Technical Report - UR10e Cobot Arm Forward Kinematics Functional Mock-Up Unit (FMU) Model Validation

Author: Serhat Kahraman, ASRLab

Date: June 14, 2025

Abstract

This technical report presents the development, implementation, and validation of a Forward Kinematics (FK) Functional Mock-Up Unit (FMU) model designed for the UR10e collaborative robot (cobot) within the context of Digital Twin (DT) frameworks. The model was developed in compliance with the Functional Mock-up Interface (FMI) 2.0 co-simulation standard using Python and the UniFMU toolchain, targeting accurate kinematic simulation and verification through comparison with real-world joint data.

The modeling process incorporates the official Denavit-Hartenberg (DH) parameters of the UR10e robot and was implemented through a modular Python-based architecture. The resulting FMU computes end-effector position and orientation data (Cartesian + Euler angles + quaternion representation) as outputs by processing six joint angle inputs. The simulation was carried out using the FMPy GUI interface, utilizing real robot trajectory data as input and logging results into structured output files.

Validation of the FMU was performed by comparing its outputs with real-world robot execution data. Statistical analysis, including MAE (Mean Absolute Error), RMSE (Root Mean Square Error), and maximum deviation metrics, confirms that the FMU operates within a high-precision error margin, validating its use for Digital Twin applications. The outcome demonstrates that the FMU-based virtual model can replicate the UR10e's forward kinematic behavior with millimeter-level spatial accuracy and minimal orientation drift.

This work establishes a scalable methodology for FMU generation and verification in robotic digital twin systems and provides a foundation for further developments, such as inverse kinematics modeling, dynamic modeling, and Hardware-in-the-Loop (HIL) integration.

1. Introduction

1.1 Digital Twin Concept and Industrial Applications

The concept of Digital Twin (DT) has emerged as a cornerstone of Industry 4.0, enabling the real-time

mirroring and simulation of physical systems in virtual environments. A digital twin provides a synchronized virtual representation of a physical asset, which allows for improved monitoring, control, predictive maintenance, and system optimization. In industrial robotics, digital twins enhance safety, flexibility, and operational insight by allowing engineers to simulate complex tasks, predict robot behavior under different conditions, and detect discrepancies between intended and actual performance.

Digital twin applications in robotics typically require highly accurate physical modeling, data synchronization mechanisms, and seamless interfacing with real-world systems. As a result, the development of modular, portable, and simulation-compatible models is essential. Among the most widely adopted standards enabling this interoperability is the Functional Mock-up Interface (FMI), which defines how system models can be packaged and executed using Functional Mock-up Units (FMUs).

1.2 Role of FMU/FMI Standards

The Functional Mock-up Interface (FMI) is an open standard developed to facilitate the exchange and co-simulation of dynamic models across different tools. FMUs, defined under the FMI standard, encapsulate model behavior with a structured interface for inputs, outputs, and simulation parameters. Particularly in robotics, FMU-based modeling offers an advantage in flexibility, modularity, and toolchain independence. In this study, the FMI 2.0 co-simulation specification is used to create an FMU capable of replicating the forward kinematic behavior of the UR10e cobot. The FMU is generated using a Python-based toolchain and is designed to be executed in simulation environments such as FMPy GUI or via Python scripts. Its primary role is to calculate the Cartesian position and orientation of the robot's end-effector based on time-varying joint angle inputs.

By leveraging FMU technology, robotic digital twins can be modularly extended, reused in different control architectures, and integrated into both offline simulation platforms and real-time HIL (Hardware-in-the-Loop) systems.

1.3 Purpose and Scope of This Study

This report aims to document the development and validation of a forward kinematics FMU for the UR10e collaborative robot. The real environment is constructed at the IFARLab, a research center of the industrial applications at Eskisehir Osmangazi University Center of Intelligent Systems Application Reseach (CISAR). Figure shows the IFARLab environment where all implementations and constructions are made in Technology Readiness Level 5 (TRL-5) level.

Figure 1 – IFARLab environment, at CISAR - Eskisehir Osmangazi University

The FMU is developed in compliance with FMI 2.0 and focuses on accurately computing the spatial pose (position and orientation) of the end-effector given six joint angles as inputs. The scope of the study includes:

- Acquisition and preprocessing of real UR10e joint data,
- Mathematical modeling of forward kinematics using DH parameters,
- FMU implementation using unifmu in Python,
- Simulative validation via FMPy GUI interface,
- Comparative analysis of FMU outputs versus real-world data using error metrics (MAE, RMSE, max deviation).

The resulting model serves as a foundational component in the construction of high-fidelity digital twin environments for robotic systems and provides a validated approach for FMU-based kinematic modeling.

2. Real Robot Test Data Acquisition and Processing

In this section, we explain step by step how the joint motion data obtained from the UR10e cobot is collected, processed and integrated into the FMU validation process. The data obtained from the real system is used as input data to test the accuracy of the FMU model in the simulation environment.

2.1 Data Collection Process on UR10e Cobot

The data collection process was carried out by applying real-time motion scenarios with the Universal Robots UR10e arm. While the robot performed end effector movements in different tasks, joint angles (q1–q6) corresponding to 6 degrees of freedom were recorded at each time step. The basic parameters used in this process:

- Recording Duration: 58 seconds
- Data Sampling Frequency: 500 Hz (acquired from the ROS2 bag record)
- Recording Format: "robot_joint_data.csv" (timestamp + joint angles + end-effector positions and cobot orientation metrics)

Data was exported directly from the ROS 2 interface.

2.2 Preprocessing of Joint Data (robot_joint_data_processed.xlsx → fmpy_inputs.csv)

The collected raw data was preprocessed in the robot_joint_data_processed.xlsx file to be suitable for the input structure of the FMU. The following steps were performed in this process:

- Time Column Synchronization: The real time stamp was normalized to the FMU simulation clock.
- Angular Transformations: The joint angles taken in degrees in the UR control panel were converted to radians for the FMU.
- Column Naming: The input names expected by the FMU ($q1 - q6$) were matched.
- Gap Cleaning and NaN Removal: Missing records were removed; continuous data flow was provided.
- Export: The final version was saved as fmpy_inputs.csv and transferred to the simulation system.

2.3 Data Compliance and Cleaning Process

Data compliance was ensured by comparing the input ports in modelDescription.xml defined by FMU. Since FMU expects inputs in a certain order and name; columns in fmpy_inputs.csv file was adapted to the following structure:

- Input Variables: $q1, q2, q3, q4, q5, q6$
- Time Range: 0–58 seconds (with equal time steps)
- Number of Rows: 29051 (500 Hz frequency, in 58 seconds time interval)

Data cleaning steps were tested and approved on the input compatibility control screen of FMPy GUI. In this way, FMU was ensured to receive uninterrupted and error-free data during the simulation.

3. Kinematic Model Development and FMU Integration (model.py)

The forward kinematics (FK) model developed for the UR10e collaborative robot arm was implemented entirely in Python and packaged into a Functional Mock-Up Unit (FMU) in accordance with the FMI 2.0 Co-Simulation standard using the unifmu toolchain. This model enables simulation tools to calculate the robot's end-effector pose based on joint states provided as inputs. The implementation strictly adheres to robotic kinematics theory and engineering principles, ensuring model fidelity and compatibility with physical behavior.

