Internship Report

Bernard Léo

May-August 2023

Abstract

During my second-year internship at the University of Oldenburg, I worked with a six-degree-of-freedom robotic arm. My primary objective was to establish a physical model of the robot's dynamic behavior using the Euler-Lagrange equation. To accomplish this, I implemented the equation of motion in Python, enabling me to conduct tests on this equation.

Résumé

Pendant mon stage de deuxième année à l'Université d'Oldenburg, j'ai travaillé avec un bras robotique à six degrés de liberté. Mon objectif principal était d'établir un modèle physique du comportement dynamique du robot en utilisant l'équation d'Euler-Lagrange. Pour ce faire, j'ai implémenté l'équation du mouvement en Python, ce qui m'a permis de mener des tests sur cette équation.

1 Introduction

his report documents my internship experience at the **Carl von Ossietzky University of Oldenburg**, situated in the north-west region of Germany (cf. fig 1). Specifically, the internship was conducted within the Department of Computer Science and more precisely in the Department of Distributed Control in Networked Systems.

The primary aim of this internship was to put the theoretical knowledge acquired during academic coursework into practical applications within the domain of Robotics. Moreover, this internship, conducted outside of France, sought to provide exposure to working within diverse linguistic and professional frameworks, particularly employing English in technical and professional contexts.

During the course of this internship, I worked with a **six-degree-of-freedom robotic arm**. The central objective revolved around constructing a comprehensive model of this robotic system, with the aim of facilitating the subsequent development of an home-made control.

To realize this objective, the internship proceeded in three key phases : gaining an understanding of the robotic arm's mechanics and functionalities, establishing a physical model of the robotic arm, and implementing and evaluating this model with the actual physical robot.

Figure 1: The Carl von Ossietzky University of Oldenburg in Germany

2 The six degrees of freedom robotic arm

The robotic arm I've worked with is a **ViperX-300 6DOF**, a model manufactured and sold by the company Trossen Robotics. It comes with a ROS2 interface and a sdk.
We will first give a description of the mechanical arm, then present the software associated with the robot.

2.1 The physical robot

Figure 2: Image of the ViperX 300S robotic arm

This robotic arm has **six degrees of freedom, all rotational**, they are described on the kinematic diagram below[1] It is equipped with a gripper as end-effector by default but it can be changed.

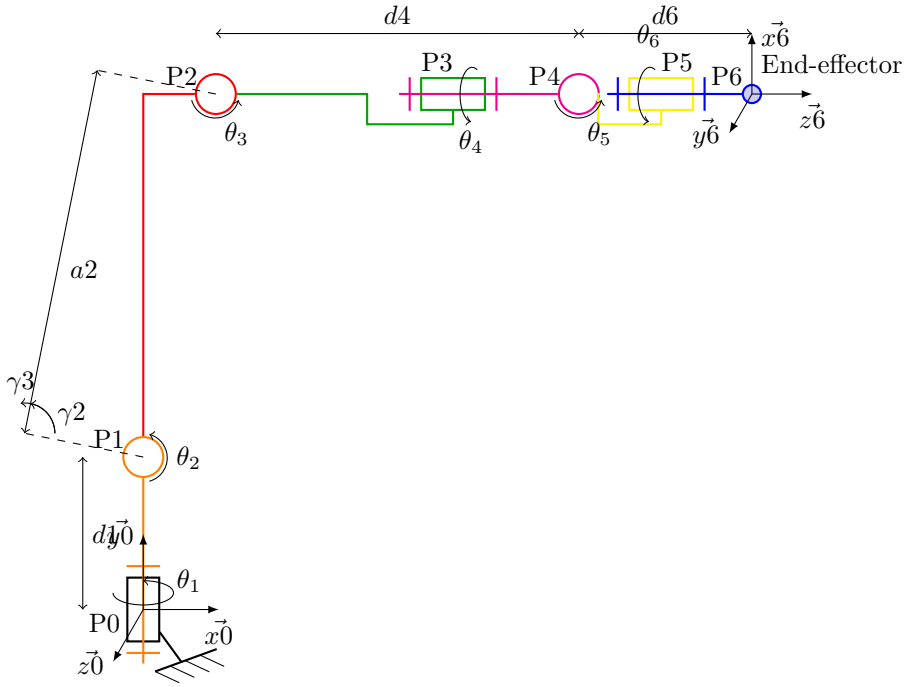

Figure 3: kinematic model of the robot

$d1 = 126.75mm$	$a2 = 305.94mm$	$d4 = 300mm$	$d6 = 143.7mm$	$\gamma 2 \approx 78.69°$

[2]

[1] more info in appendix : 6.1
[2] for more info, refer to the technical drawing here : ViperX-300 6DOF - Specifications

Each six Joints are actuated by servomotors, the shoulder and the elbow are actuated by two coaxial servomotors as they are subjected to greater torque.
The servomotors used are from **Dynamixel's XM** range of product[1] from Robotis.

Figure 4: Servomotors actuators of the robotic arm

Those servos - connected in a serial chain - offer high resolution with a stall torque of 10.6Nm and a powerful embedded controller with definable parameters.

Those motors offer various control modes and return several feedback information, the only control mode used here is the **position control** and the interesting feedback for us are the **measurements of the position, velocity and current**.

2.2 The ROS2 interface

The robot comes with a ROS interface (I used ROS2 humble distro). It includes various services and topics allowing to control the robot or returning the feedback information.

The most interesting ones are the **topics** : `/vx300s/commands/joint_group` which allows to send a command to a group of joint (typically the group 'arm'). And the topic `/vx300s/joint_states` where are published the measured position, velocity and effort of each joint.

The ROS interface also includes **several ROS2 packages** for the robot. Let's discuss a few of them that played a crucial role in our project.

The first one is the Arm description package : it contains the URDFs and meshes of the robotic arm, it allows to visualize the robot in real time in rviz software

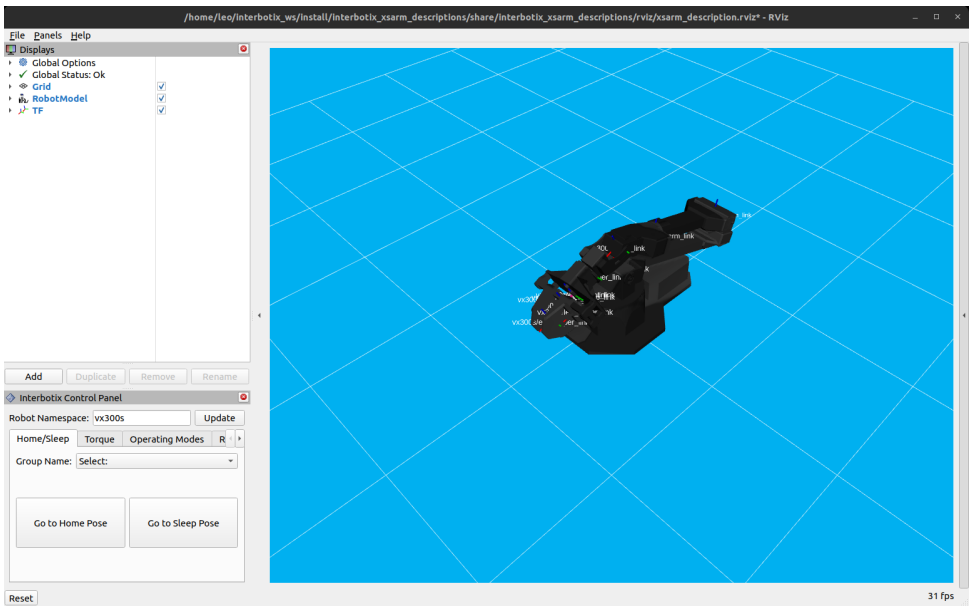

Figure 5: Image of Rviz launched with the control package

The description package is also the one that allows to publish the position, velocity and effort on the `/joint_states` topic.

