RESEARCH ON A HYBRID TIME-EVENT-DRIVEN CO-SIMULATION FRAMEWORK BASED ON THE FMI STANDARD

Jielin Tang[1], Nan Xie[1,2], Beier Lu[2], Jing Li[3], Weifeng Xu[3], Wenhao Tang[2]*
[1]Shanghai Research Institute for Intelligent Autonomous Systems, Tongji University, Shanghai, 201210, China
[2]Sino-German Institute of Applied Technology, Tongji University, Shanghai, 201804, China
[3]Simulation and Digital Twin group, Siemens Technology China, Shanghai, 200082, China
Email: xienan115@tongji.edu.cn, 2311786@tongji.edu.cn

Abstract – With the rapid development of artificial intelligence, machine learning, and robotics, the application of robots in various fields has been increasingly widespread. However, traditional robotic systems face challenges in cross-environment operation and information synchronisation during multi-platform co-simulation, limiting their use in complex environments. This paper proposes a co-simulation framework based on the Functional Mock-up Interface (FMI) standard, fully leveraging the cross-platform versatility, operational simplicity, and bidirectional information transmission capabilities of Functional Mock-up Units (FMUs). By enabling bidirectional information synchronisation between different simulation modules and coordinating the time-stepping and event-driven state-switching mechanisms, the framework achieves efficient model reuse. The framework integrates a time-event hybrid-driven simulation mode, addressing the challenge of co-simulation between robots and their environments across different platforms, thereby improving development efficiency and synchronisation stability. Furthermore, by incorporating a deep reinforcement learning model in dynamic environments, this framework enhances the robot's capability to grasp dynamic objects. Experimental results demonstrate that the framework not only facilitates efficient multi-platform co-simulation but also successfully handles complex grasping tasks, showcasing excellent performance and stability.

Keywords: FMI Standards, Co-Simulation, Robot, hybrid Time-Event-Driven, Deep Reinforcement Learning.

1. Introduction

In recent years, with the swift advancements in artificial intelligence, machine learning, and robotics, the deployment of robots has become increasingly prevalent across numerous sectors [1]. From automated production lines in the manufacturing industry to assistive robots in healthcare and service sectors, intelligent robotics is taking on an ever more critical role. However, traditional robotic systems often remain confined to specific operational environments, lacking the flexibility to collaborate across different contexts, thereby limiting their range of application and adaptability [2,3].

Co-simulation is a methodological approach that enables the global simulation of coupled systems by integrating multiple simulators [4,20]. Each simulator, broadly defined as a "black box," is capable of displaying behaviour, consuming inputs, and generating outputs. Examples of simulators range from dynamic systems integrated with numerical solvers and software platforms to specialised real-time hardware, physical test benches, or even human operators [5].

In a co-simulation framework, subsystems exchange information through fixed or adaptive time-stepping, governed by specific time coordination schemes. Depending on the coupling structure or the nature of the models, co-simulation can be classified into several categories [6].

The current classification of co-simulation approaches follows different methodologies. One method focuses on the structural compatibility of subsystem models, assessing operational simplicity and practical feasibility as the primary criteria. This classification can be based on hierarchy, solver types, and interface structures. The Functional Mock-up Interface (FMI), renowned for its high versatility, has become a widely adopted co-simulation interface, applied extensively in software engineering and development projects [7,21]. An alternative classification is model-based, using discrete events and continuous time as the primary differentiation criteria [6,19].

Grasping tasks represent one of the core challenges in robotics [8].

For successful grasping, a robot must possess precise capabilities in object recognition, localisation, and manipulation [9]. Traditional rule-based methods, however, often struggle when faced with complex and dynamic environments. Consequently, the integration of deep learning techniques and computer vision to enhance robotic grasping capabilities in such environments has become a central focus in current research.