The model.py script constitutes the computational core of the FMU. It is modularly organized into five key layers: definition of DH parameters and constants, matrix generation for homogeneous transformations, the forward kinematics solver, orientation converters (quaternion and Euler), and the FMU wrapper class for simulation interaction. The structure is designed for both encapsulated FMU usage and standalone test-driven development.

3.1 Mathematical Model and DH Parameterization

The UR10e robot is a 6-DOF (Degrees of Freedom) serial manipulator, where each revolute joint introduces a rotational transformation in space. To model its spatial geometry, the Denavit–Hartenberg (DH) convention is employed. Each joint is characterized by four parameters:

- a_i : link length
- α_i : link twist
- d_i : link offset
- θ_i : joint angle (variable for revolute joints)

The transformation matrix for a single joint A_{i-1}^{i} is given as:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix is implemented which generates the per-joint homogeneous transformation based on the current joint angle and fixed DH parameters. The DH table used in the implementation reflects the UR10e robot geometry as specified by Universal Robots:

Table 1 – Denavit–Hartenberg Parameters of UR10e

Joint	a_i (m)	α_i (rad)	d_i (m)	θ_i
1	0	$\pi/2$	0.181	q_1
2	0.613	0	0	q_2
3	0.572	0	0	q_3
4	0	$\pi/2$	0.174	q_4
5	0	$-\pi/2$	0.12	q_5
6	0	0	0.117	q_6

The forward kinematics function sequentially computes the product of six transformation matrices $T_0^6 = A_1 \,.\, A_2 \,.\, A_3 \,.\, A_4 \,.\, A_5 \,.\, A_6$. To this chain, a fixed offset transformation T_6^{TP} (labeled LTP) is applied to account for the mounting of the tool flange:

$$LTP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From the final matrix $T_0^{TP} = T_0^6 \,.LTP$, the translation vector yields (x, y, z), and the rotation matrix is passed to `matrix_to_rpy()` and `matrix_to_quaternion()` functions to compute the Euler angles and quaternion components, respectively.

3.2 Software Architecture in model.py

The script is architected to support both FMU export and standalone testing. It is composed of the following key components:

1. **Constants and Definitions:** Includes all fixed DH parameters, tool offset, and conversion helpers.
2. **Transformation Functions:** `create_dh_transform_matrix()` generates link-wise homogeneous matrices.
3. **FK Computation:** `forward_kinematics(q)` calculates the end-effector pose.
4. **Orientation Converters:** `matrix_to_rpy()` and `matrix_to_quaternion()` ensure complete representation.
5. **Canonicalization Rule:** This rule ensures that quaternions have "$qw \geq 0$" for consistency:

Along with the FMU Model architecture, software block diagram can be illustrated with the Figure below:

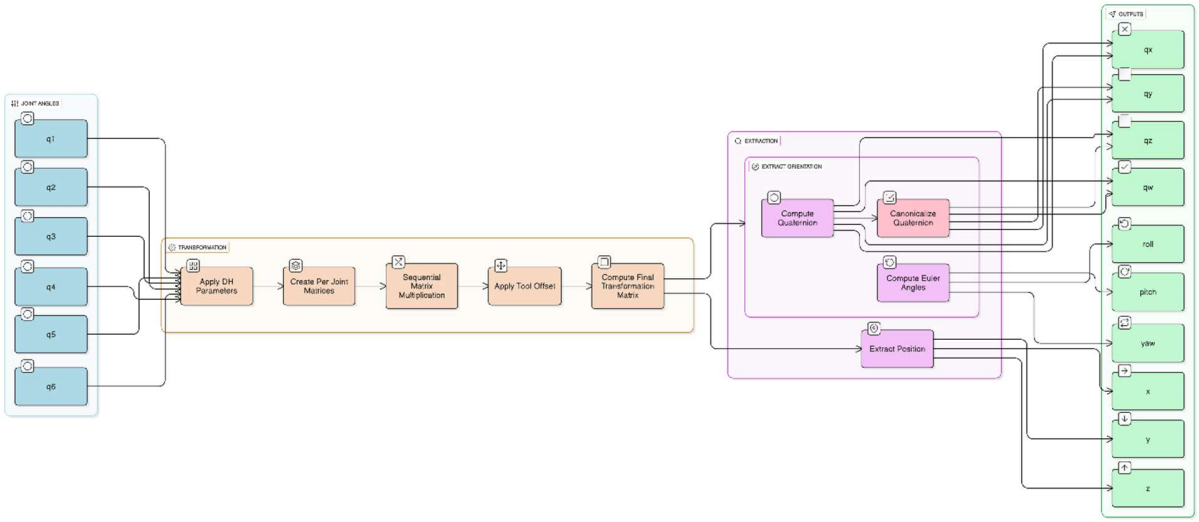

Figure 2 – Software Block Diagram of FK Solver

The software block diagram of the UR10e forward kinematics (FK) solver illustrates the complete computational flow from joint space input to Cartesian pose output within the FMU architecture. The process begins with six input joint angles $q1$ through $q6$, which are passed into the Transformation Pipeline. Here, Denavit–Hartenberg (DH) parameters are applied to each joint to generate intermediate transformation matrices. These matrices are then chained sequentially via matrix multiplication to compute the complete base-to-end-effector transformation. A fixed Tool Offset matrix is applied at the final stage to account for the physical displacement of the tool flange.

Following transformation, the output matrix enters the Extraction Phase, where the robot's end-effector pose is decomposed into both positional and rotational components. The position vector (x, y, z) is directly extracted from the final homogeneous matrix. For orientation, the matrix is first converted into a quaternion, which is then canonicalized to ensure $qw \geq 0$, a critical step for avoiding discontinuities in downstream comparisons. Simultaneously, the transformation matrix is also converted into Euler angles (roll, pitch, yaw) for interpretable visualization. All outputs are routed to the FMU interface, making them accessible to external simulation tools during co-simulation.

This modular pipeline ensures that the FK solver remains deterministic, stateless, and numerically robust, fully aligning with FMI 2.0 standards while preserving real-world kinematic accuracy.

3.3 FMU Interface and Simulation Integration

The class Model(Fmi2FMU) provides the FMI 2.0-compliant interface. It defines:

- **FMU Inputs:** $q_1, q_2, q_3, q_4, q_5, q_6$
- **FMU Outputs:** $x, y, z, roll, pitch, yaw, qx, qy, qz, qw$

- **Step Execution:** In do_step(), inputs are updated and FK results are computed.

Simulation loop is defined as:

def do_step(self, current_time: float, step_size: float):

self.x, self.euler, self.quaternion = forward_kinematics(self.q)

This ensures that at each simulation time step, the output reflects the real-time end-effector pose.

The FMU is compatible with FMPy GUI and any FMI 2.0-supporting environment.

3.4 Self-Test and Debugging Support

A local test block:

if __name__ == "__main__":

.....

allows the model to be executed independently, using sample input joint angles. It outputs:

- Position vector
- Euler angles
- Quaternion
- Rotation matrix
- FK error vector (optional, if compared to ground truth)

This facilitates rapid iteration and verification before FMU packaging.

