Assistant Engineer Internship

Luc-André TERRINE

Supervisor: Andreas Rauh
Address: Carl Von Ossietzky University, Oldenburg, Germany

October 16, 2024

Abstract

As part of my formation at ENSTA Bretagne, I was required to complete an internship in a foreign country as an engineering assistant. I was given the opportunity to fulfill this requirement at the University of Oldenburg, where I joined their research team. My work focused on optimizing the code for computing the equation of motion of a robotic arm, with the goal of enabling the forward dynamics simulation of its multiple degrees of freedom.

Résumé

Dans le cadre de ma formation a l'ENSTA Bretagne, j'étais tenu de réaliser un stage a l'étranger en tant qu'assistant ingénieur. J'ai eu l'opportunité de remplir cette obligation a l'Université d'Oldenburg, oú j'ai rejoint leur équipe de recherche. Mon travail a consisté á optimiser le code de calcul de l'équation de mouvement d'un bras robotique, dans le but de permettre la simulation de la dynamique directe de ses multiples degrés de liberté.

Contents

Introduction

0.1 The ViperX 300 S

The ViperX 300 S is a robotic arm equipped with a gripper for object handling. It is programmable using Python and ROS2, allowing for control and integration with other systems. For this project, the robot is also equipped with a camera connected to a PC, providing visual data to guide object detection and manipulation.

Figure 1: The ViperX 300 S

0.2 Objectives

The goal of the project was to program the ViperX 300 S to autonomously grasp a container and pour liquid into a glass. This required ensuring that the arm's movements were coordinated with the real-time visual data from the camera to allow for accurate positioning and manipulation.

I started from an existing base code that computed the robot's equations of motion (EOM). My task was to optimize this code to reduce computation time, ensuring the information

processed by the camera could be used effectively by the motion algorithm, maintaining a smooth flow of information between vision and movement.

Chapter 1

Equation of motion

1.1 Establishing the Equations of Motion (EOM)

The equations of motion for the **ViperX 300 S** robotic arm are derived using the *Euler-Lagrange formalism*, which provides an efficient way to model the dynamics of multi-degree-of-freedom systems. The arm has six rotational degrees of freedom (DoF), and the following generalized Euler-Lagrange equation applies to each DoF:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} = \tau_k$$

Where:

- $L = K - P$ is the **Lagrangian**, defined as the difference between the kinetic energy K and potential energy P.
- q_k are the generalized coordinates representing joint angles:

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

- τ_k are the generalized torques applied to each joint:

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

The kinetic energy K of the system is expressed as:

$$K = \frac{1}{2}\dot{q}^T D(q)\dot{q}$$

Where $D(q)$ is the **inertia matrix**. It describes the relationship between the velocities $\dot{q}$ of the robot's joints and the kinetic energy. $D(q)$ is a function of the robot's configuration (i.e., the joint angles q) and encapsulates the mass distribution and geometrical properties of the robot's links.

The potential energy P is given by the sum of the gravitational forces acting on each link:

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

$$\sum_{i=1}^{n} d_{ij}(q) \cdot \ddot{q}_j + \sum_{i,j} c_{ijk}(q) \cdot \dot{q}_i \cdot \dot{q}_j + g_k(q) = \tau_k$$

Where:

- $d_{ij}(q)$ are the elements of the inertia matrix $D(q)$, which encapsulates the inertial properties of the system.
- $c_{ijk}(q)$ represents the Coriolis and centrifugal forces affecting the system.
- $g_k(q)$ is the gravitational vector, representing the gravitational forces acting on the system.
- $\ddot{q}_j$ is the joint acceleration, $\dot{q}_i$ and $\dot{q}_j$ are joint velocities.
- τ_k are the torques applied at each joint.

We can now express this equation in matrix form as:

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

Where:

- $D(q)$ is the inertia matrix.
- $C(q,\dot{q})$ is the Coriolis and centrifugal matrix.
- $G(q)$ is the gravitational vector.
- τ is the vector of torques applied to the joints.

Note: The derivation of the equations of motion and the associated matrices $D(q)$, $C(q,\dot{q})$, and $G(q)$ were adapted from the research of Léo Bernard, University of Oldenburg (2023).

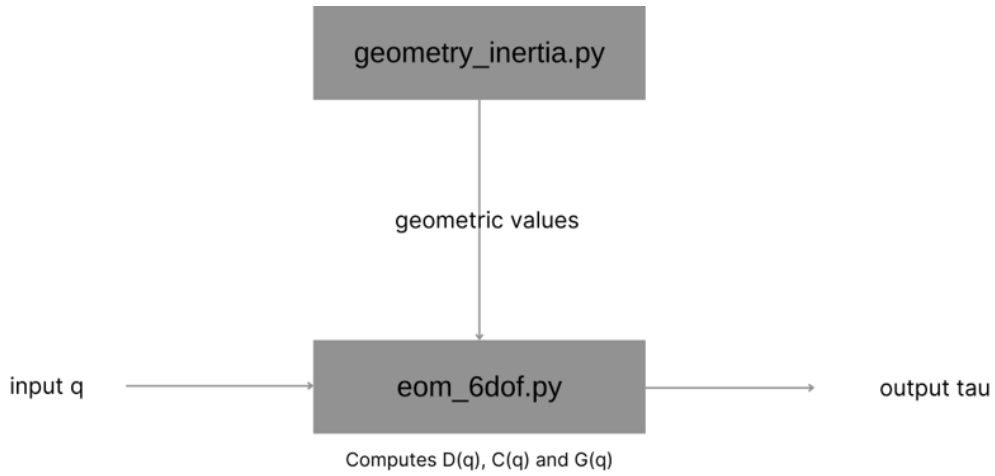

Figure 1.1: Program structure diagram

1.2 Computing the Equations of Motion

To compute the equations of motion for the robotic arm, we needed a method to calculate the terms of the required matrices using a Python algorithm. The structure of the program is illustrated in the following diagram:

All the geometrical and physical properties of the robot are stored in a Python script called `geometry_inertia.py`. This file contains values such as the **inertia matrices**, the **mass** of each part of the robot, and the **transformation matrices** between the different reference frames of each robotic link.