In modern industrial contexts, different software platforms often excel at handling specific forms of data [1,3,10]. In the domain of robotic control, it is crucial to integrate these data streams within real-world scenarios, enabling data synchronisation across disparate platforms to facilitate co-simulation. Nevertheless, one of the key challenges lies in the limited reusability and portability of models across different simulation platforms, owing to the highly customised nature of these models, which complicates their transfer between different environments [11]. This issue significantly undermines the efficiency of applying models across various simulation software. Moreover, the lack of an efficient time-synchronisation mechanism between different simulation systems often results in redundancy and operational inefficiency [12,13,18].

The instability of time synchronisation can lead to data processing delays, which in turn causes inconsistencies in simulation outcomes. Additionally, the integration of multiple complex systems and tools requires highly specialised technical expertise, increasing both development costs and system complexity. Furthermore, the inherent uncertainty in the motion states of target objects, influenced by environmental conditions and sensor accuracy, adds an additional layer of difficulty to robotic grasping operations, compromising both simulation accuracy and real-world performance [14].

This study addresses the challenge of effective multi-platform co-simulation by developing a framework based on the FMI standard. The proposed framework implements a time-event hybrid-driven co-simulation model that orchestrates timing and information exchange across diverse simulation modules, thereby enhancing interoperability and coordination efficiency. By integrating technologies such as FMI, TensorFlow, and OpenCV, the framework is designed to significantly improve multi-platform simulation efficiency, addressing key issues of synchronization stability and data consistency. This capability is expected to enhance the operational robustness of robotic systems in complex and uncertain environments, supporting precise and efficient execution of intelligent grasping tasks. Ultimately, this study aims to establish a versatile co-simulation framework that facilitates seamless co-simulation across various simulation platforms.

2. A Framework for Hybrid Time-Event-Driven Co-Simulation Systems Based on the FMI Standard

2.1 Co-Simulation System Framework

FMI is nowadays widely mentioned and applied as an open standard for importing and exporting simulation models. It primarily facilitates model exchange and co-simulation across different simulation platforms and tools. As a component-based simulation standard, FMI essentially encapsulates the required simulation models into a standardized Functional Mock-up Unit (FMU) and defines an interface for importing and exporting FMUs across various simulation platforms and tools [15,16]. The FMU can be regarded as a “black box” that interacts with external systems only through the interface and contains information such as the mathematical description of the model, parameters, variables, inputs and outputs, events, and solvers. Thanks to the versatility and many advantages of FMI in different simulation platforms, this paper proposes a co-simulation framework based on the FMI standard, as shown in Figure 1.

This framework utilizes a common server as the core relay for information transfer, where FMUs imported into different platforms synchronize and exchange information. The system comprises four main modules, with three of them, excluding the physical model, relying on client-side software or platforms. The dynamics simulation module provides the basic motion parameters for the virtual environment and continuous-time motion simulation based on the initial state. The motion simulation calculation module mainly receives the external motion data and renders the simulated movements of the robot. The robot's current state and the actual state of objects serve as event markers to control the system’s start and stop functions. External video data from the vision processing module identifies real-time parameters of the physical model, which assists in correcting the current simulation data, transmitting this data to the calculation module, and sending feedback to the dynamics simulation module. The logic for controlling the robot is trained using TensorFlow, executing different actions based on the various states of the model in the motion simulation module, ultimately achieving the desired goal and synchronizing with the real robot.

2.2 Vision Processing Module

The vision module is primarily used to identify information from the real-world model, extracting it in real-time and transmitting it to the server through the client for subsequent updates and corrections of the model information across different platforms and modules.

Specifically, environmental images are captured through a camera, and image recognition programs process the images and identify objects to extract positional information. This information can then be used as input for a deep reinforcement learning model, enabling decision-making and control.