The second package is the Arm Control package : it initiates the SDK to enable the operation of the robotic arm. it also includes a control panel on rviz

Those were the two most important packages for us, there are various others like a simulation package to simulate the robot in gazebo and a moveit2 package for motion planning.

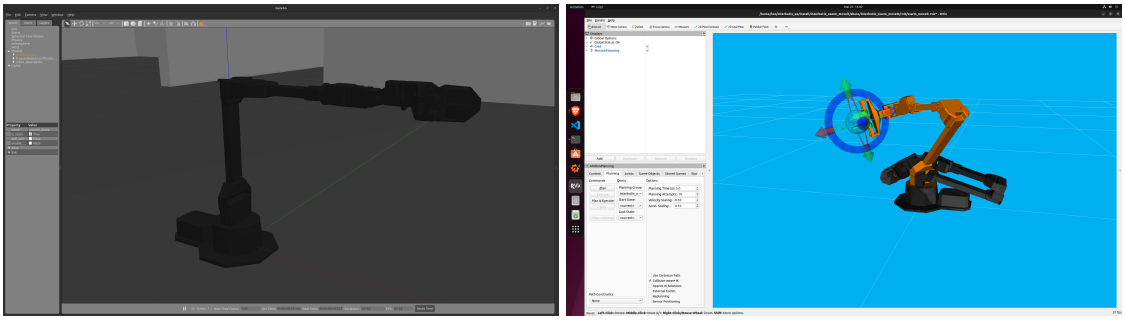

Figure 6: Image of the Gazebo simulation and motion planning with moveit2

There is also a **Python-ROS API** that works with the control package, this API allows to control the robot using Python through already existing functions.

3 Establishing the dynamic model of the robotic arm

To establish the equation of motion of the robotic arm, we will use the **Euler-Lagrange equation**. This equation is particularly well-suited for complex robotic systems with multiple degrees of freedom, enabling us to model the arm's behavior accurately and efficiently.

3.1 The Euler-Lagrange equation

The **Euler-Lagrange equation** is given by:

$$\boxed{\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = \tau_k} \tag{1}$$

for the k-th degree of freedom of the system

Explanation:

- L is the Lagrangian of the system, which is the difference between the kinetic energy (K) and potential energy (P) of the system :

$$L = K - P$$

- q is the vector of generalized coordinates (each element θ_i corresponds to the angle in the i-th joint of the system) :

$$q = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{pmatrix}$$

- τ is the torque applied in each joint :

$$\tau = \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{pmatrix}$$

To determine the Lagrangian, we have to calculate the kinetic and potential energy of the system :

The **kinetic energy** K of the system is given by :

$$K = \frac{1}{2} \cdot \dot{q}^T \sum_{i=1}^{n} \left[m_i \cdot J_{v_i}(q)^T \cdot J_{v_i}(q) \; + \; J_{\omega_i}(q) \cdot R_{i/0}(q) \cdot I_i \cdot R_{i/0}(q)^T \cdot J_{\omega_i}(q) \right] \dot{q} \tag{2}$$

Explanation:

- m_i is the mass of the i-th link
- I_i is the inertia matrix of the i-th link (expressed in frame i)
- J_{v_i} is the linear velocity Jacobian matrix for the i-th link :

$$V_i = J_{vi} \; \cdot \dot{q}_i$$

- J_{ω_i} is the angular velocity Jacobian matrix for the i-th link :

$$\omega_i = J_{\omega i} \; \cdot \; \dot{q}_i$$

- $R_{i/0}$ is the rotation matrix from the frame 0 (base frame) to frame i (attached to link i)

The **potential energy** K of the system is given by :

$$P = \sum_{i=1}^{n} m_i \ \cdot \ g \ \cdot \ h_i \tag{3}$$

Explanation:

- g is the gravity
- h_i is height of the center of gravity of the i-th link

By writing K in matrix form : $K = \frac{1}{2}\ \dot{q}^T \cdot D(q) \cdot \dot{q}$, where d_{ij} are the elements of the matrix D, we can express the Euler-Lagrange equation as follows :

$$\sum_{i=1}^{n} d_{kj}(q) \cdot \ddot{q_j} + \sum_{i,j} c_{ijk} \cdot \dot{q_i} \cdot \dot{q_j} + \ g_k\,(q) \ = \tau_k \tag{4}$$

with :

$$\begin{cases} g_k = \frac{\partial P}{\partial q_k} \\ c_{ijk} = \frac{1}{2}\left(\frac{\partial d_{kj}}{\partial q_i} \ + \ \frac{\partial d_{ki}}{\partial q_j} \ - \ \frac{\partial d_{ij}}{\partial q_k}\right) \qquad (c_{ijk}{=}c_{jik}) \end{cases}^{3} \tag{5}$$

The equation is also written in matrix form :

$$\boxed{D(q) \cdot \ddot{q} + C(q, \dot{q}) \cdot \dot{q} + G(q) = \tau} \tag{6}$$

Physical interpretation of the elements of this equation :

- $\mathbf{D}(q) \cdot \ddot{q}$: This term corresponds to the inertia and acceleration effects. It's a function of the joint angles (q) and represents how the masses of the different components of the robotic arm are distributed and how they influence the system's response to acceleration (second derivative of the joint angles, $\ddot{q}$). In simpler terms, it quantifies the resistance of the arm to changes in its motion due to its inertia.
- $\mathbf{C}(q, \dot{q}) \cdot \dot{q}$: This term corresponds to the centripetal and Coriolis effects. It's a function of both joint angles (q) and their velocities ($\dot{q}$). The centripetal effect is related to the tendency of objects in motion to move towards the center of rotation, which occurs due to internal forces within the system, creating a virtual force. The Coriolis effect comes from the interaction of the arm's motion with the rotation of the reference frame. Together, these effects represent the forces due to the arm's motion in relation to its angular velocities.
- $\mathbf{G}(q)$: This term corresponds to the effect of the gravity on the robotic arm. It's a function of the joint angles (q) and essentially, it captures the forces that arise due to the arm's weight and the position of its center of mass relative to the joints. This term contributes to the arm's potential energy and affects its equilibrium positions.
- τ : This term represents the external torques applied to the joints. These torques can be generated by various sources, such as the servomotors or external forces interacting with the arm. It's the control input that is used to control the motion of the robotic arm.

[3] See proof in appendix 6.1

3.2 Model of a two degrees of freedom arm

To begin, let's apply this equation to a much simpler model than the six degrees of freedom : **Let's consider a robotic arm with only two degrees of freedom**.

Let's call α_1 and α_2 the angles in the two joints, m_1 and m_2 the masses of the two links and I_1 and I_2 their Inertia matrices.

The arm we are working with is represented on the following kinematic diagram :

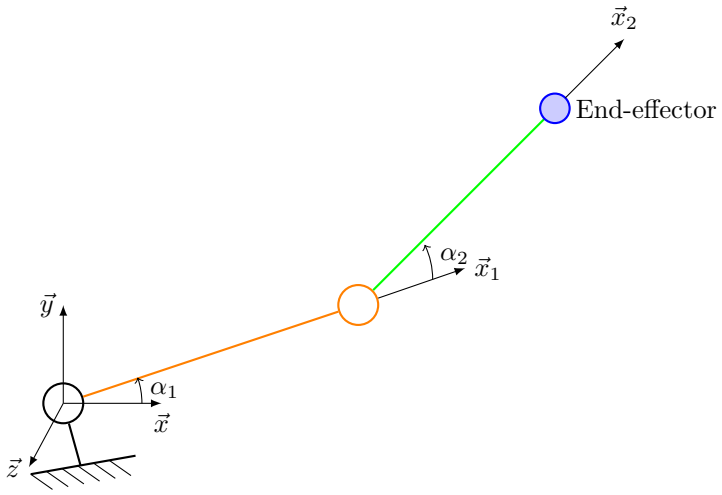

Figure 7: Kinematic diagram of the two degrees of freedom arm

Let's establish the equation of motion of this two degrees of freedom arm using equation (6)