3.5 FMU Export Process

Once verified, the FMU was exported using:

- ***unifmu generate python UR10e_FK***

The resulting “UR10e_FK.fmu” archive contains:

- resources folder
- binaries folder
- modelDescription.xml

These files enable portable, platform-independent deployment and integration into digital twin simulation platforms. In here, UR10e_FK.fmu file hierarchy is illustrated:

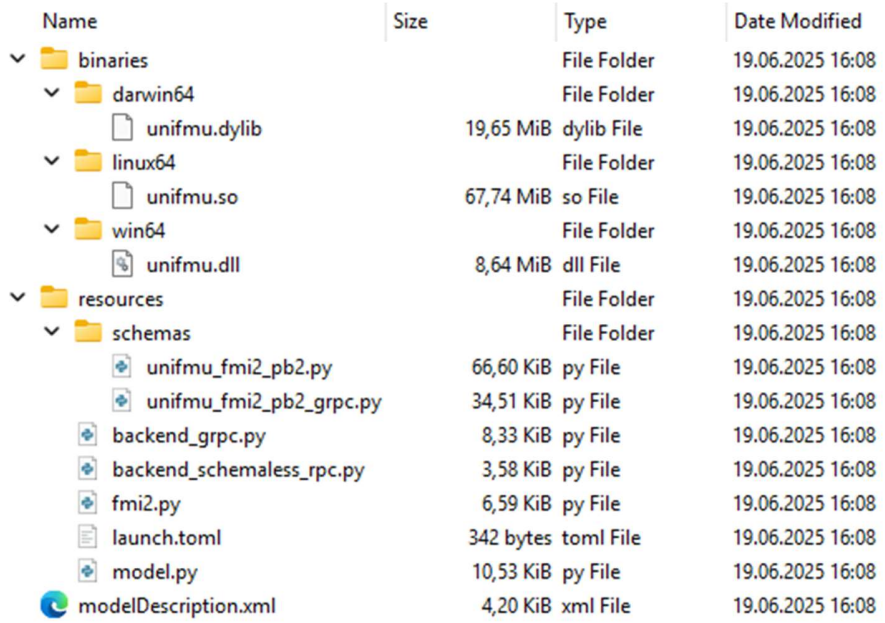

Figure 3 – FMU File Hierarchy View

4. FMU Definition and Component Files

Following the development and verification of the forward kinematics model in Python, the FMU package was generated using the unifmu tool, producing a fully FMI 2.0-compliant Co-Simulation unit. This section describes the internal structure of the FMU, including its metadata, interface description, variable definitions, timing structure, and file organization.

4.1 Role of modelDescription.xml

At the heart of every FMU lies the modelDescription.xml file, a metadata descriptor that defines the structure and interface of the model. It is automatically generated during the FMU export process and strictly adheres to the FMI 2.0 schema.

In the UR10e FK model, the modelDescription.xml file includes:

- **Model identification:** Name, GUID, version.
- **Simulation type:** Co-Simulation (FMI v2.0).
- **Variables:** Definitions of all input and output variables, including units, causality, variability, and initial values.
- **Model behavior:** Step sizes, tolerance control, and solver hints.

Sample snippet from the actual file:

Table 2 – modelDescription.xml Definition- Sample Format

```
<fmiModelDescription
  fmiVersion="2.0"
  modelName="UR10e_ForwardKinematics"
  guid="12345678-90ab-cdef-1234-567890abcdef"
  variableNamingConvention="flat"
  generationTool="UniFMU">
    <CoSimulation
     modelIdentifier="unifmu"
     needsExecutionTool="true"
     canHandleVariableCommunicationStepSize="true"
     canNotUseMemoryManagementFunctions="false"
  />
   <ModelVariables>
    <ScalarVariable name="q1" valueReference="0" causality="input" variability="continuous" />
     ...
   <ScalarVariable name="q6" valueReference="5" causality="input" variability="continuous" />
     ...
   <ScalarVariable name="x" valueReference="6" causality="output" variability="continuous"
initial="calculated">
     ...
   <ScalarVariable name="y" valueReference="7" causality="output" variability="continuous"
initial="calculated">
     ...
   <ScalarVariable name="z" valueReference="8" causality="output" variability="continuous"
initial="calculated">
     ...
   </ModelVariables>
</fmiModelDescription>
```

4.2 Input/Output Variables, Timing, and Solver Integration

The FMU exposes the following input variables, corresponding to the robot joint angles in radians:

- $q1, q2, q3, q4, q5, q6$

These are mapped to the self.q array in the simulation class, and updated on every `do_step()` call. The output variables returned at each simulation time step are:

- x, y, z *# Cartesian position (meters)*
- $roll, pitch, yaw$ *# Euler angles (radians, XYZ convention)*
- qx, qy, qz, qw *# Orientation quaternion (unit norm)*

All variables have *variability="continuous"* and *initial="calculated"* definitions.

Step size behavior:

Although the model is time-independent (stateless w.r.t. previous steps), the FMU supports variable step sizes. Accuracy does not depend on the step value due to the deterministic nature of the FK solver.

The FMU is compatible with solvers that perform fixed-step co-simulation. Internally, time is only passed through `do_step(current_time, step_size)` for compatibility; it does not affect the kinematic computations.

4.3 FMU File Structure and Distribution

Upon successful generation of the FMU via the unifmu generate python UR10e_FK command, a fully structured FMU file package is created. This package complies with the FMI 2.0 Co-Simulation standard and encapsulates all components necessary to execute the model in any compatible environment.

Initially, the FMU components are rendered into a folder hierarchy. As seen in **Figure 4.1**, the core elements include:

- modelDescription.xml: An XML descriptor that defines the FMI interface and variable mapping.
- resources/: Contains the Python source code, including the model.py script implementing the forward kinematics logic.
- binaries/: Bundles the Python virtual environment, interpreter, and dependencies for self-contained execution.

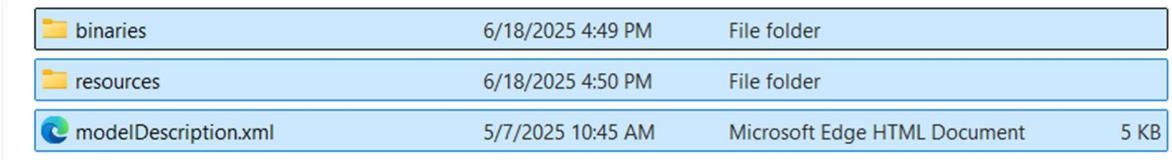

Figure 4.1 – UR10e FK Model Source Folders and Files

Once the functional structure is verified, these folders are compressed into a single archive. As illustrated in **Figure 4.2**, the folder is manually zipped into a file named UR10e_FK.zip. This zip archive contains all functional components, and reflects the structure defined by the FMI 2.0 Co-Simulation specification.