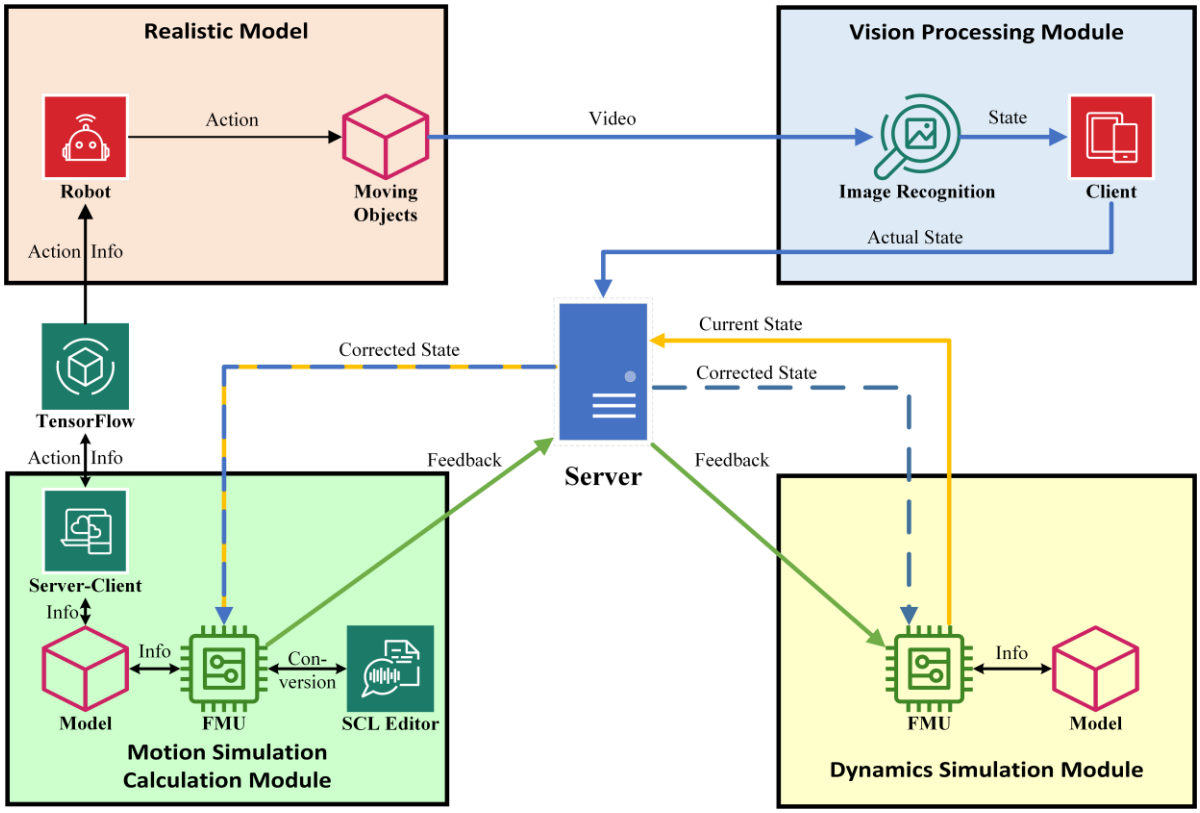


Figure 1: Framework of the Co-simulation system

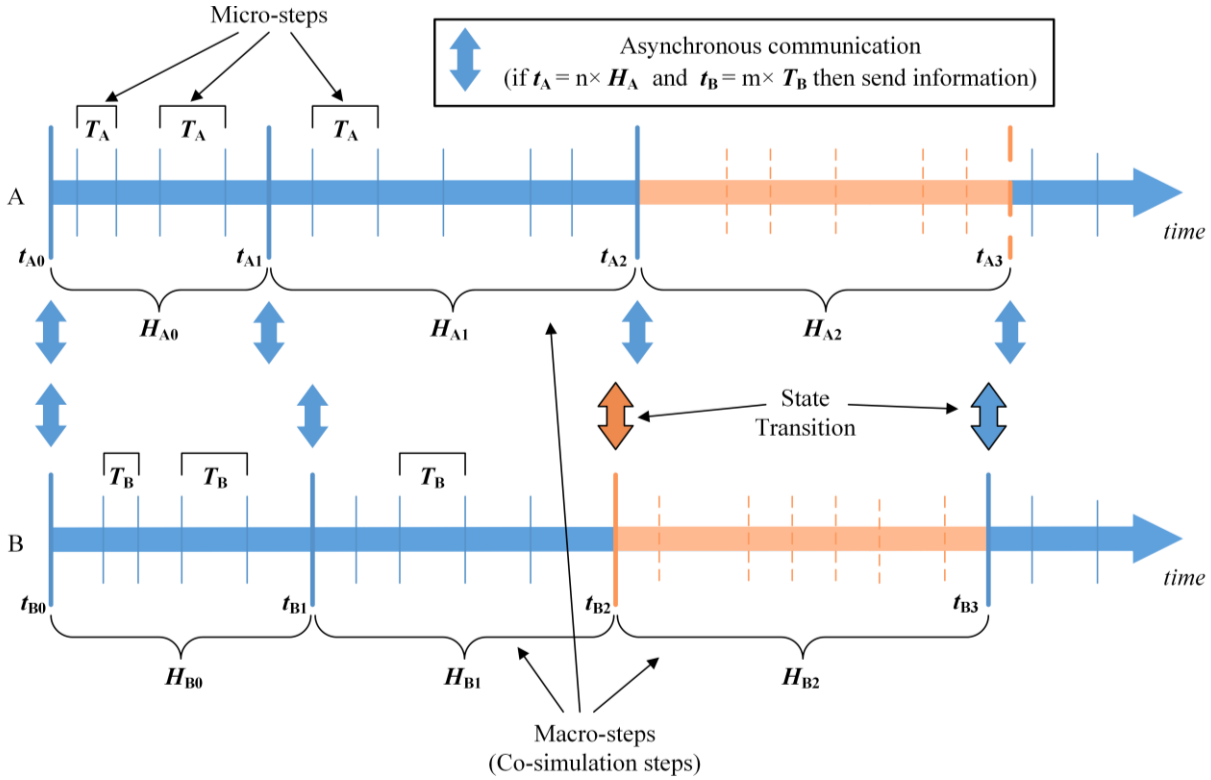


Figure 2: Exchange of information in the Co-simulation

2.3 Dynamics Simulation Module

As the primary module providing the simulation in the co-simulation framework, the dynamics simulation module needs to synchronize the critical information of moving objects with the physical model, and FMUs embedded within the model facilitate bidirectional information transfer with the simulation calculation module. The exact information exchanged is determined by the properties of the object and the input-output definitions of the FMU, accommodating the needs of different scenarios and models.

Although the dynamics simulation operates as a continuous-time system, synchronization with the physical system is constrained by unequal time steps and processing speed limitations, making it impossible to achieve perfect synchronization at every timestep. In such cases, interpolation or extrapolation methods are employed to exchange information at specific macro-steps, known as co-simulation steps, denoted as $\boldsymbol{H}$. As depicted in Figure 2, the simulation units exchange information at their respective macro-steps. Since FMUs allow for the exchange of coupling variables at macro-steps before individual integration, the system adopts a commonly used parallel (Jacobi) coordination method [4,17].

In Figure 2, Simulation Unit A corresponds to the dynamics simulation module, while Unit B corresponds to the motion simulation calculation module. Unit A operates as a time-driven continuous simulation that follows the physical laws and is updated according to the time step T_A. In most of the physical scenarios, the continuous evolution of the simulation system is based on solving ordinary differential equations (ODEs), as shown in Equation1:

$$s_A(t_{k+1}) = f_A\left(s_A(t_k), u_A(t_k)\right) \tag{1}$$

where s_A is the state of the Simulation Unit A and $u_A(t_k)$ is its input at timestep t_k. The state of Unit A continuously updates at each micro-step, making the simulation "time-continuous".