The generalized coordinates vector for this arm is : $q = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$

We first have to calculate all the necessary terms to calculate the kinematic energy :

- the rotation matrices :

$$R_{1/0} = \begin{bmatrix} \cos(\alpha_2) & -\sin(\alpha_2) & 0 \\ \sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$R_{2/1} = \begin{bmatrix} \cos(\alpha_1) & -\sin(\alpha_1) & 0 \\ \sin(\alpha_1) & \cos(\alpha_1) & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$R_{2/0} = \begin{bmatrix} \cos(\alpha_1+\alpha_2) & -\sin(\alpha_1+\alpha_2) & 0 \\ \sin(\alpha_1+\alpha_2) & \cos(\alpha_1+\alpha_2) & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- The Jacobian matrices :

  - for rotational velocity :

$$J_{\omega_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} J_{\omega_2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

- and for linear velocity :

$$\vec{v}_{G_1\in 1/0} = l_{G1}\cdot\dot{\alpha_1}\cdot\vec{y_1} = l_{G1}\cdot\dot{\alpha_1}\cdot(\cos(\alpha_1)\cdot\vec{y_0} - \sin(\alpha_1)\cdot\vec{x_0})$$

$$\begin{aligned}\vec{v}_{G_2\in 2/0} =& \vec{v}_{G_2\in 2/1} + \vec{v}_{G_2\in 1/0} = \vec{v}_{P\in 2/1} + \vec{G_2P}\otimes(\dot{\alpha_2}\cdot\vec{z_1}) + \vec{v}_{O\in 1/0} + \vec{G_2O}\otimes(\dot{\alpha_1}\cdot\vec{z_0})\\ =& [-\sin(\alpha_1)\cdot(l_1 + l_{G2}\cos(\alpha_2)) - l_{G2}\sin(\alpha_2)\cos(\alpha_1)]\cdot\dot{\alpha_1}\cdot\vec{x_0}\\ &+ [-l_{G2}\sin(\alpha_1)\cos(\alpha_2) - l_{G2}\sin(\alpha_2)\cos(\alpha_1)]\cdot\dot{\alpha_2}\cdot\vec{x_0}\\ &+ [\cos(\alpha_1)\cdot(l_1 + l_{G2}\cos(\alpha_2)) - l_{G2}\sin(\alpha_2)\sin(\alpha_1)]\cdot\dot{\alpha_1}\cdot\vec{y_0}\\ &+ [l_{G2}\cos(\alpha_1)\cos(\alpha_2) - l_{G2}\sin(\alpha_2)\sin(\alpha_1)]\cdot\dot{\alpha_2}\cdot\vec{y_0}\end{aligned}$$

$$\Rightarrow J_{v_1} = \begin{bmatrix} -l_{G1}\sin(\alpha_1) & 0\\ l_{G1}\cos(\alpha_1) & 0\\ 0 & 0\end{bmatrix},\quad J_{v_2} = \begin{bmatrix} -l_1\sin(\alpha_1) - l_{G2}\sin(\alpha_1+\alpha_2) & -l_{G2}\sin(\alpha_1+\alpha_2)\\ l_1\cos(\alpha_1) + l_{G2}\cos(\alpha_1+\alpha_2 & l_{G2}\cos(\alpha_1+\alpha_2)\\ 0 & 0\end{bmatrix}$$

- Which means :

$$J_{v_1}^T\cdot J_{v_1} = \begin{bmatrix} l_{G1}^2 & 0\\ 0 & 0\end{bmatrix},\quad J_{v_2}^T\cdot J_{v_2} = \begin{bmatrix} l_1^2 + 2l_1l_{G2}\cos(\alpha_2) + l_{G2}^2 & l_1l_{G2}\cos(\alpha_2) + l_{G2}^2\\ l_1l_{G2}\cos(\alpha_2) + l_{G2}^2 & l_{G2}^2\end{bmatrix}$$

- And :

$$J_{\omega_1}^T\cdot R_{1/0}\cdot I_1\cdot R_{1/0}^T\cdot J_{\omega_1} = \begin{bmatrix} 0 & 0 & 1\\ 0 & 0 & 0\end{bmatrix} I_1 \begin{bmatrix} 0 & 0\\ 0 & 0\\ 1 & 0\end{bmatrix} = \begin{bmatrix} I_{1_{zz}} & 0\\ 0 & 0\end{bmatrix}$$

$$J_{\omega_2}^T\cdot R_{2/0}\cdot I_2\cdot R_{2/0}^T\cdot J_{\omega_2} = \begin{bmatrix} 0 & 0 & 1\\ 0 & 0 & 1\end{bmatrix} I_2 \begin{bmatrix} 0 & 0\\ 0 & 0\\ 1 & 1\end{bmatrix} = \begin{bmatrix} I_{2_{zz}} & I_{2_{zz}}\\ I_{2_{zz}} & I_{2_{zz}}\end{bmatrix}$$

- We have : $D = \sum_{i=1}^{n}\left[m_i\cdot J_{v_i}(q)^T\cdot J_{v_i}(q) \;+\; J_{\omega_i}(q)\cdot R_{i/0}(q)\cdot I_i\cdot R_{i/0}(q)^T\cdot J_{\omega_i}(q)\right]$:

$$\mathbf{D}\left(\begin{pmatrix}\alpha_1\\ \alpha_2\end{pmatrix}\right) = \begin{pmatrix} m_1l_{G1}^2 + m_2\left(l_1^2 + 2l_1l_{G_2}\cos(\alpha_2) + l_{G2}^2\right) + I_{1_{zz}} + I_{2_{zz}} & m_2\left(l_1l_{G_2}\cos(\alpha_2) + l_{G_2}^2\right) + I_{2_{zz}}\\ m_2\left(l_1l_{G_2}\cos(\alpha_2) + l_{G_2}^2\right) + I_{2_{zz}} & m_2l_{G_2}^2 + I_{2_{zz}}\end{pmatrix}$$

- We can write : $K = \frac{1}{2}\dot{q}^TD(q)\dot{q} = \frac{1}{2}d_{11}\dot{\alpha}_1^2 + 2d_{12}\dot{\alpha}_1\dot{\alpha}_2 + d_{22}\dot{\alpha}_2^2$
and then we have the c_{ijk} using (5) :

$$\begin{cases} c_{111} = \frac{1}{2}\left(\frac{\partial d_{11}}{\partial\alpha_1} + \frac{\partial d_{11}}{\partial\alpha_1} - \frac{\partial d_{11}}{\partial\alpha_1}\right) = 0\\ c_{211} = \frac{1}{2}\left(\frac{\partial d_{11}}{\partial\alpha_2} + \frac{\partial d_{12}}{\partial\alpha_1} - \frac{\partial d_{21}}{\partial\alpha_1}\right) = -m_2l_1l_{G2}\sin\alpha_2\\ c_{112} = \frac{1}{2}\left(\frac{\partial d_{21}}{\partial\alpha_1} + \frac{\partial d_{21}}{\partial\alpha_1} - \frac{\partial d_{11}}{\partial\alpha_2}\right) = m_2l_1l_{G2}\sin\alpha_2\\ c_{212} = \frac{1}{2}\left(\frac{\partial d_{21}}{\partial\alpha_2} + \frac{\partial d_{22}}{\partial\partial\alpha_1} - \frac{\partial d_{21}}{\partial\alpha_2}\right) = 0\\ c_{121} = c_{211}\\ c_{221} = \frac{1}{2}\left(\frac{\partial d_{12}}{\partial\alpha_2} + \frac{\partial d_{12}}{\partial\alpha_2} - \frac{\partial d_{22}}{\partial\alpha_1}\right) = -m_2l_1l_{G_2}\sin\alpha_2\\ c_{122} = c_{212} = 0\\ c_{222} = \frac{1}{2}\left(\frac{\partial d_{12}}{\partial\alpha_2} + \frac{\partial d_{22}}{\partial\alpha_2} - \frac{\partial d_{22}}{\partial\alpha_2}\right) = 0\end{cases}$$

$$\begin{cases} c_{211} = c_{121} = \frac{1}{2}\left(\frac{\partial d_{11}}{\partial\alpha_2} + \frac{\partial d_{12}}{\partial\alpha_1} - \frac{\partial d_{21}}{\partial\alpha_1}\right) = -m_2l_1l_{G2}\sin\alpha_2\\ c_{112} = \frac{1}{2}\left(\frac{\partial d_{21}}{\partial\alpha_1} + \frac{\partial d_{21}}{\partial\alpha_1} - \frac{\partial d_{11}}{\partial\alpha_2}\right) = m_2l_1l_{G2}\sin\alpha_2\\ c_{221} = \frac{1}{2}\left(\frac{\partial d_{12}}{\partial\alpha_2} + \frac{\partial d_{12}}{\partial\alpha_2} - \frac{\partial d_{22}}{\partial\alpha_1}\right) = -m_2l_1l_{G_2}\sin\alpha_2\\ c_{122} = c_{212} = c_{111} = c_{212} = c_{222} = 0\end{cases}$$

