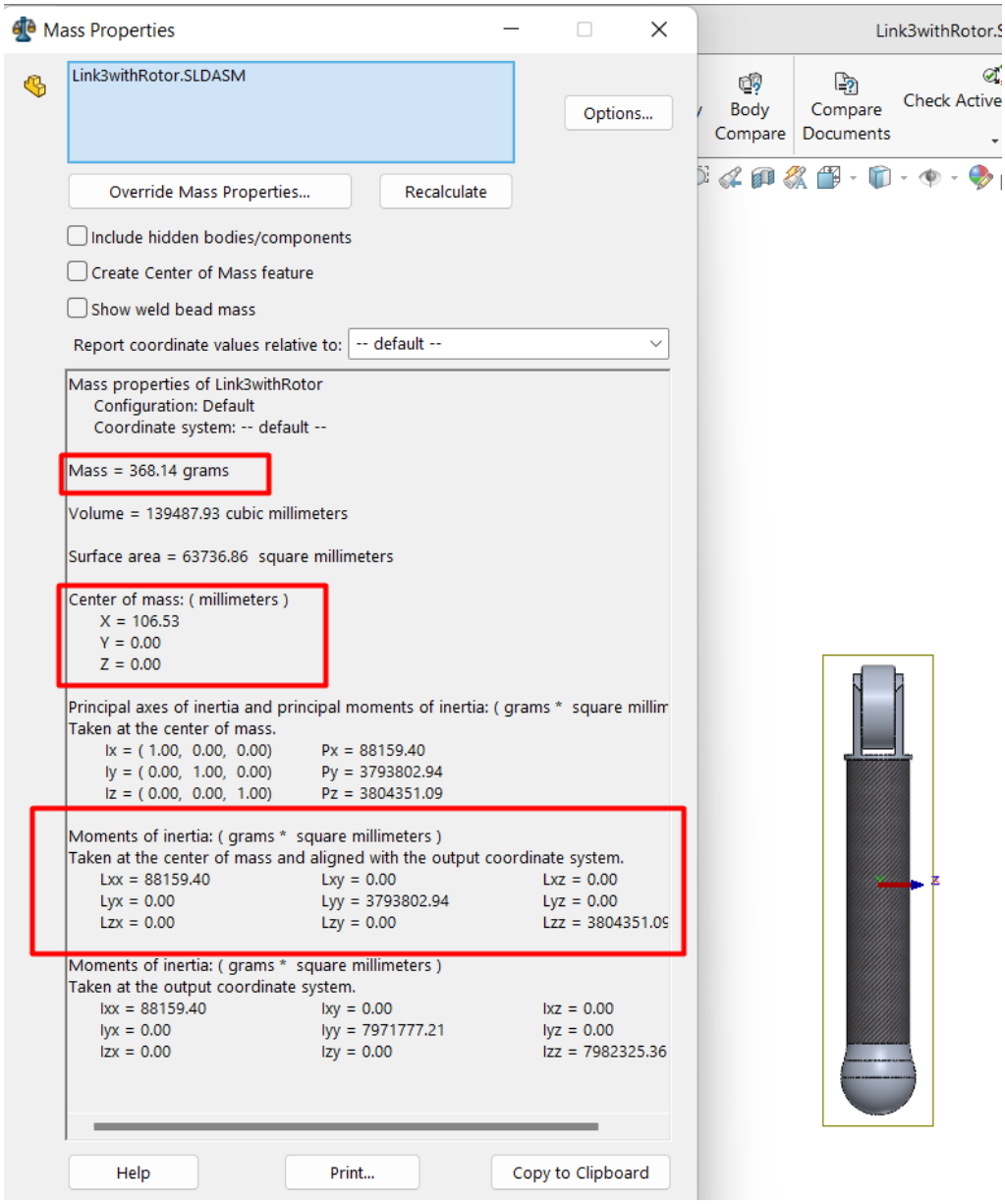


Figure 54: Link 3 inertial properties.