Similarly, in its free state, Unit B behaves like Unit A, running as a continuous-time simulation and also based on ODEs, as shown in Equation 2:

$$s_B(t_{k+1}) = f_B\left(s_B(t_k), u_B(t_k)\right) \tag{2}$$

At the initial simulation state, both Unit A and Unit B independently update their states at each micro-step. During these intervals, no information is exchanged between them except at macro-step points. At these points, the system operates purely under a time-driven simulation mechanism. However, when Unit B reaches a specific state (as shown in Equation 3), it triggers an event that changes its own state and alters the control state of Unit A. At this moment, an event-driven mechanism replaces the time-driven one, and the simulation transitions into a B-dominant mode, with the state updates occurring as described in Equation 4:

$$h_B(s_B(t_k)) = \mathrm{C} \tag{3}$$

$$s_A(t_{k+1}) = g_B\left(s_B(t_k)\right) \tag{4}$$

where the function h_B denotes a state or external input condition, and the event is triggered when the condition of Equation 3 is satisfied. Unit A ceases to evolve according to its own dynamics and is now controlled by the event logic of Unit B, operating under an event-driven mechanism during this phase. Regardless of the system's mode, Unit B continues to update Unit A's information at its macro-step points. At these moments, the system checks Unit B's state, as shown in Equation 5, to determine whether to resume Unit A's continuous-time dynamics simulation.

$$(s_A(T_{m+1}), s_B(T_{n+1})) = \mathrm{Update}(s_A(T_m), s_B(T_n)) \tag{5}$$

Although there is still a micro-step change within A in the event-driven state, this information is not exchanged at some macro-step point until the state of A and B is judged at the macro-step point, and if A returns to its free-running state, the exchange of information between the two units can continue.

2.4 Motion Simulation Calculation Module

This module is responsible for data processing and executing tasks step by step. Through the use of motion simulation and rendering, it carries out functions such as collision detection, motion range analysis, and reachability analysis. The imported FMU receives motion model data sent from the server and combines this data with a pre-trained deep reinforcement learning (DRL) policy model. The DRL model controls the simulated robot, which, in turn, governs the movement of the physical robot. Additionally, the states of both the robot and the object models serve as key event triggers, initiating the system's operation at critical points.

In this paper, the Deep Deterministic Policy Gradient (DDPG) algorithm within TensorFlow is used to develop a DRL model. This algorithm is particularly advantageous for solving optimization problems in continuous action spaces [23]. The DRL model is structured with a policy network for decision-making and a value network for evaluation purposes. Together, these networks allow the model to efficiently learn and handle complex robotic tasks such as grasping. The parameters and learned data of the DRL model are synchronized with the simulation platform through a built-in server-client architecture using TCP/IP communication.

During the system's operation, the DDPG algorithm makes real-time decisions based on the current state of the environment, which is continuously updated and transmitted to the control module. This data flow happens via the FMU within the motion simulation calculation module. The control module processes sensory inputs, including perception data and the robot's dynamic state, and passes this information on to the DRL module. The robot then performs specific actions based on the outputs from the policy network. Following this, the system adjusts the robot's future actions based on feedback from the updated state values.

Regarding information transmission, the FMU in the calculation module receives a constant stream of state data from the motion model. Once the robot executes a movement, its updated state directly influences the object model in the simulation, which in turn provides feedback to the dynamics simulation module via the FMU. At this point in the system's operation, task completion becomes the key determinant of the overall system state. FMUs play a critical role in enabling bidirectional data transmission and synchronisation, ensuring seamless communication between modules. This results in the realisation of a time-event hybrid-driven co-simulation within the framework, enhancing system robustness and flexibility.

3. Experiments

3.1 Hardware and Software Platform Construction

Based on the co-simulation framework proposed in Chapter 2, several modules have been implemented using software and hardware platforms adapted to the experimental environment. The actual task involves enabling the robot to grasp a regularly moving object (specifically a swinging wooden rod) within its workspace. In this study, the scene's visual processing module is implemented using OpenCV and associated libraries, while the dynamic simulation is carried out using Adams, a simulation software that offers excellent compatibility with FMU imports. Adams is primarily used to simulate the movement of the swinging rod and obtain the relevant motion parameters. The motion simulation calculation module is executed on Siemens Process Simulate software (abbreviated as the PS platform).