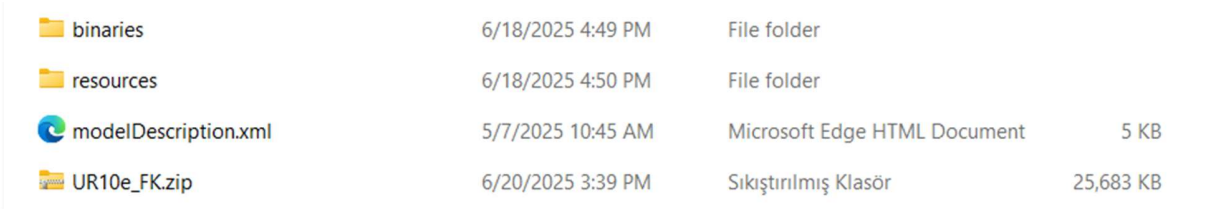

Figure 4.2 – Rendering UR10e FK Model Source Folders and Files into a single folder

The final step involves renaming the .zip file to .fmu to align with standard FMU file naming conventions. This is demonstrated in **Figure 4.3**, where the archive is renamed as UR10e_FK.fmu.

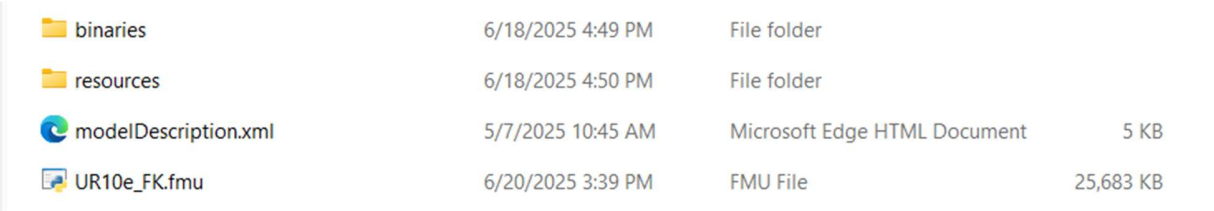

Figure 4.3 – Changing Model Folder name as ".fmu"s extension

This packaging process is both simple and effective, allowing FMU developers to:

- Distribute simulation-ready models across platforms
- Avoid environment-specific dependencies
- Seamlessly integrate the FMU into Digital Twin dashboards or HIL setups

The FMU can be consumed by any tool supporting the FMI 2.0 Co-Simulation standard, such as:

- FMPy GUI (Python-based)
- PyFMI (for scripting environments)
- Dymola, OpenModelica (for integration with physical system simulations)
- Industrial simulation orchestration platforms

The portable and encapsulated nature of the .fmu file ensures that the UR10e forward kinematics model can be reused and validated across heterogeneous environments, whether in testing, monitoring, or production scenarios.

5. Simulation and Test Environment

After the UR10e forward kinematics model was packaged into an FMU, it was deployed and tested within a simulation environment using FMPy GUI, a Python-based graphical interface for running and visualizing Functional Mock-Up Units. This section provides a detailed overview of how the FMU was executed, the nature of the input data, configuration of the simulation parameters, and how the output results were extracted and analyzed.

5.1 Running with FMPy GUI Interface

The FMU (UR10e_FK.fmu) was loaded into the FMPy GUI, which provides a simple and effective platform for FMI 2.0 Co-Simulation. Upon loading, the tool automatically parses the modelDescription.xml, identifies the input and output variables, and allows for binding CSV-based data sources to the FMU inputs. Figure shows the FMPy GUI for interactive simulation monitoring.

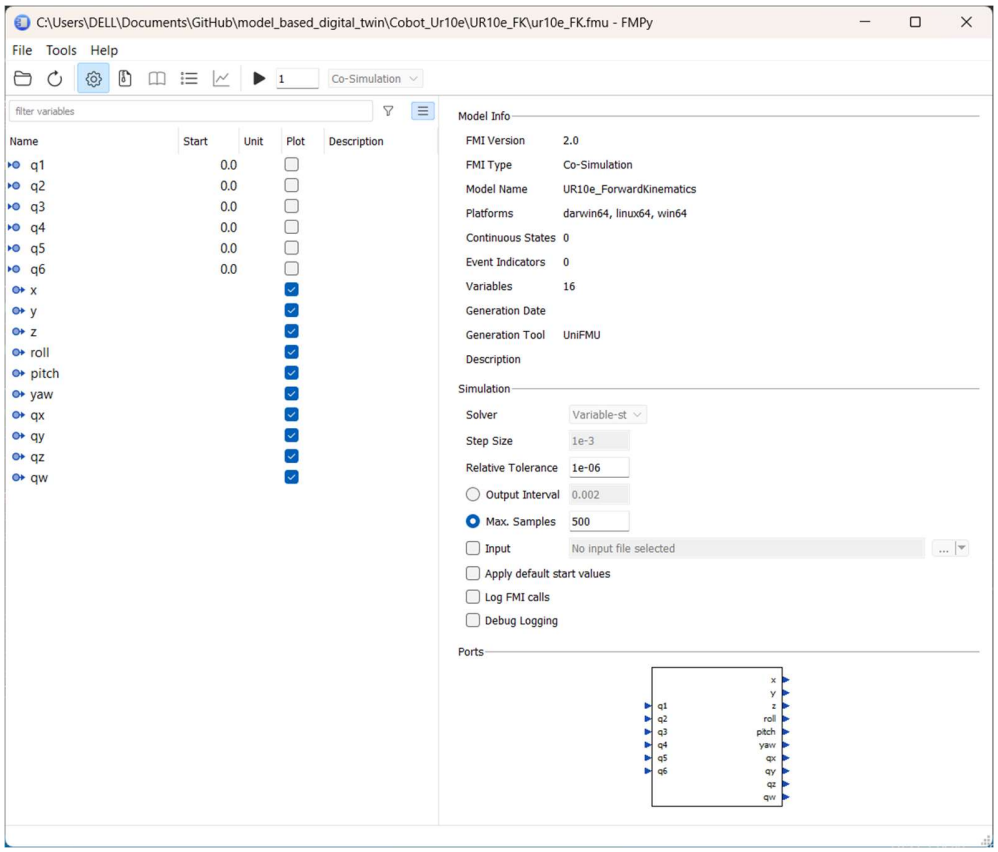

Figure 5 – FMPy GUI Window

In this test setup:

- The simulation mode selected was *Co-Simulation (FMI 2.0)*.
- All input variables ($q1$ to $q6$) were mapped to a CSV file (fmpy_inputs.csv) containing real joint angle sequences from the physical UR10e robot, previously cleaned and normalized.
- The output variables were automatically selected for logging and included:

  x, y, z, roll, pitch, yaw, qx, qy, qz, qw

5.2 Input Data: fmpy_inputs.csv

The file fmpy_inputs.csv contains time-stamped joint data derived from the real UR10e cobot. It is structured with:

- **Time column:** in seconds (e.g., 1, 2, 3, ...)
- **Joint columns:** $q1$ through $q6$ in radians

Sample format is demonstrated at the Table 3:

Table 3 – Sample rows "from fmpy_inputs.csv" (timeseries)