8.3 Inertial Conventions

There are two conventions regarding the products of inertia:

- Negative products of inertia:

$$I_{xy} = -\int_V xy\rho dV, ...$$

  with:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

  This convention is used in simscape[42] and URDFs[43].

- Positive products of inertia:

$$Ixy = \int_V xy\rho dV, ...$$

  with :

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

  This convention is used in solidworks [44].

If one has the inertia data ($I_{xx},\ I_{xy},\ I_{xz},\ I_{yy},\ I_{yz},\ I_{zz}$) without knowing which inertia convention has been used, it can be determined if one also has the principal axes[45]. The eigenvectors of the inertial tensor must match the principal axes.

[42] https://www.mathworks.com/help/releases/R2021b/physmod/sm/ug/specify-custom-inertia.html#mw_b043ec69-835b-4ca9-8769-af2e6f1b190c

[43] http://wiki.ros.org/urdf/XML/link

[44] https://help.solidworks.com/2020/English/SolidWorks/sldworks/HIDD_MASSPROPERTY_TEXT_DLG.htm?id=08ca07d7443c4172beea359efad56a08#Pg0

[45] https://en.wikipedia.org/wiki/Moment_of_inertia#Determine_inertia_convention_(Principal_axes_method)

8.4 Actuator Inertia

This paragraph investigates whether it is possible to add the inertial properties of the actuator **rotor** to the link it actuates, as the URDF format doesn't include them separately.

According to Siciliano[46] ([2]), the kinetic energy of link i is given by equation (61) and the kinetic energy of the **rotor** of the actuator i is given by equation (62) (it is assumed that the actuator i is connected to link $i-1$):

$$T_{l_i} = \frac{1}{2} m_{l_i} \dot{\mathbf{q}}^T \mathbf{J}_P^{(l_i)T} \mathbf{J}_P^{(l_i)} \dot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{J}_A^{(l_i)T} \mathbf{R}_{l_i} ({}^{l_i}\mathbf{I}_{l_i}) \mathbf{R}_{l_i}^T \mathbf{J}_A^{(l_i)} \dot{\mathbf{q}}^T \tag{61}$$

$$T_{m_i} = \frac{1}{2} m_{m_i} \dot{\mathbf{q}}^T \mathbf{J}_P^{(m_i)T} \mathbf{J}_P^{(m_i)} \dot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \mathbf{J}_A^{(m_i)T} \mathbf{R}_{m_i} ({}^{m_i}\mathbf{I}_{m_i}) \mathbf{R}_{m_i}^T \mathbf{J}_A^{(m_i)} \dot{\mathbf{q}}^T \tag{62}$$

The following are true:

$$\mathbf{J}_P^{(l_i)} = [\mathbf{J}_{P_1}^{(l_i)} \ \dots \ \mathbf{J}_{P_i}^{(l_i)} \ 0 \ \dots \ 0] \tag{63}$$

$$\mathbf{J}_{P_K}^{(l_i)} = \begin{cases} \mathbf{z}_k & \text{for a prismatic joint} \\ \mathbf{z}_k \times (\mathbf{p}_{l_i} - \mathbf{p}_k) & \text{for a revolute joint} \end{cases} \tag{64}$$

$$\mathbf{J}_P^{(m_i)} = [\mathbf{J}_{P_1}^{(m_i)} \ \dots \ \mathbf{J}_{P_{i-1}}^{(m_i)} \ 0 \ \dots \ 0] \tag{65}$$

$$\mathbf{J}_{P_K}^{(m_i)} = \begin{cases} \mathbf{z}_k & \text{for a prismatic joint} \\ \mathbf{z}_k \times (\mathbf{p}_{m_i} - \mathbf{p}_k) & \text{for a revolute joint} \end{cases} \tag{66}$$

Generally $\mathbf{p}_{m_i} \neq \mathbf{p}_{l_i}$, where $\mathbf{p}_{l_i}$ is the position of the center of mass of the link i and $\mathbf{p}_{m_i}$ is the position of actuator. Thus $\mathbf{J}_P^{(m_i)} \neq \mathbf{J}_P^{(l_{i-1})}$.