In the experiment, both the KUKA KR6 R700 robot model and the moving object model are imported into the PS platform to analyse the robot's range of motion and reachability. The six-axis angle values and other status information from the robot are transmitted in real time to the deep reinforcement learning training module using the TCP/IP communication protocol. Figure 3 shows the PS platform interface and the imported robot model.

This paper employs the TensorFlow machine learning platform and the DDPG algorithm as the deep reinforcement learning approach to train the robotic motion control strategy. The state space encompasses the robot's coordinates and the angles of each axis, as well as the coordinates and offset angles of the object model. The action space consists of the angular positions of the robot's motion axes. The robot's movements are based on the previous state space, followed by the execution of the next action. This process is repeated multiple times until the robot reaches the target endpoint.

The robotic models used in this study are KUKA's KR6 R700 and KR10 R1100-2, both of which are six-axis robots known for their high speed and precision [22]. KUKA robots utilize the open EthernetKRL control package to facilitate bidirectional data transmission between the actual robot and the PC. This is achieved by packaging the transmitted data into XML files, thereby enabling the real-time control of the robot's movements.

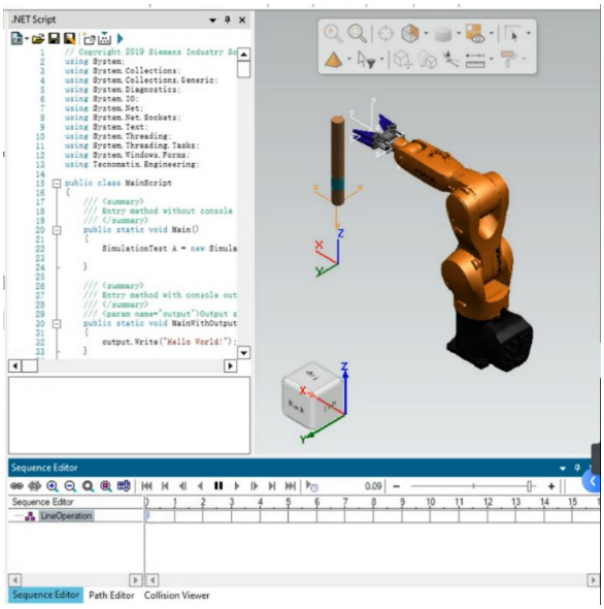


Figure 3: PS platform interface and imported KUKA robot model

3.2 Operational Process Framework

For the working scenario discussed in this paper, by aligning the experimental software and operations with Figure 1, a complete operational workflow of the co-simulation system can be obtained, as illustrated in Figure 4.

The preparation phase mainly involves setting up the project model and adjusting parameters. The environment includes the actual model and the runtime program files of the host, with the essential software and hardware libraries preloaded. The scene is equipped with a dedicated workstation and camera for real-time data acquisition of robot operations and motion models.

Different models focus on various types of data; the motion object model is defined with detailed physical properties and motion behaviours, whereas the robot model is configured with its physical parameters and control logic.

The training phase utilises TensorFlow to process the robot's grasping strategies. The training environment is rendered within Siemens PS platform, and the DDPG algorithm is employed for calibration to ensure the effectiveness of both the model and the algorithm. A specific reward function R is set for the environment in this paper, as shown in Equation 6, where s represents the robot's state, D denotes the distance to the target, and t refers to the time step.

$$R_t = \begin{cases} -1, & s \text{ is out of bound} \\ -1, & s \text{ is NearMiss} \\ -2, & s \text{ is a collision} \\ -10, & t > 256 \\ 100, & s \text{ is reaching the object} \\ \frac{0.1}{D_s}, & D_s < D_{s'} \\ -0.01, & \text{otherwise} \end{cases} \quad (6)$$

After initial training tests, the trained strategy models are generated and stored for future use. During this process, Adams software is used for dynamic simulation of the moving objects. OpenCV, through the laboratory's camera system, captures real-time image data of the moving objects, as shown in Figure 5. This data is used as a dataset to train the robot's reinforcement learning model in TensorFlow.