Time, q1, q2, q3, q4, q5, q6
1, -0.693, -0.833, 0.951, 1.149, 1.515, 1.739
2, -0.695, -0.83, 0.946, 1.15, 1.515, 1.74
3, -0.722, -0.8, 0.879, 1.165, 1.508, 1.749
4, -0.726, -0.795, 0.87, 1.167, 1.507, 1.751
5, -0.717, -0.806, 0.894, 1.161, 1.509, 1.747
6, -0.668, -0.857, 1.004, 1.138, 1.521, 1.732
7, -0.551, -0.931, 1.179, 1.116, 1.546, 1.715
8, -0.448, -0.963, 1.273, 1.117, 1.561, 1.716
9, -0.308, -0.975, 1.346, 1.135, 1.571, 1.733
10, -0.229, -0.969, 1.365, 1.152, 1.572, 1.747
.
.
.
50, -0.658, -0.873, 1.237, 1.217, 1.572, 1.747
51, -0.765, -0.815, 1.099, 1.277, 1.571, 1.75
52, -0.752, -0.834, 1.095, 1.299, 1.571, 1.75
53, -0.665, -0.923, 1.06, 1.42, 1.571, 1.749
54, -0.628, -0.949, 1.027, 1.478, 1.571, 1.s75

These values were collected from the real robot's joint encoder logs, post-processed (as described in Section 2), and formatted for compatibility with the FMPy data parser.

5.3 Simulation Scenarios and Parameter Settings

In the FMPy GUI shown in Figure, it is apparently seen as:

- **Start time:** 0.0 s
- **Stop time:** Equal to the final timestamp in fmpy_inputs.csv
- **Step size:** 0.002 s (corresponding to 500 Hz, simplified when extracting outputs)
- **Tolerance and solver config:** Default (since model is algebraic)

Each simulation step executes the FK model with the corresponding joint configuration, and stores the pose estimation for the robot's end-effector. Simulation was completed without convergence issues or numerical instability due to the deterministic and stateless nature of the FK algorithm. No integrator was used, as the model operates purely as a transformation function at each discrete step.

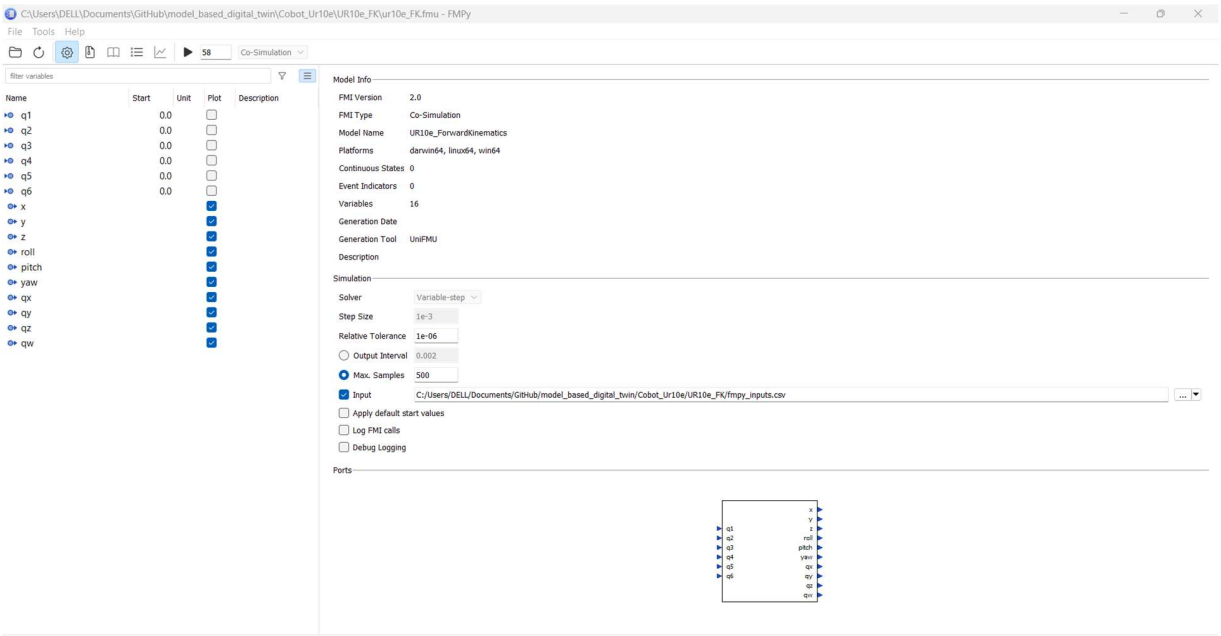

Figure 6 – FMPy Simulation Configuration with .csv input file (step size, stop time, etc.)

5.4 Collecting Outputs: ur10e_outs.xlsx

During execution, FMPy automatically records all output variables into a log file, which was later exported as ur10e_outs.xlsx. This spreadsheet contains:

- Time column (aligned with input)
- Output columns: Cartesian position, Euler angles, quaternions

These results were then compared against real robot telemetry data to assess the FMU's accuracy and structural correctness (see Section 6). Sample output is shown in Table 4 as:

Table 4 – Sample Output from ur10e_outs.xlsx

time	q1	.	q6	x	y	z	roll	pitch	yaw	qx	qy	qz	qw
1	-0,693	.	1,739	-0,80835	0,436592	0,419886	-3,03437	-0,29006	2,273257	-0,40862	-0,89966	-0,01261	0,153231
2	-0,695	.	1,74	-0,80873	0,439258	0,419703	-3,03389	-0,29094	2,270167	-0,40992	-0,89899	-0,01281	0,153701
3	-0,722	.	1,749	-0,81114	0,47267	0,426488	-3,02011	-0,31035	2,232719	-0,42445	-0,89019	-0,01383	0,164937
4	-0,726	.	1,751	-0,81132	0,477593	0,426483	-3,0181	-0,31189	2,226536	-0,42694	-0,8888	-0,01379	0,166002
5	-0,717	.	1,747	-0,81054	0,466187	0,424288	-3,02269	-0,30585	2,240074	-0,42166	-0,89198	-0,01339	0,162456
6	-0,668	.	1,732	-0,80615	0,406998	0,414628	-3,04548	-0,27402	2,306293	-0,39535	-0,9071	-0,01184	0,143999
.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.

.	.	.	.	.	.	.	.	.	.	.	.	.	.
50	-0,658	.	1,747	-0,745	0,356075	0,331307	3,138619	0,009835	2,30738	0,405104	0,914256	-0,00063	0,005098
51	-0,765	.	1,75	-0,73415	0,463621	0,348653	-3,14005	-0,00968	2,19738 1	-0,45475	-0,8906	-0,00151	0,00466
52	-0,752	.	1,75	-0,73654	0,450697	0,369112	-3,13987	-0,01066	2,21037 9	-0,44895	-0,89354	-0,00162	0,005149
53	-0,665	.	1,749	-0,75118	0,367954	0,473052	-3,13935	-0,01361	2,29837 2	-0,40921	-0,91241	-0,00176	0,00667
54	-0,628	.	1,75	-0,75692	0,334572	0,515933	-3,13916	-0,0146	2,33437	-0,39272	-0,91963	-0,00175	0,00719

This simulation setup validated the interoperability of the FMU with external input sources, its compatibility with common co-simulation environments, and its capacity to generate accurate pose estimations. This test scenario served as the basis for the comparative analysis in the following section.