Also :

$$\mathbf{J}_A^{(m_i)} = [J_{A_1}^{(m_i)} \ \dots \ J_{A_i}^{(m_i)} \ 0 \ \dots \ 0] \tag{67}$$

$$J_{A_K}^{(m_i)} = \begin{cases} J_{A_K}^{(l_i)} & \text{for } j \neq i \\ k_{ri} \mathbf{z}_{m_i} & \text{for } j = 1 \end{cases} \tag{68}$$

It must be investigated if there is an I_i' such that:

$$\mathbf{J}_A^{(l_i)T} \mathbf{R}_{l_i} (I_i') \mathbf{R}_{l_i}^T \mathbf{J}_A^{(l_i)} = \mathbf{J}_A^{(m_i)T} \mathbf{R}_{m_i} ({}^{m_i}\mathbf{I}_{m_i}) \mathbf{R}_{m_i}^T \mathbf{J}_A^{(m_i)} \tag{69}$$

If there is such an I_i', then it is possible to directly add I_i' to the link i inertia. In the following investigation, the frame of link i and actuator i are parallel, so $\mathbf{R}_{m_i} = \mathbf{R}_{l_i} = R$ and $\mathbf{z}_{m_i} = \mathbf{z}_i$. The RHS of the equation (69) is:

$$\begin{bmatrix} \left[\mathbf{J}_{A_1}^{(l_i)T}\right]_{1x3} \\ \dots \\ k_{ri} \left[\mathbf{J}_{A_K}^{(l_i)T}\right]_{1x3} \\ \mathbf{0}_{1x3} \end{bmatrix} \underbrace{\mathbf{R} ({}^{m_i}\mathbf{I}_{m_i}) \mathbf{R}^T}_{\mathbf{RIR}_{3x3}^T} \begin{bmatrix} \left[\mathbf{J}_{A_1}^{(l_i)}\right]_{1x3}, \ \dots \ , \ k_{ri} \left[\mathbf{J}_{A_K}^{(l_i)}\right]_{1x3}, \ \mathbf{0}_{1x3} \end{bmatrix}$$

And finally:

$$\begin{bmatrix} \left[\mathbf{J}_{A_1}^{(l_i)T}\right]_{1x3} \mathbf{RIR}^T \left[\mathbf{J}_{A_1}^{(l_i)}\right]_{1x3} & \dots & k_{ri} \left[\mathbf{J}_{A_1}^{(l_i)T}\right]_{1x3} \mathbf{RIR}^T \left[\mathbf{J}_{A_K}^{(l_i)}\right]_{1x3} & \mathbf{0} \\ \vdots & \ddots & \vdots & \mathbf{0} \\ k_{ri} \left[\mathbf{J}_{A_K}^{(l_i)T}\right]_{1x3} \mathbf{RIR}^T \left[\mathbf{J}_{A_1}^{(l_i)}\right]_{1x3} & \dots & k_{ri}^2 \left[\mathbf{J}_{A_K}^{(l_i)T}\right]_{1x3} \mathbf{RIR}^T \left[\mathbf{J}_{A_K}^{(l_i)}\right]_{1x3} & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

If in the above equation, the variables are set to $k_{ri} = 1$ and $I = I_i'$ in order to get the LHS of the equation (69):

[46] The symbols used by Siciliano are adopted in the following text and their definitions are not repeated for brevity.