Initially, the program followed a symbolic approach for computing the matrices. Using the Sympy library, symbolic variables were defined to represent the angles of the robotâs degrees of freedom and their derivatives.

```
theta1, theta2, theta3, theta4, theta5, theta6 = symbols("θ1 θ2 θ3 θ4 θ5 θ6")
theta1_dot, theta2_dot, theta3_dot, theta4_dot, theta5_dot, theta6_dot = symbols("θ1_dot θ2_dot θ3_dot θ4_dot θ5_dot θ6_dot")
```

Figure 1.2: Symbolic definition of joint angles and their derivatives

These symbolic elements were then loaded into the `eom_6dof.py` program, which took these inputs and computed the **D**, **C**, and **G** matrices in symbolic form. However, this symbolic approach led to significant computational delays, as the expressions for these matrices can contain hundreds of complex terms, involving many iterations of trigonometric functions.

According to the documentation, the base program took up to **1 hour** to compute the **D**, **C**, and **G** matrices, and their evaluation time could take **1.6 seconds** for every second of recorded data. In practice, however, I encountered an issue where the program would never complete its calculations, even after being left to run for multiple nights.

My task was to optimize this program to make it feasible for real-time use with the robotic arm.

1.2.1 Addressing the First Issue: Trigonometric Simplification

The first objective was to identify the functions in the program that caused unreasonably long calculations. By adding time checks at key points in the code, I determined that the `trigsimp` function (used to simplify trigonometric expressions) was causing the program to hang when processing highly complex expressions.

To address this, I developed a new function called `safe_trigsimp`. This function attempts to simplify the expression for a set time and aborts the operation if the time limit is exceeded. This adjustment ensured that the program would no longer get stuck indefinitely.

```python
### Tries calling the trigsimp function, aborts if the instruction takes too long to avoid infinite loops
def safe_trigsimp(expr, timeout_duration=5):
    tstart=time()
    # Set an alarm for the timeout duration
    signal.alarm(timeout_duration)
    try:
        # Try to simplify the expression
        print("trigsimp start")
        result = trigsimp(expr)
    except RecursionError:
        result=expr
        print("trigsimp hit recursion limit and was termnated")
    except TimeoutException:
        # If a timeout occurs, move on
        result = expr
        print("trigsimp took too long and was terminated.")
    finally:
        # Disable the alarm
        signal.alarm(0)
    print ("Time taken {:.2f} seconds.".format(time()-tstart))
    return result
```

Figure 1.3: The `safe_trigsimp` function

With this issue resolved, the next step was to find additional optimization methods to reduce the overall calculation time.

1.2.2 Reducing Trigonometric Function Calls

The next goal was to reduce the number of trigonometric function evaluations in the symbolic matrices. To achieve this, I created new symbolic variables to represent trigonometric values of the joint angles. These new symbols replaced the multiple function calls within the matrices, allowing for more efficient computation.

This was implemented using a **dictionary** that stored the relationships between the symbolic variables and their corresponding trigonometric function calls. After computing the **D matrix**, which is the simplest and is used to construct the terms of the **C matrix**, these replacements were applied to streamline the calculation process.

1.2.3 Optimization of the C Matrix Calculation

The computation of the C matrix is notably the most complex among the three matrices $D(q)$, $C(q, \dot{q})$, and $G(q)$. This complexity arises because the C matrix depends on both the

```
thetai = symbols("θi")

thetai_dot = symbols("dθi")

s_thetai, c_thetai = symbols ("s_θi c_θi")

s_thetai_dot, c_thetai_dot = symbols ("s_dθi c_dθi")

replacement = {
    sin(thetai) : s_thetai,
    cos(thetai) : c_thetai,
    sin(thetai_dot) : s_thetai_dot,
    cos(thetai_dot) : c_thetai_dot
}
```

Figure 1.4: Dictionary mapping trigonometric functions to symbolic variables

joint angles and their derivatives. The terms of the C matrix are computed using the following formula:

$$c_{ijk} = \frac{1}{2}\left(\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k}\right)$$

In the original code, the calculation of the C matrix was performed in a nested loop structure as shown below:

```
C = Matrix(np.zeros((n_dof,n_dof)))  # n_dof is equal to 6
for i in range(n_dof):
    for j in range(n_dof):
        Cki = 0
        for dof in range(1, n_dof+1):
            k = dof - 1
            cijk = (1/2) * (diff(D[k,j], q[i]) + diff(D[k,i], q[j]) - diff(D[i,j], q[k]))
            Cki += cijk * q_dot[j]
        C[k,i] = Cki
```

This approach, however, led to inefficiency. The code was structured to calculate three different partial derivatives of the same column of the D matrix within three nested loops, which resulted in unnecessary calculations being repeated, consuming significant computational resources.

To optimize this process, I introduced a new approach that precomputes the derivatives of the D matrix and stores them in a 3D array. This allowed for faster retrieval of these terms during the calculation of the C matrix, avoiding redundant calculations.