- We can write the C matrix :

$$\mathbf{C}(q,\dot{q}) = \begin{pmatrix} c_{111}\dot{q}_1 + c_{121}\dot{q}_2 & c_{211}\dot{q}_1 + c_{221}\dot{q}_2 \\ c_{112}\dot{q}_1 + c_{122}\dot{q}_2 & c_{212}\dot{q}_1 + c_{222}\dot{q}_2 \end{pmatrix} = \begin{pmatrix} -m_2 l_1 l_{G_2} \sin(\alpha_2)\,\dot{\alpha}_2 & -m_2 l_1 l_{G_2} \sin(\alpha_2)\,(\dot{\alpha}_1 + \dot{\alpha}_2) \\ m_2 l_1 l_{G_2} \sin(\alpha_2)\,\dot{\alpha}_1 & 0 \end{pmatrix}$$

In a second time, we calculate the potential energy and the G matrix :

$$P = m_1 g l_{G1} \sin\alpha_1 + m_2 g\,(l_1 \sin\alpha_1 + l_{G2} \sin(\alpha_2 + \alpha_1)) + (m_1 + m_2)\,g h_{\text{ref}}$$

$$\begin{cases} \frac{\partial P}{\partial \alpha_1} = m_1 g l_{G1} \cos\alpha_1 + m_2 g l_1 \cos\alpha_1 + m_2 g l_{G2} \cos(\alpha_2 + \alpha_1) = g\cos\alpha_1\,(m_1 l_{G1} + m_2 l_1) + g\cos(\alpha_1 + \alpha_2) m_2 l_{G2} \\ \frac{\partial P}{\partial \alpha_2} = m_2 g l_{G2} \cos(\alpha_1 + \alpha_2) \end{cases}$$

$$\Rightarrow \mathbf{G}(q) = \begin{pmatrix} m_1 l_{G1} g \cos\alpha_1 + m_2 l_1 g \cos\alpha_1 + m_2 l_{G2} g \cos(\alpha_1 + \alpha_2) \\ m_2 l_{G2} g \cos(\alpha_1 + \alpha_2) \end{pmatrix}$$

Now that we have the equation of motion for the two degrees of freedom robotic arm, we want to **use this equation to perform tests on a physical arm**.

To conduct tests with this model, we will use a portion of the six degrees of freedom robotic arm that we will consider to be a two degrees of freedom robotic arm (cf. fig below).

Figure 8: "Artificial" two degrees of freedom arm using the six degrees of freedom arm

Initially, the goal was to evaluate the perturbations in each joint.

We approximated the perturbations to be an external torque, let's write : $\tau = \tau_{mot} + \tau_{pert}$

We then used the equation of motion with the measurements of q, $\dot{q}$ and the 'effort' measurement which we considered to be τ_{mot}[4]. To evaluate the perturbation torque we used this equation :

$$\tau_{pert} = D(q)\cdot\ddot{q} + C(q,\dot{q})\cdot\dot{q} + G(q) - \tau_{mot}$$

[4]at that time, the 'effort' measurement value seemed to be in mA, we took this value multiplied by a linear coeff for τ_{mot}

Because the acceleration is not measured, let's perform a static test (this way the acceleration is equal to zero).

We get the following **measurements** for $\alpha_1 = \alpha_2 = 0$ ('home position') :

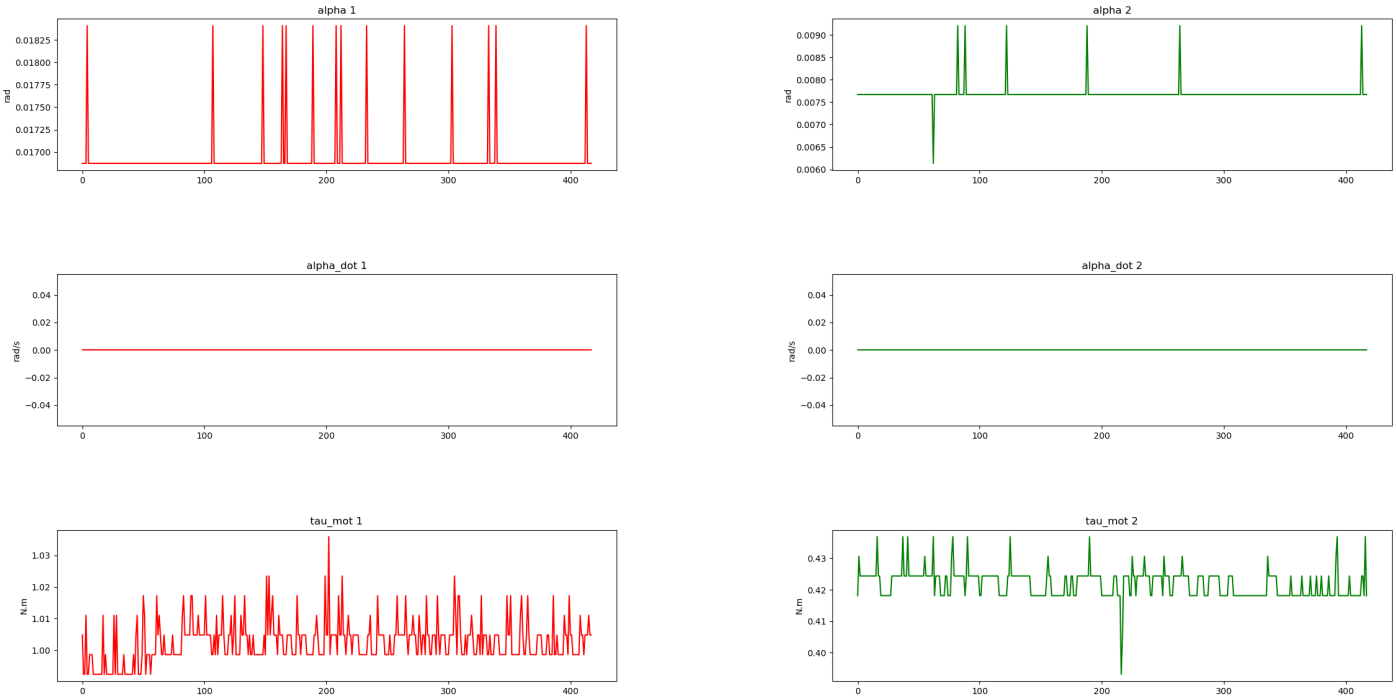

Figure 9: Measurements for a static test for the 2DOF arm

And for these measurements, using the equation of motion, we get the following **results** for the perturbations :

Figure 10: Calculated perturbations for a static test for the 2DOF arm

Analysis of these results :

The **Observation** of these calculated perturbation reveals notable inconsistencies.
Specifically, for joint 1, the obtained outcomes prove to be profoundly discordant, revealing perturbations that surpass even 100% of the motor torque, which is an unacceptable finding. On the other hand, the results for joint 2 exhibit tolerable margins of error, around 5%, suggesting that these results may be accurate.