The FMU generation is primarily achieved through JSON files and program execution, automatically generating FMU files that meet the required specifications. This addresses issues of low reusability and portability in simulation models. Input and output parameters from different software must be set, and the corresponding modifiable JSON configuration files must be adjusted to configure nodes and links within the distributed system. These include project basic information, node information, and link information. The generated FMU files are then used for information transmission and synchronisation between Adams and the PS platform, facilitating the co-simulation of moving objects.

The testing and validation phase focuses on the co-simulation between two industrial software platforms, PS and Adams. The trained grasping strategy model is preloaded, and robot motion tests are initiated. Once the client and server are activated, the system begins the co-simulation process. TensorFlow processes the real-time camera data received via TCP/IP, generates control commands, and feeds them back into the PS platform and Adams for simulation.

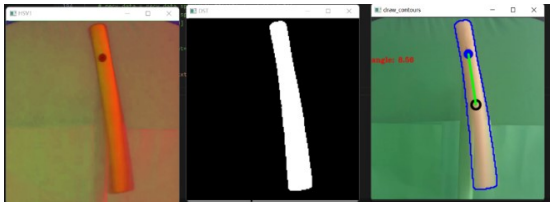


Figure 5: OpenCV recognises recorded moving objects

During the co-simulation, the server communicates with both the PS platform and TensorFlow via .NET Script, processing the trained models and transmitting control commands. The trained models in PS are invoked through the program, conducting real-time kinematic simulations. The simulations in Adams validate the accuracy and reliability of all modules and algorithms.

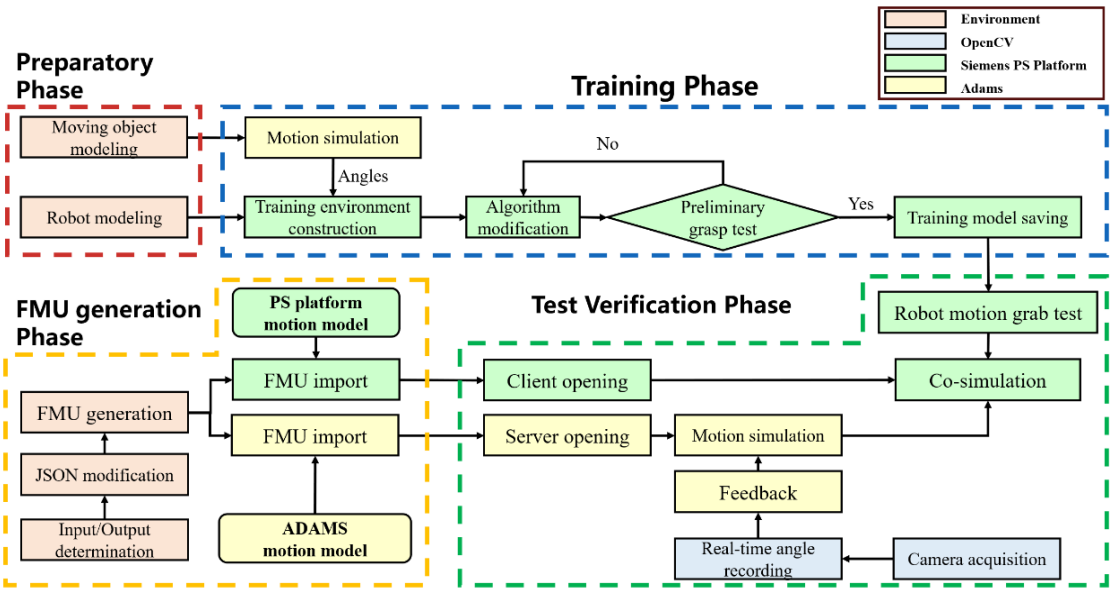


Figure 4: Operation flow of the Co-simulation system

3.3 FMU Generation and Import

As previously mentioned, an FMU can be regarded as a black box, with its storage structure akin to a zip file, meaning the method of file generation is relatively fixed. To avoid compiler library limitations, MinGW is employed for compilation.