The optimized version starts by computing and storing the derivatives of the D matrix in a 3D Jacobian array:

```
# Creates a 3D Jacobian matrix of D derivatives to simplify the computation of C
div_matrix = np.zeros((n_dof, n_dof, n_dof), dtype=object)
for i in range(n_dof):
    for j in range(n_dof):
        print("Coefficient", i, j)
        for k in range(n_dof):
            div_matrix[i,j,k] = diff(D[i,j], q[k])


            # Replaces symbols with trigonometric functions
            div_matrix[i,j,k] = expression_sub(div_matrix[i,j,k], replacement)
```

By storing the partial derivatives in `div_matrix`, we can directly fetch the required terms during the calculation of the C matrix. This approach significantly reduced the computation time:

```
C = Matrix(np.zeros((n_dof, n_dof)))
for i in range(n_dof):
    for j in range(n_dof):
        print("ck:", i, j)
        Cki = 0
        for dof in range(1, n_dof+1):
            k = dof - 1
            cijk = (1/2) * (div_matrix[k,j,i] + div_matrix[k,i,j] - div_matrix[i,j,k])
            Cki += cijk * q_dot[j]
        C[k,i] = Cki
```

This method greatly reduced the number of repetitive calculations by precomputing the derivatives, thus streamlining the evaluation of the C matrix. This optimization was a key step in improving the overall efficiency of the algorithm for calculating the equation of motion.

1.2.4 Optimization Results

The optimization techniques applied to the symbolic computation of the D, C, and G matrices yielded significant improvements in runtime and computational efficiency. Initially, the program used a symbolic approach to define and compute the matrices, which resulted in extensive calculation times due to the complexity of the trigonometric and algebraic expressions involved.

The optimizations were achieved through two primary strategies:

- Reducing redundant calculations by storing all derivative terms of the inertia matrix $D(q)$ in a 3D array, allowing for efficient access to these terms during the computation of the Coriolis matrix $C(q, \dot{q})$.

- Simplifying the symbolic expressions by introducing a time-limited trigonometric simplification function, ensuring that overly complex expressions do not stall the program.

After implementing these methods, the symbolic matrices were transformed into numerical functions using the `lambdify` method from the `SymPy` library, which allowed the matrices to be evaluated directly as functions of q and $\dot{q}$.

1.2.4.1 Time Reduction for Matrix Construction

The most significant improvement was observed in the total runtime of the `eom_6dof.py` script. Before optimization, constructing the matrices took approximately 1 hour. After optimization, this time was reduced to 17 minutes â a reduction of over 70%.

1.2.4.2 Evaluation Time Improvement

In addition to matrix construction, the evaluation of the matrices (i.e., converting symbolic expressions into numerical values during real-time operation) was greatly optimized. The evaluation time for the D, C, and G matrices after optimization was only 9% of the time required before optimization, making the system more responsive to real-time inputs from the camera and sensor feedback.

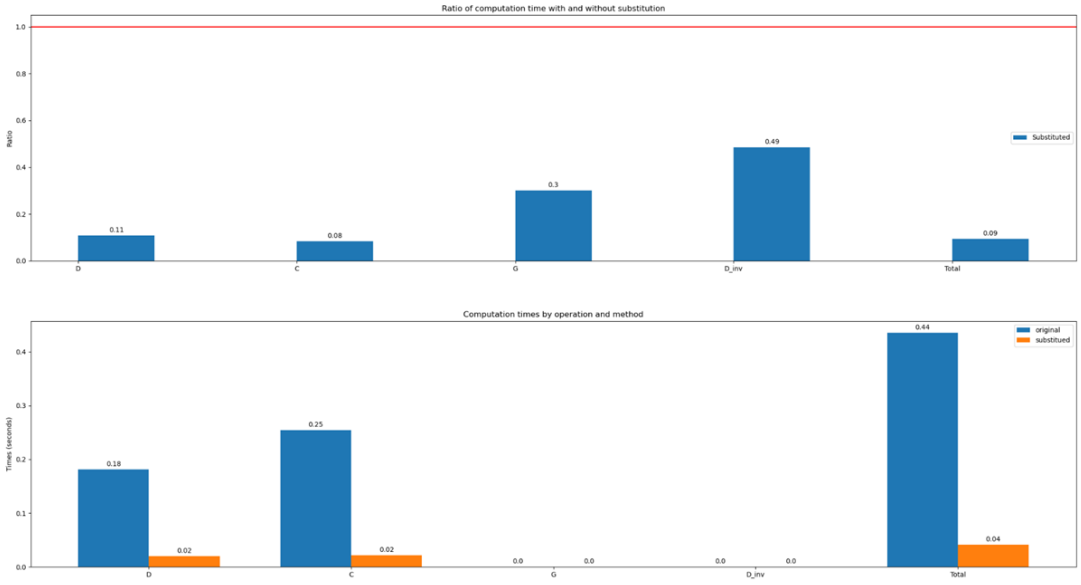

Figure 1.5: Graph of the result of the optimization of the program in term of time

The reduced computation and evaluation times directly impacted the performance of the robot's control system. Prior to optimization, the lengthy computation times hindered the robot's ability to process information from the camera and execute movements in real-time.

By improving the efficiency of matrix calculations, the robot can now exchange information with the camera and other sensors in a timely manner. This ensures smoother and more responsive motion control, making the system viable for real-time operations like object manipulation, as initially intended in the project.

1.2.5 Conclusion of the Optimization Phase

The optimizations carried out on the symbolic computation process were crucial for enabling the robotic arm to function effectively within the project's real-time constraints. By reducing both the matrix construction time and the evaluation time, the robot is now capable of integrating the camera data into its movement calculations. This improvement makes it possible for the robot to execute complex tasks, such as grasping objects and performing precision movements, in a more reliable and efficient manner.

Chapter 2

Integration Model for Forward Dynamics

In this chapter, we discuss the process of developing and refining a model for the forward dynamics of the robotic arm to compute the state vector q (degrees of freedom) from the joint torques τ. This was a challenging task that required several iterations to find a suitable approach, starting from the base matrix equation of motion.