$$\begin{bmatrix} \begin{bmatrix} \mathbf{J}_{A_1}^{(l_i)T} \end{bmatrix}_{1x3} \mathbf{R}\mathbf{I}_i'\mathbf{R}^T \begin{bmatrix} \mathbf{J}_{A_1}^{(l_i)} \end{bmatrix}_{1x3} & \dots & \begin{bmatrix} \mathbf{J}_{A_1}^{(l_i)T} \end{bmatrix}_{1x3} \mathbf{R}\mathbf{I}_i'\mathbf{R}^T \begin{bmatrix} \mathbf{J}_{A_K}^{(l_i)} \end{bmatrix}_{1x3} & \mathbf{0} \\ \vdots & \ddots & \vdots & \mathbf{0} \\ \begin{bmatrix} \mathbf{J}_{A_K}^{(l_i)T} \end{bmatrix}_{1x3} \mathbf{R}\mathbf{I}_i'\mathbf{R}^T \begin{bmatrix} \mathbf{J}_{A_1}^{(l_i)} \end{bmatrix}_{1x3} & \dots & \begin{bmatrix} \mathbf{J}_{A_K}^{(l_i)T} \end{bmatrix}_{1x3} \mathbf{R}\mathbf{I}_i'\mathbf{R}^T \begin{bmatrix} \mathbf{J}_{A_K}^{(l_i)} \end{bmatrix}_{1x3} & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

Thus, there is no I_i' that satisfies the equation (69). So, the inertia of the rotor cannot be included in the inertia of the link. Intuitively, if the i-th rotor inertia is included in the inertia of the i-th link as $k_{i,r}^2 I_{rotor,i}$, then this "equivalent" inertia will be in effect even in rotations caused by previous actuation units.

For the example, if the actuator $i-1$ in figure 55 (whose axis of rotation is parallel to the axis of rotation of actuator i, as indicated by the double lines) causes links $i-1$ and i to rotate, then the rotor inertia should not be multiplied by $k_{i,r}^2$.

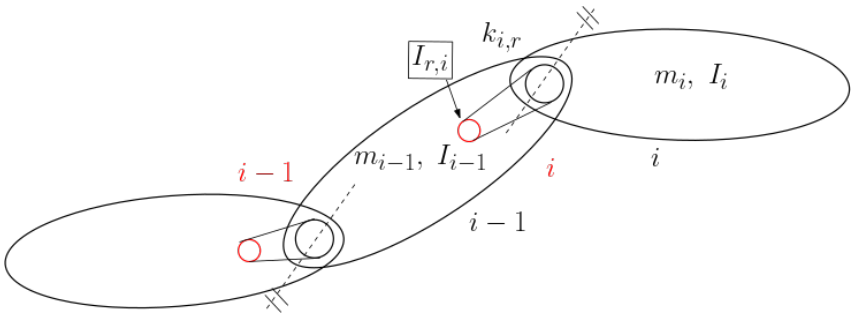


Figure 55: Actuator rotor Inertia. (Rotors are shown in red color.)

This fact has been observed in this post[47].

[47] https://github.com/ros/urdfdom/issues/78

8.5 Basic simulation in the simscape framework

A basic matlab script to run the simulation is presented below in listing 1. In this script:

1. The user sets up some simulation parameters (like sampling rate and controller gains),
2. creates the controller and simulation handler objects,
3. activates the custom-made simulations (`s.customSimulationsSwitch(true)`),
4. sets up the task they want to simulate, creates the corresponding events, and
5. runs the simscape simulation

```
%% Simulation paramters: ------------------------
Tsampling = 1e-3;

%set K for class implementation:
Kp = [10;10;10];
Kd = [1;1;1];
Ki = 0*[1;1;1];
K = [Kp,Kd,Ki];

%%  Set up simulation: --------------------------
r= legRobot(tmax=1.007344*[7;10;10]);
qinitial=[0;0;0]; %set initial conditions for simscape

PC = PosControl(r,Tsampling,K); %Position Controller
TC = TrajControl(r,Tsampling,K); %Trajectory Controller
EC = TrajControl(r,Tsampling); %Trajectory Controller

% simulation_preparation
s = simulation_preparation(r,PC,TC,EC);
s.setInitialConditions([0;0;0;0;0;0],0);
s.customSimulationsSwitch(true); % false -> do not run custom sim

%%  set up tasks: -------------------------------
PC.setTarget([0.1667;0.3;0.5]);
s.setEvent(0,1,1);

%script to set up a 3xN vector x, and a 1xN vector tw. N = #
    waypoints
testXpos
TC.generateQ(x,tw,1);
s.setEvent(1,2,tw(end));

PC.setTarget([0.1667;0.3;0.5]);
s.setEvent(tw(end)+1,1,1);

%% Run simscape simulation:
[Select_Controller,Qpos,TSqd,TSqtd,TSh] = s.SimScapeSimulation();
out = sim('Leg_Simulation_Framework_test_2.slx',[0 s.t_total]);
```