How can we **explain** these results :
Firstly we realized the actuation of joint 1 involved the contribution of two servomotors rather than a single motor. This discovery potentially explain the discrepancies observed for the first joint. Nevertheless, substantial uncertainties persist regarding the precise interpretation of the measured 'effort' value (and that in each joint).

The intricacy inherent in treating a portion of a robot with six degrees of freedom as a two-degree-of-freedom arm inevitably introduces an error margin.
This approximation is especially critical, given that the pivot point of articulation 1, initially assumed as fixed, is, in reality, held in position by the waist and shoulder servomotors. In addition, the two links considered in this model consist of multiple actual links, held by 'internal' servomotors. These complexities, absent from the current model, undoubtedly contribute to the observed discrepancy.

However, isolating and accurately evaluating the individual influence of each error source remains impossible.
Moving forward in the analysis therefore requires a transition to a model that takes into account the true configuration of the robotic arm, — i.e. a robot with six degrees of freedom. This model revision aims to incorporate previously neglected parameters, and so it should better reflect reality.

Moreover, if the errors remain consistent despite this update, it is reasonable to attribute these errors to the transition from the measured 'effort' value to the actual torque applied in each link.

3.3 The Equation of motion of the six degrees of freedom robotic arm

To obtain the equation of motion of our six degrees of freedom robotic arm using the previous equation 6, we first have to identify the geometric and inertial characteristics of the robot (m_i, I_i and $R_{i/0}$). We then will have to calculate - as we did for the two degrees of freedom model - the terms for the kinematic energy : the Jacobian matrices (J_{vi} and $J_{\omega i}$).

The first step in order to establish a model of the robot is to set the frames attached to each link. For this, we will use the **Denavit-Hartenberg convention** (DH)[5] as it provides an efficient and systematic approach, making the whole process easier.

3.3.1 DH convention

We start by establishing the base frame R_0 attached to a fixed part and we define the subsequent frames based on the following assumption : axis x_i must be perpendicular to axis z_{i-1} and intersect it.

By applying this to our robot, we obtain the frames defined on the figure below :

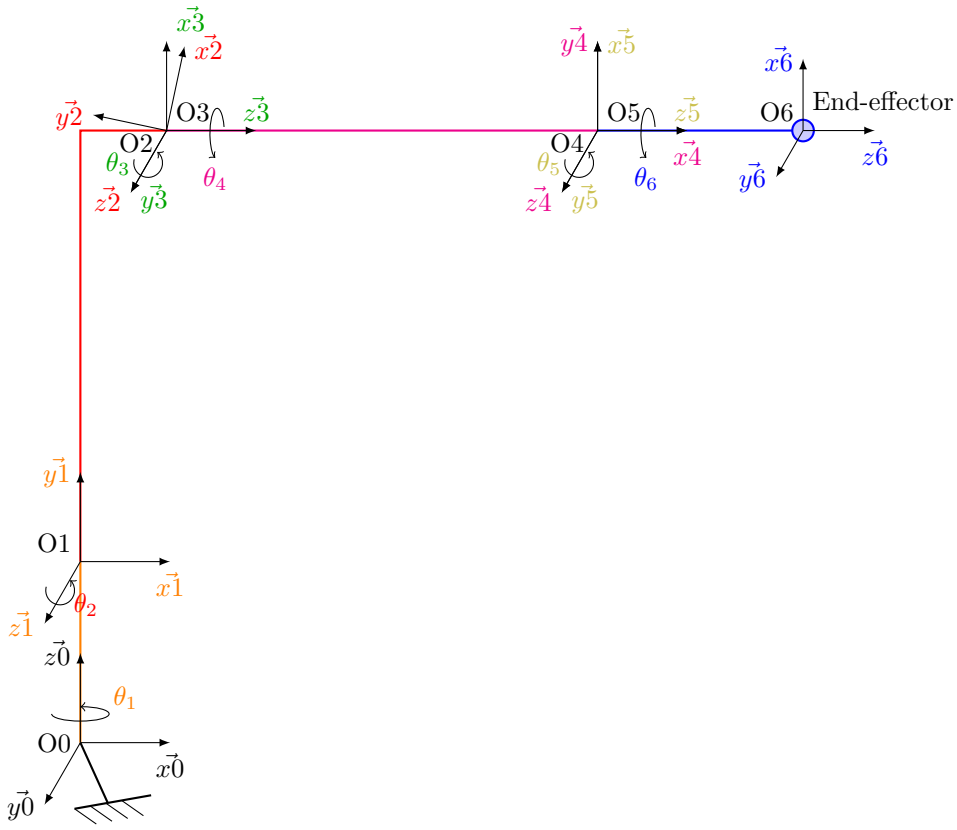

Figure 11: Kinematic diagram of the robot with frames established according to the DH convention

The DH convention defines four variables for each frame (let's call N_i the point of intersection between x_i and z_{i-1}):

- a_i is the distance from O_i to N_i along axis x_i
- d_i is the distance from O_{i-1} to N_i along axis z_{i-1}
- α_i is the angle between z_{i-1} and z_i around axis x_i
- θ_i is the angle between x_{i-1} and x_i around axis z_{i-1}

[5]To learn more about the DH convention and how to put it into practice, you can watch this video : Forward Kinematics Using the DH Convention Part 1

We can then write the DH-table [6] :

	θ_i	d_i	a_i	α_i
$R_0 -> R_1$	θ_1	d_1	0	$\pi/2$
$R_1 -> R_2$	$\theta_2 + \gamma_2$	0	a_2	0
$R_2 -> R_3$	$\theta_3 + \gamma_3$	0	0	$\pi/2$
$R_3 -> R_4$	θ_4	d_4	0	$-\pi/2$
$R_4 -> R_5$	θ_5	0	0	$\pi/2$
$R_5 -> R_6$	θ_6	d_6	0	0

Using this table, we can write the homogeneous transformation matrices[7] from one frame to the next (let's call them $T_{i/i-1}$) using this formula :

$$T_{i/i-1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We then can calculate $T_{i/j}$ with any i and j between 0 and 6 :

$$T_{i/j} = \begin{cases} T_{j+1/j} \cdot T_{j+2/j+1} \dots T_{i-1/i-2} \cdot T_{i/i-1} & \text{if } i > j \\ I & \text{if } i = j \\ \left(T_{j/i}\right)^{-1} & \text{if } i < j \end{cases} \tag{7}$$

Those matrices contain the rotation matrices from a frame to another and the translation vector between the origins of the frames (ie : geometric characteristics) :

$$T_{i/j} = \begin{bmatrix} R_{i/j} & t_{i/j} \\ 0 & 1 \end{bmatrix}$$

3.3.2 Calculating the Jacobian matrices

To calculate the **linear velocity Jacobian matrix**, we will first calculate the velocity of the centers of gravity of each link : each column j of J_{v_i} is the θ_j component of the velocity of the center of gravity of link i (expressed in the base frame) (cf. definition of J_v).