2.1 Initial Forward Dynamics Model

We remind that the components of the state vector q are represented by

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

and its derivative is given by

$$\dot{q} = \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{pmatrix}.$$

We began with the robot's equation of motion in its standard form:

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

From this equation, we derived the expression for the acceleration of the state vector q as follows:

$$\ddot{q} = D(q)^{-1} \left(\tau - C(q, \dot{q})\dot{q} - G(q)\right)$$

To facilitate numerical integration, we defined the state vector Q as:

$$Q = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix}$$

where:

$$Q_1 = q \quad \text{and} \quad Q_2 = \dot{q}$$

The derivatives of these components are:

$$\dot{Q}_1 = Q_2$$

$$\dot{Q}_2 = D(q)^{-1} \left(\tau - C(q, \dot{q})Q_2 - G(q)\right)$$

With these elements in place, we used the `odeint` function to numerically integrate the equations of motion, using the initial conditions of q and the torque τ as inputs.

2.2 Initial Results and Issues

Despite setting up the integration process based on the equations, the results indicated a significant flaw in the model. The computed solutions tended to diverge rapidly as soon as they started increasing in value, as shown in the graph below.

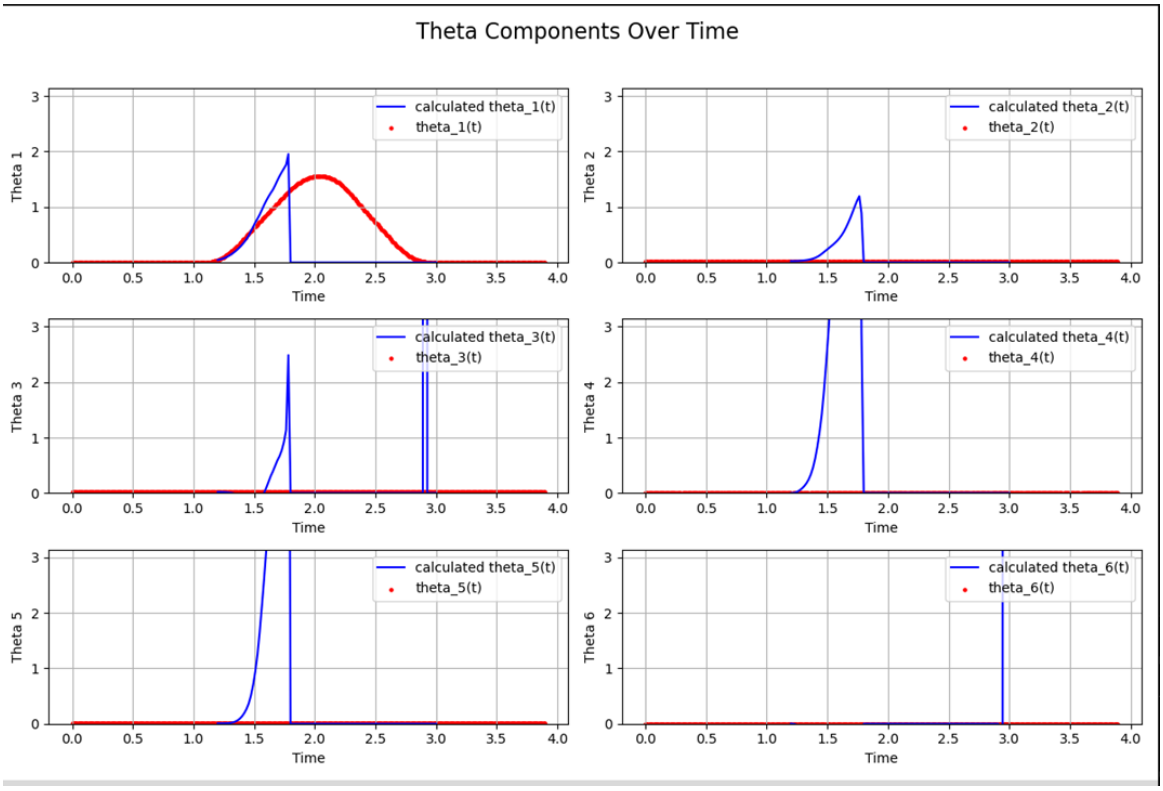

Figure 2.1: Computation of the theta components over time

Its derivative is shown in the following graph:

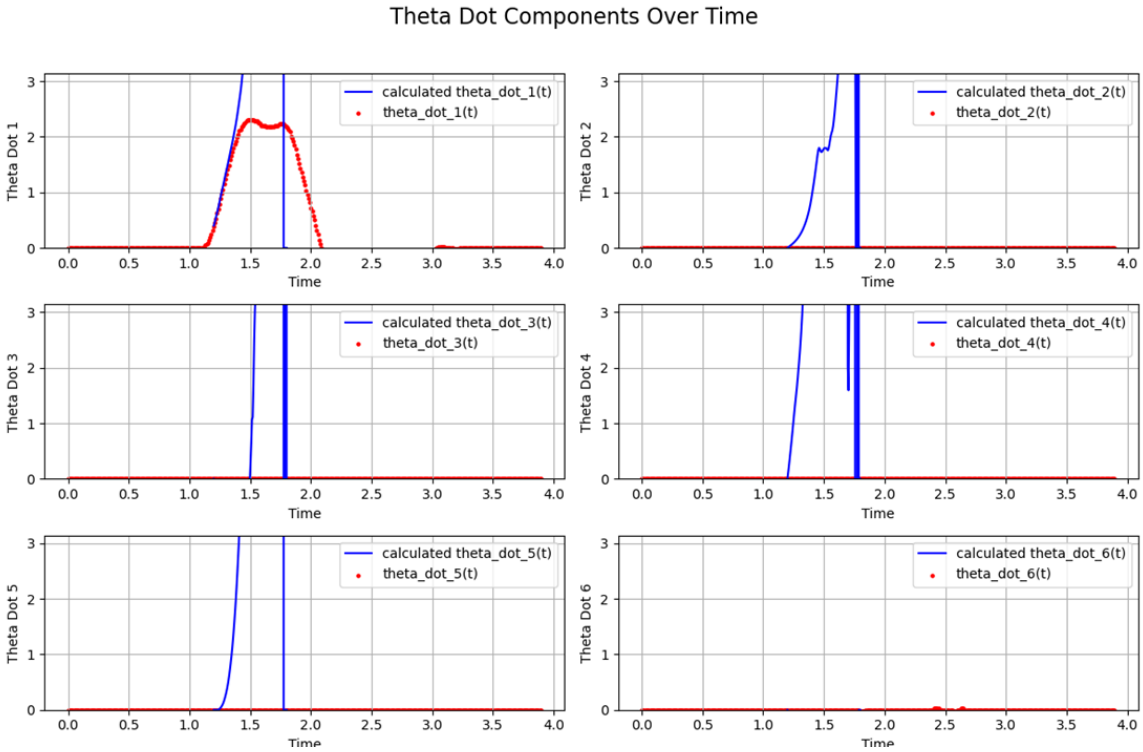

Figure 2.2: Computation of the theta dot components over time

In these graphs, the theoretical model of the joint angles is shown in red, while the computation of the forward dynamics is shown in blue.

This divergence suggested that our model failed to account for certain physical phenomena, specifically the friction forces acting on the system. Without incorporating friction, the values grew uncontrollably, resulting in an unrealistic simulation of the robot's motion.

2.3 Incorporating Friction into the Model

To address this issue, we modified the equation of motion to include friction forces:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau - \tau_{\text{friction}}(\dot{q})$$

We defined the frictional torque $\tau_{\text{friction}}(\dot{q})$ as a linear function of velocity:

$$\tau_{\text{friction}_i}(\dot{\theta}_i) = a_i \cdot \dot{\theta}_i + b_i$$

for each of the six components of the q vector.

However, in practice, the friction behavior depended on the sign of the velocity ($\dot{q}$), leading to different linear equations for acceleration and deceleration. We formulated these conditions as follows:

$$\text{If } \ddot{\theta} > 0, \quad \tau_{\text{friction}_i} = a_i^{+} \cdot \dot{\theta}_i + b_i^{+}$$

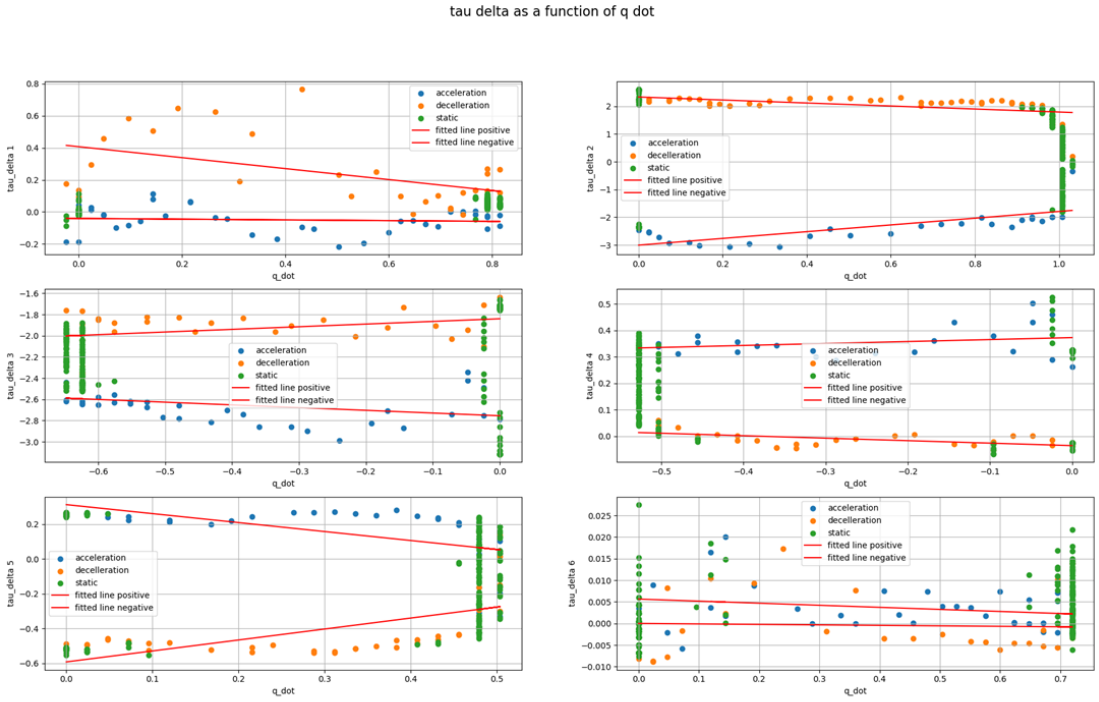

Figure 2.3: Representation of the friction torque in relation of the angle speed

$$\text{If } \ddot{\theta} < 0, \quad \tau_{\text{friction}_i} = a_i^- \cdot \dot{\theta}_i + b_i^-$$

We created graphs of the torque τ as a function of $\dot{q}$ and identified these linear friction components for each element of the system.

2.4 Revised Model with Friction Consideration

Upon reattempting to compute the equation of motion with the new friction model, the results showed improvements. The solution no longer diverged as rapidly, indicating that our friction model had a stabilizing effect. However, the function still did not accurately follow the theoretical trajectory of the robot's motion.