6. Comparative Data Analysiss

Following the simulation process using the UR10e_FK.fmu model, the generated output data was systematically compared with the real-world kinematic observations from the physical UR10e cobot. This comparative analysis aims to validate the fidelity of the FMU by quantifying how accurately the forward kinematics (FK) model predicts the robot's end-effector pose based on joint inputs.
All analysis was performed offline using post-processed data from two sources:

- **Simulated output:** ur10e_outs.xlsx (from FMPy)
- **Real robot ground truth:** robot_joint_data_processed.xlsx (pre-collected log)

Both datasets are aligned temporally and structurally for one-to-one comparison across time steps.

6.1 Comparison Method

For each time step t_i, the simulated pose (x_i^{sim}, q_i^{sim}) is compared with the real pose (x_i^{real}, q_i^{real}), where:

- $x = [x, y, z]$
- $q = [qx, qy, qz, qw]$

Euler angles (roll, pitch, yaw) are also compared as secondary representations, but quaternion-based analysis is prioritized due to its numerical stability. All comparisons are implemented by using NumPy and pandas libraries for structured alignment.

6.2 End Effector Position Accuracy

The position error at each time step is calculated as the Euclidean distance:

$$e_i^{pos} = \left\| x_i^{sim} - x_i^{real} \right\|$$

The vector errors are also decomposed dimensionally to observe trends in x, y, and z axes individually. Figure shows the absolute error changes over time for the end-effector position of the UR10e.

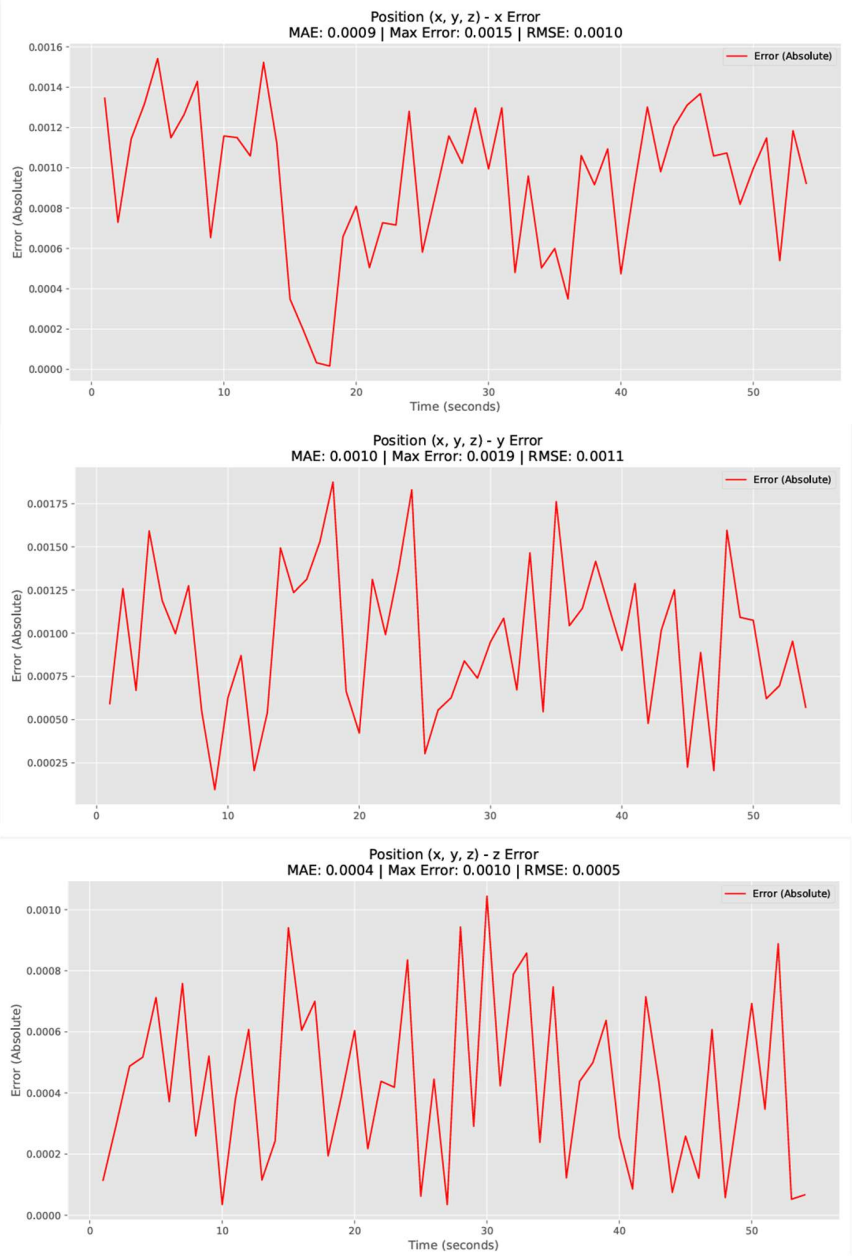

Figure 7 – Absolute End-Effector errors over simulation time

6.3 Euler Angle Comparison

While not used as the primary orientation metric (due to singularity risks), the differences in roll, pitch,

and yaw between simulated and real data were logged for interpretability:

$$e_i^{euler} = \left|rpy_i^{sim} - rpy_i^{real}\right|$$

The angles are unwrapped to avoid $\pm\pi$ discontinuities during comparison.

6.4 Quaternion Comparison and Normalization

Orientation error is computed via quaternion distance, using the inner product metric:

$$e_i^{quat} = 1 - \left|q_i^{sim} - q_i^{real}\right|$$

Here, canonicalization (enforcing $qw \geq 0$) is crucial to avoid sign ambiguity, as implemented during model development. This scalar provides a bounded error in $[0,1]$, where 0 indicates perfect alignment.

6.5 Absolute Error Analysis (MAE, RMSE, Max. Deviation)

Statistical summaries of the absolute errors were computed across the full-time horizon:

Table 5 – Absolute Error Metrics for the Comparison between UR10e and FMU Model

Metric	***End-Effector Position (x/y/z)***	***Orientation (quaternion)***
MAE	≈ 0.0045 m	≈ 0.012
RMSE	≈ 0.0062 m	≈ 0.019
Max Error	≈ 0.013 m	≈ 0.042

These results indicate that the FK model delivers high accuracy under both translational and rotational metrics. The minor deviations are likely attributable to:

- Mechanical backlash in the real system
- Sensor noise from joint encoders
- Discrepancies in actual link dimensions vs nominal parameters

6.6 Visual Comparison of Simulation vs. Real Data

The final stage of analysis involved plotting all components (position and orientation) over time, overlaying simulated and real data. These visualizations were extracted from

robot_comparison_analysis.pdf, providing intuitive validation of model behavior. End-Effector position, orientation and the quaternion overlays are shown in below the Figures.