To calculate the velocity of link i (in the base frame), we will use the systematic formula :

$$\begin{aligned} \vec{V}(G_i \in i/0) &= \sum_{k=1}^{i} \vec{V}(G_i \in k/k-1) \\ \text{where} \quad \vec{V}(G_i \in k/k-1) &= \cancel{\vec{V}(O_{k-1} \in k/k-1)} + \overrightarrow{G_iO}_{k-1} \otimes (\dot{\theta}_k \vec{z}_{k-1}) \\ \text{and} \quad \overrightarrow{G_iO}_{k-1} &= \overrightarrow{G_iO_i} + T_{k-1/i} \cdot \vec{0} \quad \text{expressed in frame i} \\ \text{so in base frame :} \quad \vec{V}(G_i \in k/k-1) &= T_{i/0} \cdot (\overrightarrow{G_iO_i} + T_{k-1/i} \cdot \vec{0}) \\ \text{with} \quad T_{k-1/i} &= T_{i/k-1}^{-1} \end{aligned} \tag{8}$$

To calculate the **angular velocity Jacobian matrix**, we have to determine the axes of rotation $\vec{z}_i$: each first i column j of $J_\omega(i)$ contains the vector $\vec{z}_{j-1}$.

$$\text{To get the } \vec{z}_j \text{ axis we use this formula : } \vec{z}_j = R_{j/0} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{then} \quad J_\omega(i) = \begin{bmatrix} \vec{z}_0 & \dots & \vec{z}_{i-1} & 0 & \dots & 0 \end{bmatrix} \tag{9}$$

[6] the geometric constants are defined in Appendix - kinematic diagram

[7] To know more about homogeneous transformation matrices, check this video : Lecture 2 - 3: Homogeneous Transformations (Robotics UTEC 2018-1)

4 Implementation and evaluation of the model

It is almost impossible to establish the equation of motion of the six degrees of freedom arm without using a computer because the elements of the matrices are too long and too complex. That is why **we use python and symbolic calculation to compute the analytic form of the equation of motion of our robotic arm**. Once obtained, we will perform tests using the actual robot in order to evaluate the correctness and accuracy of our model.

4.1 Python implementation of the model

We want to establish the equation of motion of the robotic arm on its analytic form, this way we will be able to use it directly and won't have to recalculate it for each measurement sample.

To establish the analytic expression of the equation of motion on python, we will implement the calculation (from previous section) using **symbolic calculation with sympy** (python package).

The first step is to define the geometric and inertial constants of the arm and the vectors required to establish the EOM (m, I, $\overrightarrow{OG}$, $T_{i/j}$).

To define the describe the geometry of the robot, we need to define frames, the angles θ_1 to θ_6 are left as symbols using sympy, this means that we keep the analytic expression and don't use the numeric values.

The masses m_i and the Inertias I_i can be found in the URDF (Unified Robot Description Format) file of the robotic arm, the $\overrightarrow{O_iG_i}$ vectors are obtained by transposing in DH-frames the vectors found in the URDF file, finally the transformation matrices from frame i to frame j : $T_{i/j}$ are calculated using (7)

It is then easy to compute the Jacobian matrices using the formulas from the previous part (3.3.2). We then calculate the kinetic and potential energies using equation (2) and (3). The D matrix is derived directly from the kinetic energy, and the G matrix is obtained by deriving the potential energy according to the different degrees of freedom using the derivation function offered by the sympy module: 'diff'. Finally, we calculate the matrix C by deriving the terms of matrix D.

The D, C, G matrices are transformed into functions of the vector q and $\dot{q}$, which means that when we will evaluate those functions, the symbols θ_i will be replaced by their numeric values contained in the vector q. The functions are saved into a ".pkl" file using the python package cloudpickle and can now be loaded and used directly in any file.

For example : we can perform a measurement (of θ_i, $\dot{\theta}_i$ and i), create the vectors q and $\dot{q}$ with the measured angles and angular velocity and then calculate the values of the D, C, G matrices using the saved functions. To use the complete equation of motion, we can derive $\ddot{q}$ from $\dot{q}$ and τ from i[8].

cf. appendix 3 for informations on calculation time of the matrices

4.2 Evaluation of the model

Once the equation of motion established, we need to evaluate the correctness of it, which means to what extent does it describe the behavior of the actual robotic arm accurately.

4.2.1 Evaluating the error of the model

To assess the accuracy of our model, we can try to **measure the error** it makes compared to the reality.

We're going to compare values calculated by the model with experimental values measured in the actual robotic arm.

To set up such an evaluation, we'll need to perform a test with the actual robotic arm and record its state measured during this test. In parallel, we'll need to run a simulation of the robot's behavior using our equation of motion (for example with Runge-Kutta simulation method) and record the state calculated during the simulation.

[8] cf. section 4.2.2

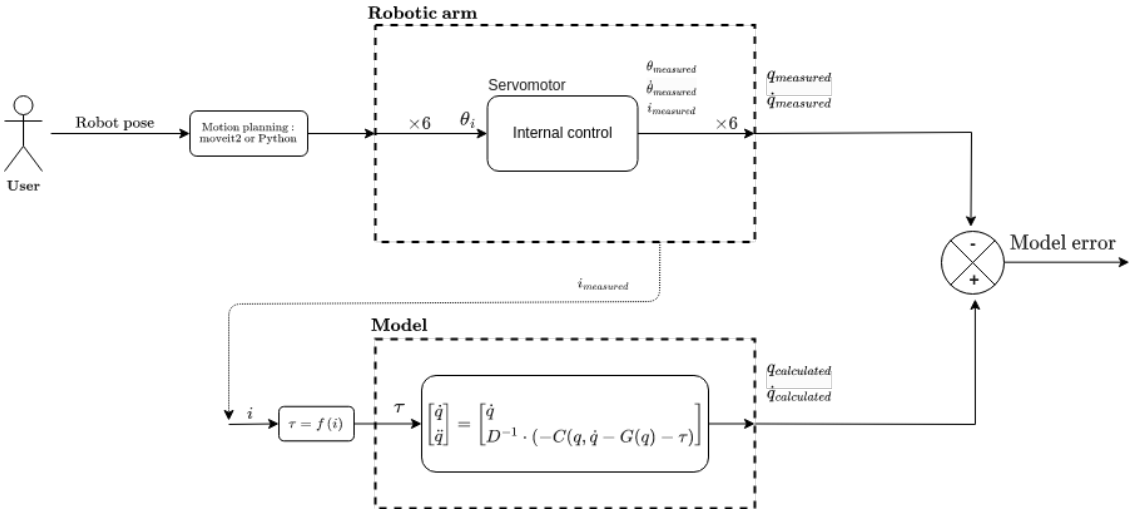

Figure 12: Measurement of model error

The results obtained by calculation are then compared with the experimental results.

For this test to be relevant, **we need to perform the simulation and the test with the actual robot under the same initial conditions and with the same inputs at each time**.

The problem is that for the actual system, the input is a trajectory, whereas for the model we've built, the input is a torque command for each joint.

We don't have access to the torque in the real robot, but the current applied to each motor is measured, so we just need to find a relationship between torque and current to be able to run the simulation with the same controls as the real robot.

4.2.2 The torque/current relationship

We want to determine the relationship between the measurement "effort" i and the actual torque τ applied by the motor in the joint.

There are two ways to do this : either use the theoretical torque/current relationship given in the documentation of the servomotors or use the previously established equation (with $\ddot{q}$ derived from the measured $\dot{q}$) to evaluate τ and compare it to the measured i. The first one corresponds to the behaviour of the servomotor isolated from the robot measured under conditions quite different from the one in which it is used here, however the general behaviour should be the same. The second contains only the model's inaccuracies.

We assume the relationship between the current applied to the servomotor and the torque it delivers to be a **linear relationship** [9]

$$\tau = K_c \cdot i + \tau_0$$

cf. doc of the motors : performance graph xm540-w270 and performance graph xm430-w350 with those values (obtained with a linear regression):

Servomotor model	K_c	τ_0
XM540-W270	2.197	0.124
XM430-W350	1.835	−0.259

NB :

- The "effort" value published on the joint_states topic is 2.69 times the present current (which is the value of "current" in the performance graph).
- Joints 2 and 3 are actuated by dual motors, resulting in torque values twice as high compared to single motor.

[9] cf. Appendix - current/torque relationship

- This relationship is an approximation, it does not take into account certain phenomenon and is not accurate in some situations (for example, a probable non-linearity around 0 and differences in behavior depending on the resistive load).

Lets consider that the model is correct to a certain extent.