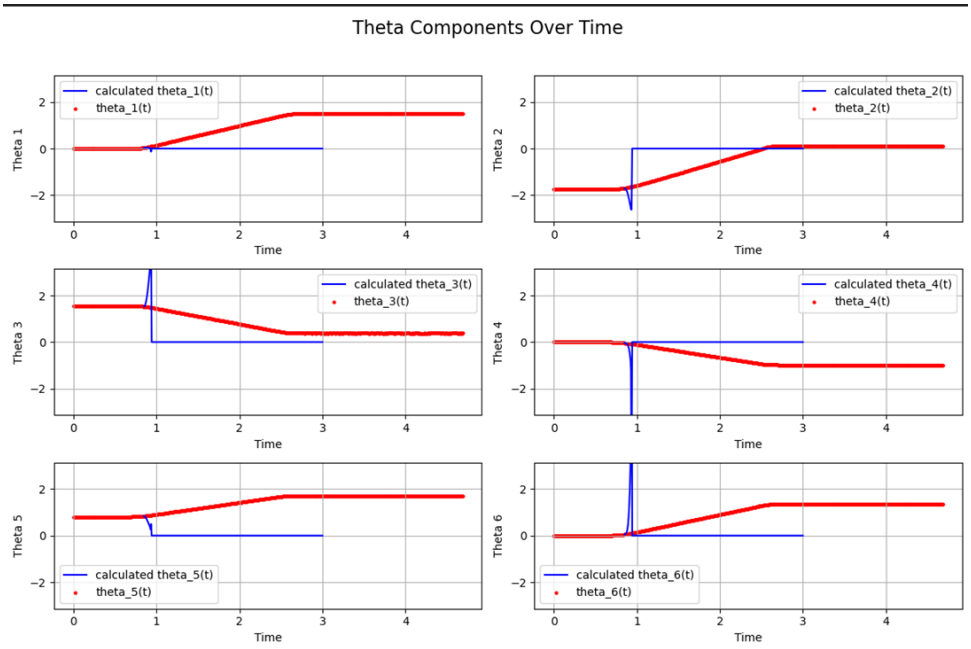

Figure 2.4: Computation of the theta components over time with the new friction model

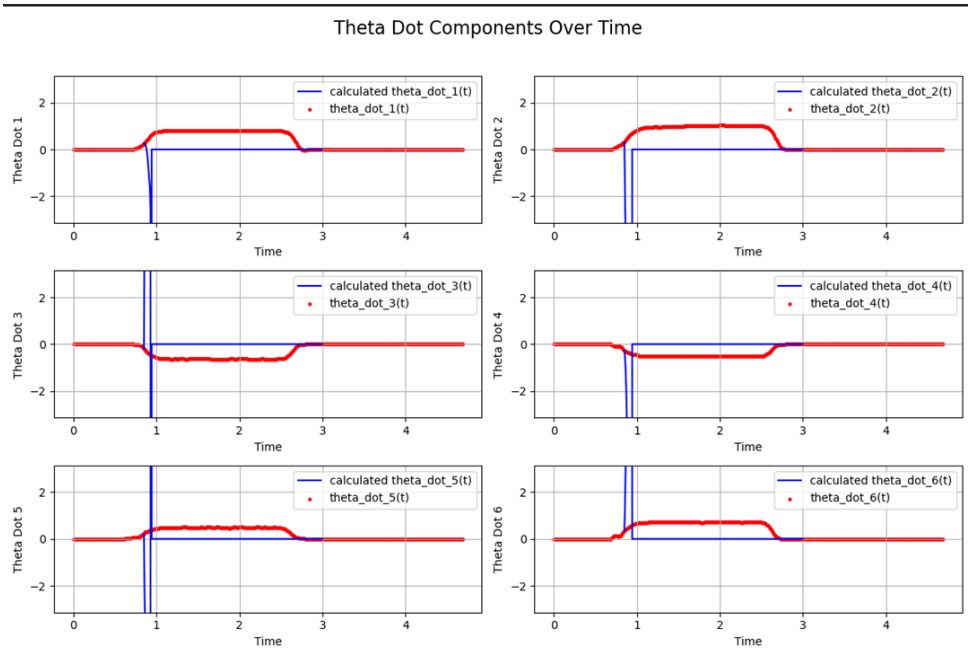

Figure 2.5: Computation of the theta dot components over time with the new friction model

2.5 Final Attempt: Swarm Optimization Approach

During the final weeks of the project, we employed a swarm optimization method to further refine the model. Using the `pyswarms` library, we applied particle swarm optimization to identify

the best parameters for the friction model that would yield a more accurate representation of the equation of motion.

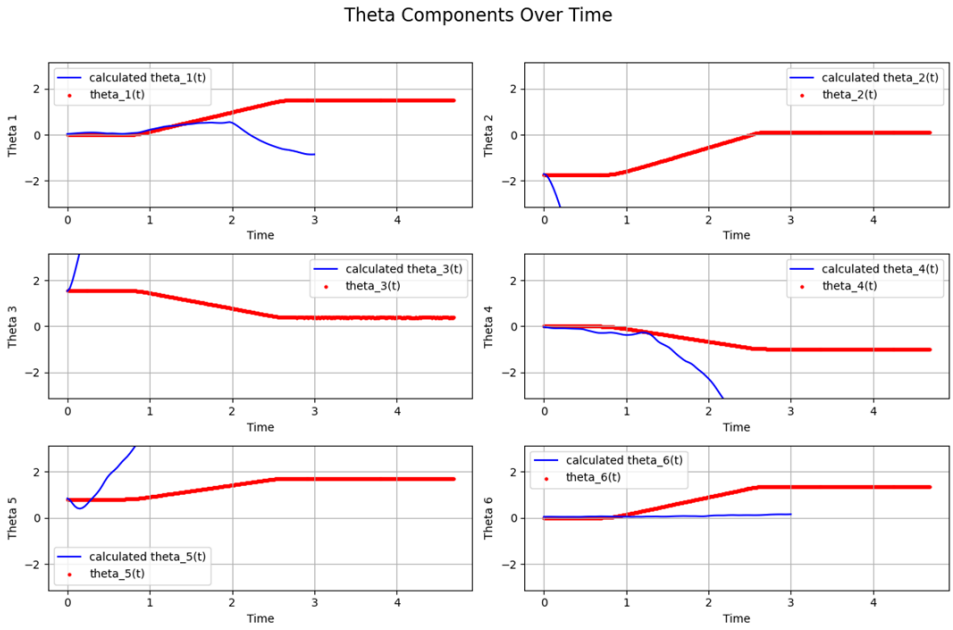

Figure 2.6: Computation of the theta components over time with the swarm optimization