Listing 1: Basic script to set up and run the matlab/simscape simulations

8.5.1 Framework available commands

For each task the user wants to simulate, the user uses the controller methods to set the task parameters and then they create an event using the command [48]:

```
s.setEvent(t_start,Controller,dt);
```

where `t_start` is the start time of the event, `Controller` specifies the controller that is activated during the task and `dt` is the duration of the task. Regarding the controllers, the following correspondence is true:

1. → Position Controller
2. → Trajectory Controller
3. → Effort Controller

Important: To run the custom-made simulations, the `customSimulationsSwitch` method must be called, passing `true`. This method must be called before setting any event!

The user can currently simulate the following tasks:

- Set a joint-angle target for the position controller:

```
PC.setAngleTarget(Qdesired)
```

  where `Qdesired` is a 3×1 column vector with the desired joint angles.

- Set a Cartesian target for the position controller:

```
PC.setTarget(Xdesired)
```

  where `Xdesired` is a 3×1 column vector with the desired position in the world frame.

- Set a custom trajectory with N waypoints for the trajectory controller:

```
TC.generateQ(Xwp,Twp,Tstart)
```

  where `Xwp` is a $3 \times N$ column vector with the desired positions of the waypoints in the world frame, `Twp` is the timestamps for each of the waypoints and `Tstart` is the starting time of the trajectory.

- Set an ellipse in the YZ plane for the trajectory controller:

```
TC.generateEllipse(a,b,DX,Dth,T,Npoints,Tstart)
```

  where `a`, `b`, `DX`, `dth` are ellipse parameters (`DX` is a 2×1 vector specifying the center of the ellipse in the YZ plane $DX = [DY, DZ]^T$), `T` is the period of the trajectory and `Tstart` is the starting time of the trajectory. The ellipse parameterization is the following:

$$r(\theta) = \frac{ab}{\sqrt{a^2 sin^2(\theta - d\theta) + b^2 cos^2(\theta - d\theta)}}$$

$${}^W P = r(\theta) \begin{bmatrix} 0 \\ cos(\theta) \\ sin(\theta) \end{bmatrix} + \begin{bmatrix} LB0 + L12 \\ DY \\ DZ \end{bmatrix}$$

- Set a effort target for the effort controller:

```
EC.setWrench(Hdesired)
```

  where `Hdesired` is a 3×1 column vector with the desired force vector in the world frame.

The user has access to the robot states and actuator inputs from the custom simulation using this command:

```
[t,y,u] = s.getCustomSimData();
```

[48] In this example `s` is the `simulation_preparation` object. In the following examples `PC`, `TC` are the position and trajectory controller objects respectively.

9. References

[1] Iro Papagiannaki. optimization-based motion planning for quadruped robots, 2022.

[2] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010.

[3] Ι Βάλβης. ΘΕΡΜΙΚΗ ΜΕΛΕΤΗ, ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΚΑΤΑΣΚΕΥΗ ΣΥΣΤΗΜΑΤΟΣ ΗΛΕΚΤΡΙΚΟΥ ΚΙΝΗΤΗΡΑ ΚΑΙ ΕΝΣΩΜΑΤΩΜΕΝΟΥ ΠΛΑΝΗΤΙΚΟΥ ΜΕΙΩΤΗΡΑ ΓΙΑ ΕΠΕΝΕΡΓΗΣΗ ΤΩΝ ΑΡΘΡΩΣΕΩΝ ΠΟΔΙΟΥ ΤΡΙΩΝ ΒΑΘΜΩΝ ΕΛΕΥΘΕΡΙΑΣ, 2020.