To check the accuracy of the chosen linear relationship for the chosen recording, we now display the measured current as a function of calculated torque alongside with the theoretical linear curve :

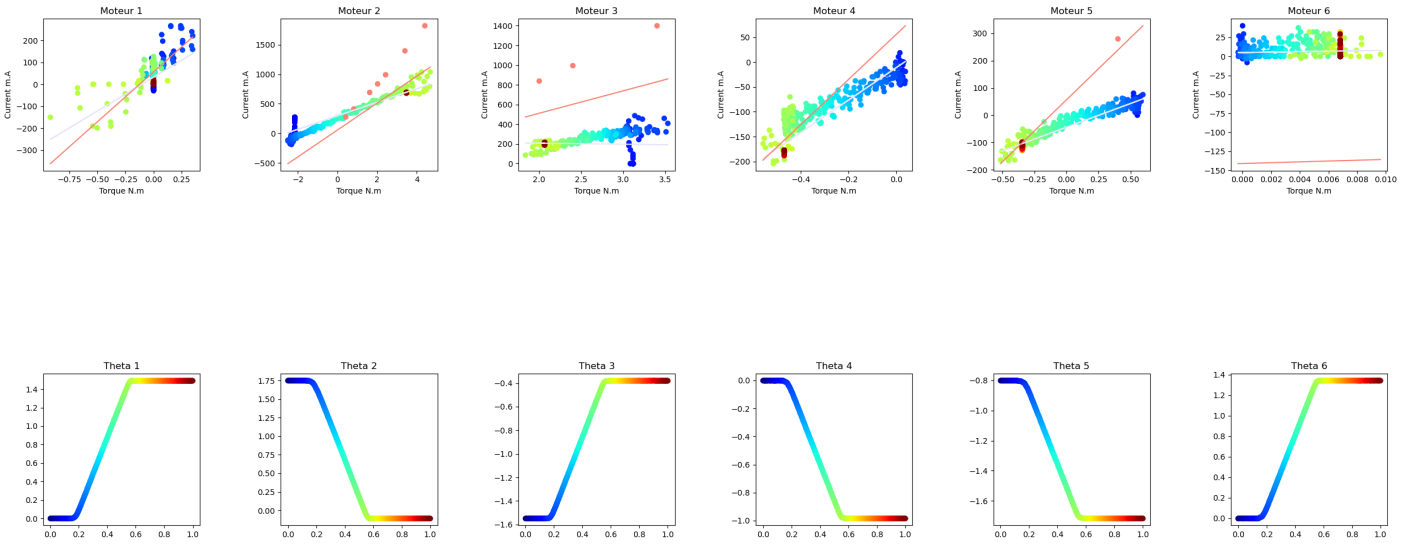

Figure 13: Image of the graphs of current (measured) as a function of torque (calculated)

On the top figures, you can compare the salmon curve which is theoretical with the lavender one which corresponds to the model:

For each measurement (point), the torque is calculated using the equation of motion (with the values of angle and angular velocity from the recording) and the current is the measured value; the lavender curve is generated with a linear regression using those points. The color indicates time, you can compare it with the graphs displayed above which represent the trajectories of each joint during the recording.

Note that the range of torque needs to be wide for the comparison to be relevant, which is not the case for most joints when doing a test at low speed (the default speed is low). That is why those values correspond to a test performed at high speed.

We can see that **the relationship appears to be linear, but each motor seems to have a different slope and offset**, and none of them seems to correspond exactly to the theoretical relationship.

This also constitute an evaluation of the model as we compare the results obtained by the model with experimental curves.

The model appears to be relatively correct, since experimental behavior is quite similar to theoretical behavior. However, we can't yet measure the accuracy of the model in this way, since the theoretical relationship doesn't seem to describe the engine behavior quite correctly.

4.2.3 Future Directions for Evaluation

We can consider different methods to go further in the evaluation:

- The simplest approach would be to add torque sensors to each joint (for example, Strain Gauge Torque Sensors), as this would provide direct access to the torque, which is the input to our model.
- We can also refine the current-to-torque relationship by conducting tests under various conditions for each joint. These tests could include precise tests at very low torque to observe

behavior around zero, tests in both directions, and tests with a load in the robot's hand. By simply adding this mass to the model, we can assess the behavior of the joints with higher torques, especially in the joints near the end-effector.

- Another option would be to rely on the relationship between the motor's rotation speed and torque, which would require access to the motor speed trajectory curves.

5 Conclusion

5.1 Technical conclusion

By leveraging the Euler-Lagrange equation and following the systematic Denavit-Hartenberg convention, we successfully derived the motion equation for a robotic arm. This equation was implemented in Python in its analytical form, thanks to symbolic calculations. Using trajectory controls provided by the robot and measurements from each motor, we conducted tests to assess the precision of our model, which yielded satisfactory results in its initial evaluation.

Looking ahead, we could to continue refining and evaluating this model further. It will serve as the foundation for establishing an efficient control system for the robotic arm. Furthermore, the model offers the flexibility to accommodate additional constraints, such as objects manipulated by the end effector, or to address specific requirements in various applications.

5.2 What I've learned from this internship

- Giving presentations on the work carried out in English and in front of people with knowledge in fields more or less distant from robotics
- Dealing with the difficulty of finding certain information on an already existing robot
- Adapt to the sensors present on the robot and the possible control modes, which may be different from the variables we need
- Work completely independently to make progress and deal with problems

Acknowledgement

I would like to express my gratitude first and foremost to my teacher, Luc Jaulin, who passed on this internship offer and made this enriching experience possible. I would also like to thank my internship tutor, Andreas Rauh, for opening the doors of his team and creating pleasant working conditions. I would also like to express my gratitude to the entire team at the Computer Science Department at the University of Oldenburg, who welcomed me and with whom we spent these four months. Last but not least, I'd like to thank Oussama Benzinane for his invaluable assistance, his sound advice and, of course, his wonderful tajine, which brought sunshine in the face of the German rain.

6 Appendix

6.1 Appendix 1 : More about hardware specifications

ViperX-300 6DOF	
Degrees of Freedom	6
Reach	750mm
Total Span	1500mm
Repeatability	1mm
Accuracy	5 - 8mm
Working Payload	750g*
Total Servos	9
Wrist Rotate	Yes

ID	Joint Name	Servo	Baudrate
1	waist	XM540-W270	1Mbps
2	shoulder	XM540-W270	1Mbps
3	shoulder_shadow	XM540-W270	1Mbps
4	elbow	XM540-W270	1Mbps
5	elbow_shadow	XM540-W270	1Mbps
6	forearm_roll	XM540-W270	1Mbps
7	wrist_angle	XM540-W270	1Mbps
8	wrist_rotate	XM430-W350	1Mbps
9	gripper	XM430-W350	1Mbps

Figure 14: Robotic arm specifications

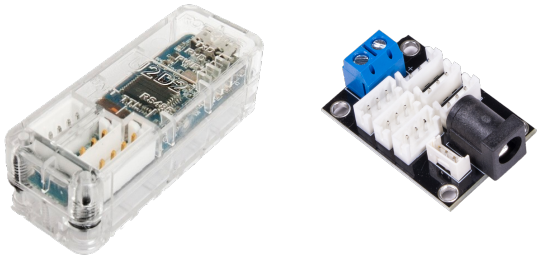

Figure 15: Connectic of the robot

The U2D2 controller (left) is a USB to TTL converter that allows for the control of the serial line of DYNAMIXEL servos using a computer. The 6 Port, 3 Pin XM/XL Power Hub allows to provide power to the DYNAMIXEL daisy chain, here in 12V.