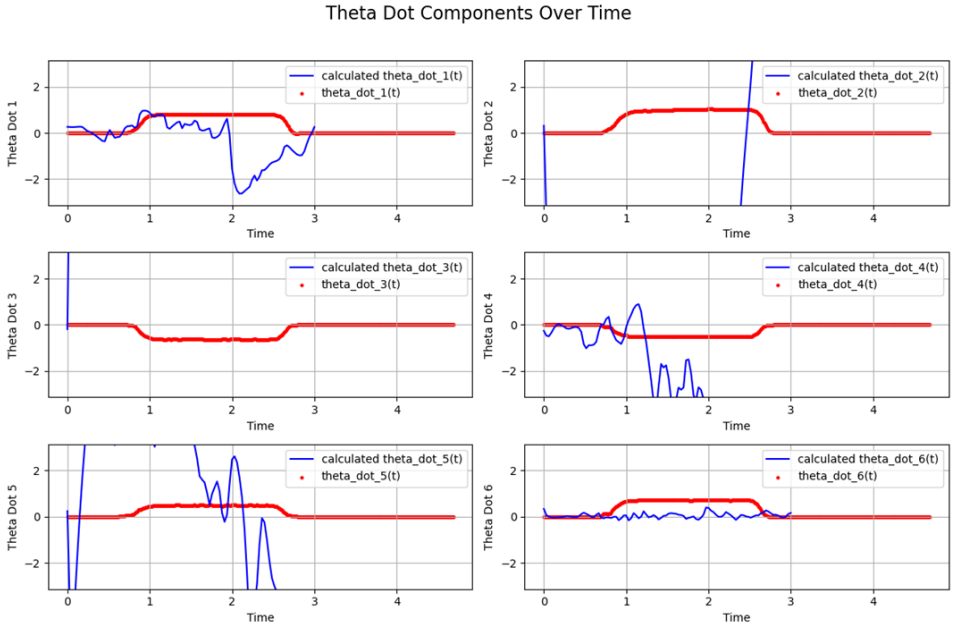

Figure 2.7: Computation of the theta dot components over time with the swarm optimization

The results showed some improvements, with the model roughly following the expected curve for the first and third component of the system, albeit for a limited duration.

2.6 Future Work and Recommendations

Despite these advancements, the optimized model did not entirely resolve the discrepancies, particularly over longer time periods. There are several avenues for further exploration to enhance the accuracy of the equation of motion, including:

- Applying the **simplex method** to fine-tune the friction model parameters.
- Exploring **disturbance observer techniques** to compensate for unmodeled dynamics and external disturbances.
- Investigating non-linear friction models that might better capture the real behavior of the system.

2.7 Conclusion of Integration Model Development

The development of an accurate integration model for the robot's forward dynamics proved to be a complex and iterative process. While significant progress was made, especially with the inclusion of friction and swarm optimization, further refinement is necessary to achieve a fully reliable model. The groundwork laid during this project a few basis for continued investigation into more sophisticated methods to model and control the robotic system's dynamics. But there are still a lot of work and experimentation that need to be done.

Conclusion

The first part of my project, focusing on the optimization of the equation of motion code, was ultimately a success. I was able to significantly reduce the computation time, making the code much more efficient and viable for real-time communication with the camera system. This improvement was crucial for enabling the robotic arm to interact dynamically with its environment, laying a solid foundation for future developments.

However, there is still considerable work to be done on the forward dynamics aspect of the project. Despite testing multiple methods, we have not yet achieved a fully successful model. This is to be expected, as finding the correct forward dynamics model often involves a process of trial and error, requiring careful adjustments and testing to identify the most accurate approach.

One of the main challenges during the optimization phase was analyzing the existing code to identify the points where it was getting stuck. This was a critical task because each code run could take several hours, and a wrong decision could lead to significant time losses. Therefore, making strategic choices on what information to print and analyze was key to understanding the bottlenecks and improving the performance.

For the forward dynamics phase, the primary difficulty lies in finding the right model and thoroughly analyzing why certain models fail to perform as expected. This part of the project remains a work in progress.

Ultimately, this internship has been a valuable learning experience. It not only deepened my technical skills in optimization and forward dynamics but also enhanced my problem-solving abilities, particularly in analyzing code and refining models for robotic systems. I know this experience will serve a lot of uses in my future career.

Acknowledgements

I would like to express my sincere gratitude to Professor Andrea Rauh, who not only gave me the opportunity to work at the University of Oldenburg but also provided valuable guidance and support throughout my internship. His insights and encouragement helped me navigate through the many challenges I faced during this project.

I would also like to thank Professor Frederike Bruns for her assistance with various coding-related questions and for her willingness to share her expertise whenever I encountered technical difficulties.

Finally, I am grateful to Léo Bernard, whose previous work laid the foundation for my internship project. His detailed documentation made it easier for me to understand the key elements of the project and build upon his research effectively.

Bibliography

[1] L. Bernard, *Establishing the equation of motion of a ViperX 300S robotic arm*, 2024.