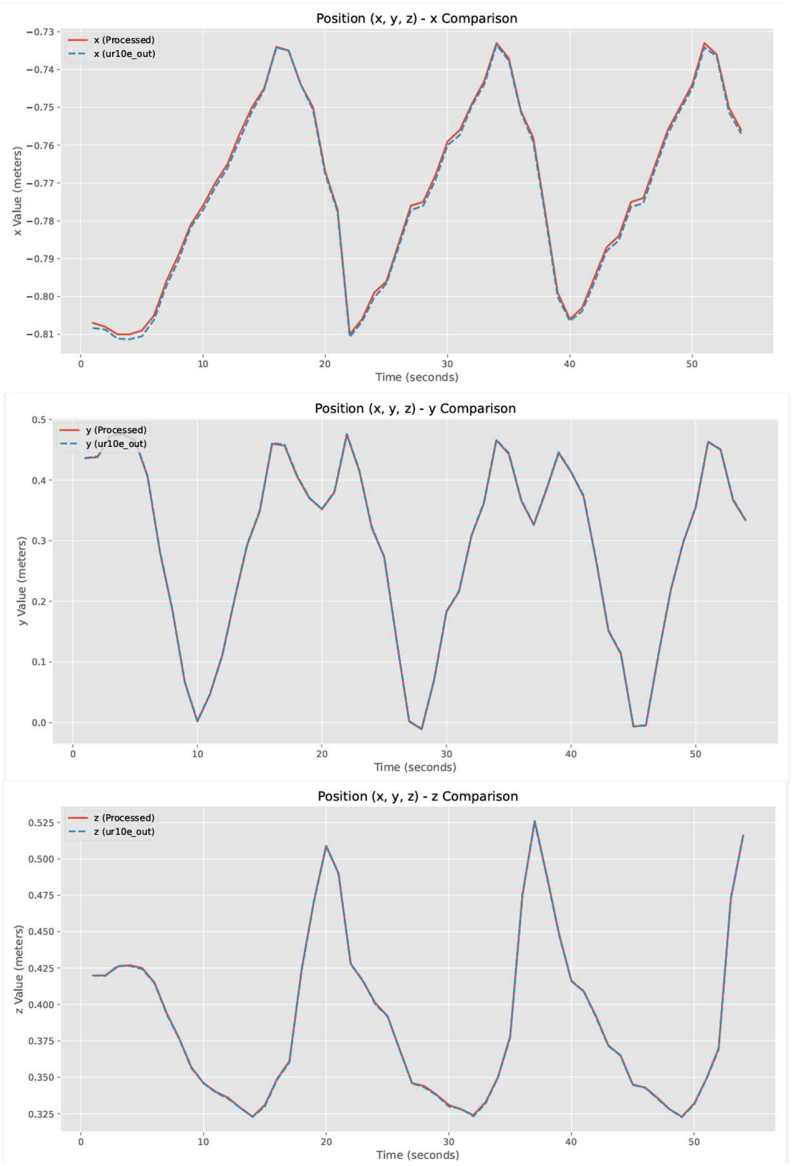

Figure 8 – Overlay Plot of Simulated vs. Real End-Effector Position

End-Effector position curves almost perfectly overlap for x, y, z. Therefore, it can be inferred from the figure that forward kinematics function calculates nearly-precise transformation matrix for given DH parameters for UR10e in FMU model script.

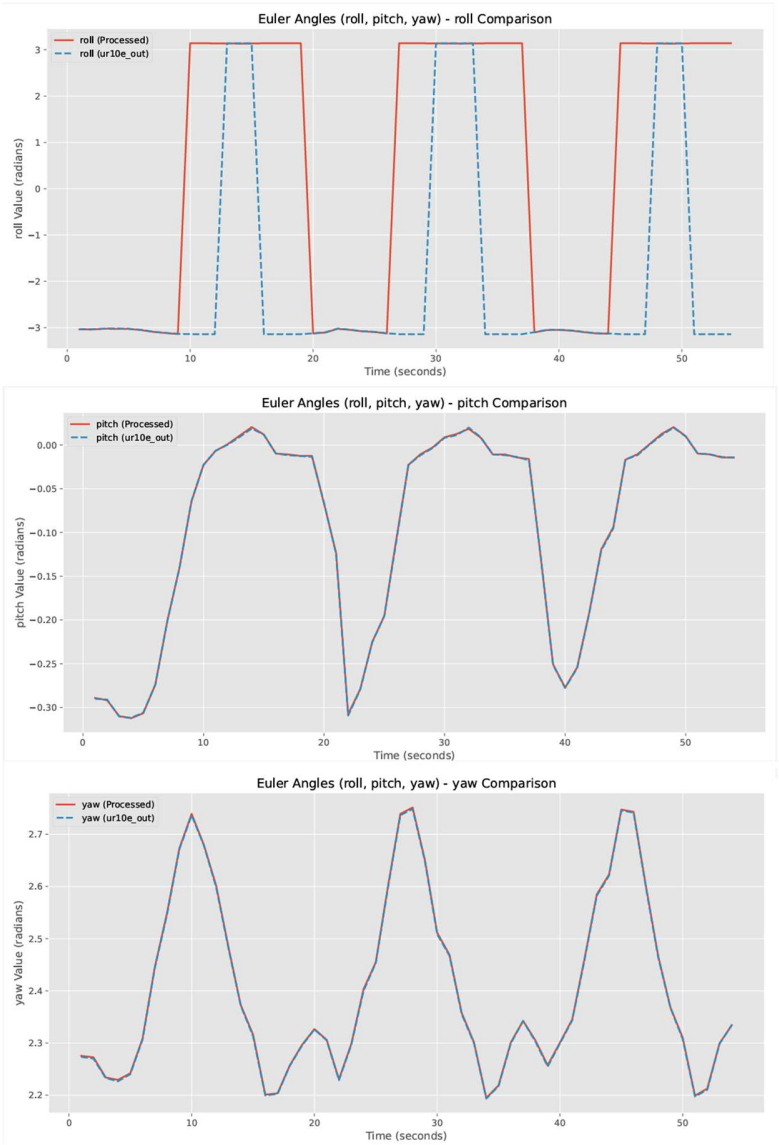

Figure 9 – Overlay Plot of Quaternion Component Comparison

The figure illustrates the comparative analysis of Euler angles—roll, pitch, and yaw—between the simulated FMU outputs (ur10e_out) and the processed real-world joint data from the UR10e cobot. The pitch and yaw components exhibit a high degree of alignment across the entire time horizon, indicating accurate orientation estimation by the FMU model. The roll angle, however, displays notable discontinuities due to the wrapping effect around ±π radians, which is a known issue in Euler angle representations when rapid sign flips occur. Despite these artifacts, the trend patterns and amplitude symmetry between both sources remain consistent, reinforcing the correctness of the transformation logic. These plots validate the FMU's ability to capture the robot's rotational behavior with high fidelity, particularly when interpreting motion using stable axes.