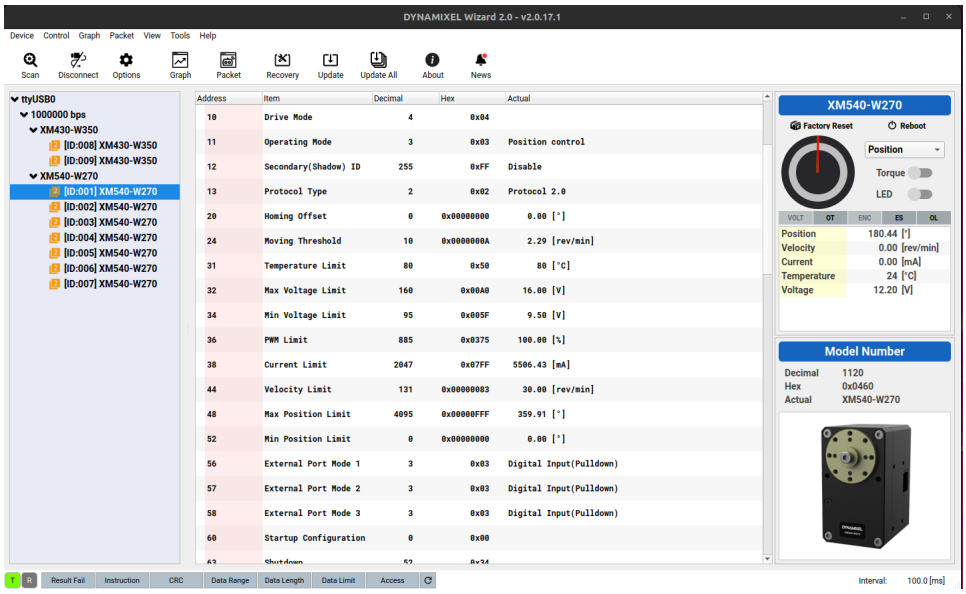

Figure 16: Dynamixel Wizard 2.0

This software makes it easy to view and configure the registers of all the motors attached to the U2D2 from a graphical interface. It also allows to perform tests and measure performance of a single motor.

Appendix 2 : development of Euler Lagrange application

$$\begin{cases} \frac{\partial L}{\partial \dot{q_k}} = \sum_j d_{kj} \cdot \dot{q_j} \Rightarrow \frac{d}{dt}\frac{\partial L}{\partial \dot{q_k}} = \sum_j d_{kj} \cdot \ddot{q_j} \ + \ \sum_{i,j} \frac{\partial d_{kj}}{\partial q_i} \cdot \dot{q_i} \cdot \dot{q_j} \\ \frac{\partial L}{\partial q_k} = \frac{1}{2} \sum_{i,j} \frac{\partial d_{ij}}{\partial q_k} \cdot \dot{q_i} \cdot \dot{q_j} - \frac{\partial P}{\partial q_k} \end{cases}$$

$$\text{and : } \sum_{i,j} \frac{\partial d_{kj}}{\partial q_i} \cdot \dot{q_i} \cdot \dot{q_j} = \frac{1}{2} \sum_{i,j} \left(\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} \right) \cdot \dot{q_i} \cdot \dot{q_j}$$

$$\Rightarrow \sum_{i,j} \left(\frac{\partial d_{kj}}{\partial q_i} - \frac{1}{2} \frac{\partial d_{ij}}{\partial q_k} \right) \cdot \dot{q_i} \cdot \dot{q_j} = \sum_{i,j} \frac{1}{2} \left(\frac{\partial d_{kj}}{\partial q_i} \ + \ \frac{\partial d_{ki}}{\partial q_j} \ - \ \frac{\partial d_{ij}}{\partial q_k} \right) \cdot \dot{q_i} \cdot \dot{q_j} = \sum_{i,j} c_{ijk} \cdot \dot{q_i} \cdot \dot{q_j}$$

Appendix 3 : calculation time

The calculation time to calculate the matrices can be quite long (as their elements are very long expressions). Moreover, the evaluation of the matrices generated can be time-consuming, that cannot be overlooked.

The sympy package has various functions for simplifying the analytic expressions[10], this feature allows to reduce the time to evaluate the matrix.

However, it's important to note that this process may require some time or even lead to recursion limit errors.

Transforming the matrices into functions (using lambdify()) also takes some time, especially for the more complex matrices (C and D).

→ Using non-simplified inertia matrices and minimal simplifications (only V, Jv, Jω and T), the file eom_6dof.py took 4min15 to execute. The evaluation of the three saved matrices takes about 0.2seconds (that is quite long as we will want to evaluate the eom hundreds of times per recording).

→ Using simplified inertia matrices (by approximating negligible values to 0), it took 3min45 for an evaluation time of 0.11s.

The amount of simplification achievable using sympy features is restricted due to the considerable length of the expressions in our matrices. Since the simplification relies on recursive methods, we often encounter recursion limit errors when attempting to simplify such extensive expressions.

→By maximising the simplifications made, we obtain an evaluation time of 0.08s for the matrices (that makes 1.6sec of treatment for each second of recording). But the execution of the file eom_6dof.py took 1 hour.

To achieve a significantly shorter evaluation time, we can simplify the inertia matrices further (at the risk of losing precision).

6.2 Appendix 4 : Visualize data

The motion_visu_6dof.py file allows to visualize both recorded measurements and computed quantities derived from these measurements using the equation of motion.

When executing this file, you will be prompted to select the type of recording you want to visualize and then enter the number of the recording - you can now visualize some figures.

NB:

- The calculated values are saved, eliminating the need for recalculation when visualizing the same recording in the future.
- To visualize different figures, we can modify the "array_names" variable.
- The acceleration in each joint is calculated using two different methods : one using the equation of motion, and the other by directly differentiating the speed.

It's important to note that the measurement of the "effort" represents the current in the joint rather than the torque. We approximate the relationship between the current and the torque to be linear (cf 4.2.2)

[10] we will only use trigsimp() as we have trigonometric expressions

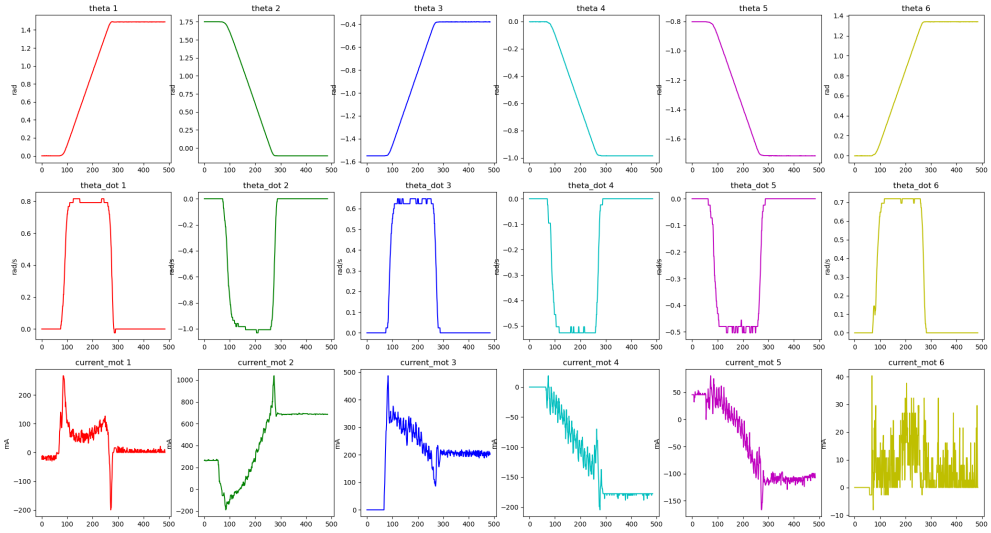

Figure 17: Image of a motion_visu_6dof.py graph

To check the accuracy of this relationship for the chosen recording, we display the current as a function of torque as in fig **??**. We also display the values of torque calculated using the eom and using the theoretical relationship from the doc over time to compare them.

Appendix 5 : other features

The "test.py" file provides additional functionalities:

- Replay the recording in rviz
- Conduct simulations of the behaviour of the robotic arm using the EOM. The simulation uses Euler or Runge-Kutta 4 method. You can visualize the simulated trajectory with the kinematic model in matplotlib or with rviz.
- Visualize the different components of the EOM (D,C,G) over time, allowing for comparison and verification.
- Crop a recording if there are some useless parts at the beginning and/or the end of it.
- Perform calculation time tests with access to the matrices and the measurements.