[2] NumPy Developers, *NumPy: Fundamental package for scientific computing with Python*, https://numpy.org/

[3] SciPy Developers, *SciPy: Scientific computing tools for Python*, https://scipy.org/

Code Documentation for ViperX Equations of Motion (EOM)

Description

This code handles the generation of equations of motion (EOM) for the ViperX robotic arm. It consists of three main components:

- **geometry_inertia.py**: Used in conjunction with `eom_6dof.py` to create the EOM matrices in symbolic form before converting them into numerical functions.
- **eom_6dof.py**: Works with the inertia and geometry calculations to derive the dynamic equations for a 6-degree-of-freedom (6-DOF) robotic system, focusing on transforming these symbolic equations into numerical models.
- **inverse_integration.py**: Focuses on forward dynamics, using the previously generated EOM to simulate the robot's behavior over time. It applies numerical integration methods to predict the system's state based on applied forces or torques. It then plots a graph comparing the the computed equation to a theoretical model
- **inverse_integration_swarm.py**: Serves the same purpose but attempts a swarm particles approach to solve the problem.

Disclaimer

To utilize the numerical functions generated by this code, ensure that the you're starting the code in the directory:
`robotic_arm-main-ros2_arm_ws_17m/robotic_arm-main-ros2_arm_ws_17m/ros2_arm_ws_17m/src/my_package`
If you intend to work with different files or configurations, you will need to redefine the file paths accordingly.

Also note that the forward integrations pis still a work in progress and doesn't give a satisfying result yet.

.1 Descriptions

geometry_inertia.py

- **T_inv()**
  - *Purpose*: Computes the inverse of a transformation matrix.
  - *Utility*: Converts coordinate frames in kinematic chains.
- **T()**
  - *Purpose*: Constructs a transformation matrix from given parameters.
  - *Utility*: Describes the relative position and orientation of robotic links.
- **expression_sub()**
  - *Purpose*: Substitutes symbolic expressions with specified values or simplified forms.
  - *Utility*: Converts symbolic equations into a form suitable for numerical evaluations.

eom_6dof.py

- **timeout_handler()**
  - *Purpose*: Handles timeouts during long-running computations.
  - *Utility*: Ensures efficient use of computational resources.
- **safe_trigsimp()**
  - *Purpose*: Safely simplifies trigonometric expressions.
  - *Utility*: Enhances the stability of numerical calculations.
- **V()**
  - *Purpose*: Computes the potential energy of the robotic system.
  - *Utility*: Used in forming the Lagrangian for deriving equations of motion.
- **Jv()** and **J**ω()
  - *Purpose*: Calculate the Jacobian matrices for linear (`Jv`) and angular (`JÏ`) velocities.
  - *Utility*: Map joint velocities to end-effector velocities, essential for robotic control.

m2f_subst()

- *Purpose*: Transforms a symbolic matrix into a numerical function

- **Dynamic Functions (G_func(), D_func(), C_func(), CG_func())**
  - **G_func()**: Computes the gravitational force vector.
  - **D_func()**: Represents the inertia matrix.
  - **C_func()**: Represents the Coriolis force matrix.
  - **CG_func()**: Combines Coriolis and gravitational forces.

inverse_integration.py

- **tau_frot()**
  - *Purpose*: Calculates the frictional torque in the robot's joints based on the model presented in the report.
  - **dy()**
    - *Purpose*: Determines the derivative of the system's state vector for numerical integration.

inverse_integration_swarm.py

The following coefficients are used to define the torque equations for both positive and negative accelerations:

- `a0, b0, m0, p0, ... , a5, b5, m5, p5`

Purpose: These coefficients describe the linear relationship between the torque and the system state, aiding in more accurate simulation of dynamic behavior.

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Carl von Ossietzky Universität Oldenburg

Adresse / *Address* Ammerländer Heerstraße 114-118
26111 Oldenburg, GERMANY

Tél / *Phone (including country and area code)* +49 441 798-4195

Nom du superviseur / *Name of internship supervisor*
Prof. Dr.-Ing. habil. Andreas Rauh and Dr.-Ing. Friederike Bruns
Fonction / *Function* Professor / Postdoctoral Researcher

Adresse e-mail / *E-mail address* andreas.rauh@uni-oldenburg.de; friederike.bruns@uni-oldenburg.de

Nom du stagiaire accueilli / *Name of intern* Luc-André Terrine

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak).***

MISSION / *TASK*

❖ La mission de départ a-t-elle été remplie ? A ~~B~~ C D E F
Was the initial contract carried out to your satisfaction?

The overall intership can be evaluated as good, with slight tendency to grade C due delays and imprecision in report preparation

❖ Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☑ non/*no*
Was the intern lacking skills?

Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

❖ Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

A ~~B~~ C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* M. Terrine integrated well into the group and was overall reliable

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?

Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

A B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* M. Terrine showed good willingness to aquire new knowledge and to familiarize with his subject. However, he had to be remined several times to search actively for discussions with his supervisors.

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? A B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* He worked very autonomously, however, the efficiency of his work could have been much better if he would have searched more actively for help. Especially, given recommendations to improve the submitted presentation slides and the report were not considered. The report was submitted with a delay of 4 weeks and will be difficult to use for follow-up students due to a lack of precision.

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? A B C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* see above

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : A B C D E F
Please evaluate the technical skills of the intern:

Despite the shortcomings mentioned above, the overall grade is B as the student tried to continuously improve his technical skills. However, his code documentation could have been much better.

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☑ oui/*yes* ☐ non/*no*

Fait à Oldenburg, le 18/10/2024
In _____, on _____

Carl von Ossietzky Universität Oldenburg
Fakultät II
Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik
Abt. Verteilte Regelung in vernetzten Systemen
Prof. Dr. Andreas Rauh
D-26111 Oldenburg

Signature Entreprise _____ Signature stagiaire
Company stamp _____ *Intern's signature*

Andreas Rauh (Prof. Dr.-Ing. A. Rauh)

Merci pour votre coopération
We thank you very much for your cooperation

(Dr.-Ing. F. Bruns)