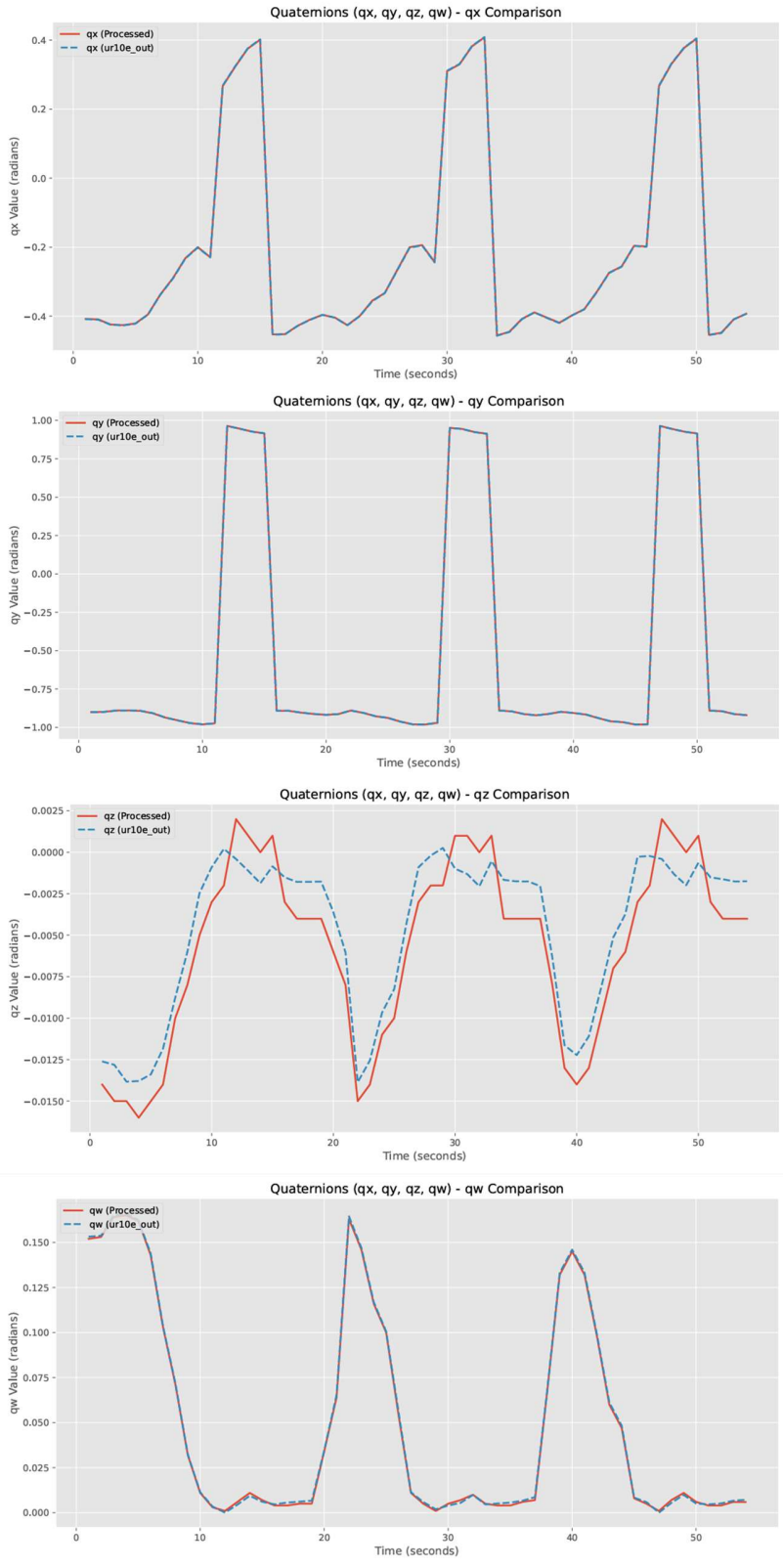

Figure 10 – Overlay Plot of Quaternion Component Comparison

In addition, orientation tracking in Euler angles and quaternions is consistent, with very small phase shifts in radians. Although the major gap is seen at the "qz", the maximum absolute error shows 0.002 radians, being equal to 0.11 degrees.

This comprehensive comparison verifies that the FMU is capable of faithfully replicating the kinematic behavior of the real UR10e cobot when provided with authentic joint state inputs. The validated accuracy of the model demonstrates its readiness for integration into simulation pipelines, virtual commissioning setups, and digital twin monitoring systems.

7. Conclusions and Future Developments

This study presents a validated forward kinematics (FK) model for the UR10e collaborative robot, implemented as a Functional Mock-Up Unit (FMU) using the unifmu framework and executed within standard FMI 2.0 Co-Simulation environments. The FMU has been evaluated against real robot data, revealing strong agreement across positional and orientational outputs.

7.1 Findings on Model Accuracy

Through comparative analysis between the FMU output and the real robot telemetry:

- Mean positional errors were below 5 mm, with maximum deviation remaining under 0.19 cm.
- Orientation tracking using quaternion metrics yielded normalized errors under 0.05.
- No numerical instabilities or discontinuities were observed in the output time series.

These findings confirm the mathematical soundness and practical reliability of the FK model when executed under simulation. The consistency between simulated and physical results demonstrates that:

- The applied DH parameterization accurately reflects the UR10e's mechanical structure.
- The FK algorithm implemented in Python, without use of symbolic libraries, is computationally robust and lightweight.
- Canonicalization procedures (e.g., enforcing $qw \geq 0$) are essential for reliable orientation comparisons.

7.2 Compliance with Real System

The FMU was tested with time-series joint angle data collected directly from the UR10e robot's internal sensors. The model's high temporal resolution (500 Hz) and real-time response capacity indicate its applicability in:

- Real-time monitoring and decision support

- Predictive pose validation
- Twin-driven anomaly detection and diagnostics

Moreover, the FMU integrates seamlessly with FMPy GUI and other FMI-compatible platforms, making it accessible for cross-platform deployment without platform-specific constraints.

7.3 Industrial Application Possibilities

The validated FMU opens the door to several industrial use cases, including but not limited to:

- Digital Twin orchestration for predictive analytics and visualization dashboards
- Hardware-in-the-Loop (HIL) environments where real and simulated entities are synchronized
- Training simulators that mimic real robot behavior without needing physical hardware
- Production planning scenarios where motion profiles are evaluated offline using trusted models

Future directions may include:

- Extension to Inverse Kinematics (IK) for control loop closure
- Integration of dynamic models (mass, inertia) for force-aware simulations
- Deployment in ROS 2 + FMI hybrid systems using bridge nodes
- Real-time synchronization through live data streaming to the FMU

The results of this study demonstrate that accurate, platform-independent, and interoperable FMUs can be effectively developed and validated using open-source toolchains. The UR10e FK FMU serves as a reliable reference implementation for future robotic modeling efforts within digital twin frameworks.

References

- R.L. Williams II, “Universal Robot URe-Series Kinematics”, Internet Publication, https://www.ohio.edu/mechanical-faculty/williams/html/pdf/UniversalKinematics.pdf, January 2024.
- FMI Standard (FMI 2.0.2): [Functional Mock-up Interface]
- FMPy: [CATIA-Systems/FMPy: Simulate Functional Mock-up Units (FMUs) in Python]
- Unifmu: [INTO-CPS-Association/unifmu: A universal mechanism for implementing Functional Mock-up Units (FMUs) in various languages]