The most critical information in an FMU is the number of nodes and their corresponding input and output types. When generating multiple FMUs, the input and output must be matched to ensure the validity of the generated FMU. Specifically, a JSON configuration file is set up in this paper, with serverIP, nodeNum, and nodeLists being key search information. The nodeList contains nodeName (the folder name for the configuration file), UID, input, and output. For each port, the corresponding input and output must be mapped to ensure normal operation. The FMUs generated by MinGW's Makefile include a DLL dynamic link library, a series of C files and header files containing models and configurations, and an XML file with important identifying information.

The FMUs generated in this paper are primarily for use in the Adams software and the PS platform.

While the basic structure of the FMUs is similar, the operations within different software platforms vary. The specific process is illustrated in Figure 6. In Adams, the FMU is treated as a special type of system variable that needs to be bound to the existing variables within Adams through the Control component. If multiple input/output pairs are required, no empty variables are allowed. On the PS platform, since the software lacks a built-in method for reading FMUs, the files must be manually extracted after decompression and copied into the software's FMI folder. Similarly, the FMU is regarded as a special variable in the PS platform, with the variable type names predefined during FMU generation, corresponding to the generated XML file. Additionally, the FMU must be called through the SCL Editor on the PS platform, and all information is passed through this intermediary station before it can be converted into variables usable by the script.

It is worth noting that to ensure the effectiveness and clarity of FMU information transmission, all original variables are closely bound to the model within the software, and once imported, the FMU becomes an integral part of the model. The synchronisation of the FMU enables the synchronisation of model states across different platforms.

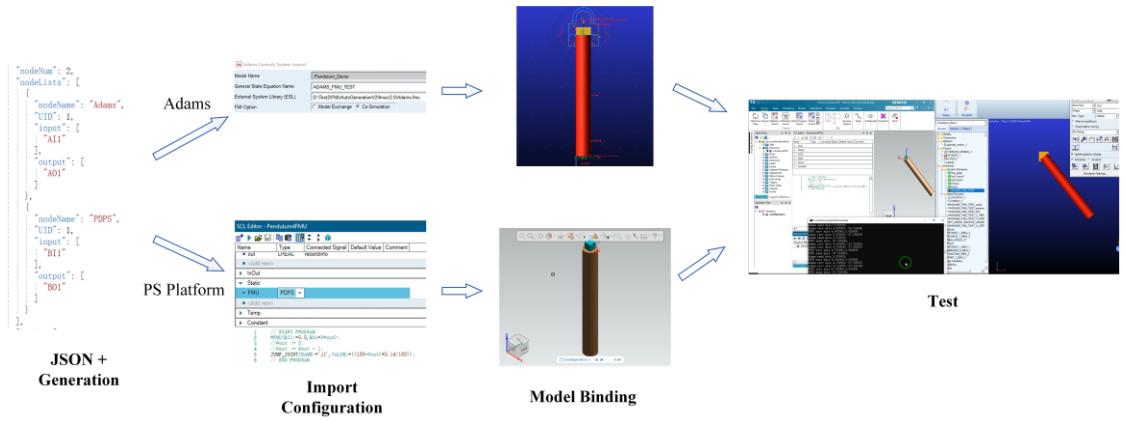


Figure 6: FMU generation and import process

3.4 Co-Simulation Test

As can be seen from Figures 1 and 4, the co-simulation testing involves three main components: Adams, the PS platform, and OpenCV. In this scenario, the robot is tasked with attempting to grasp a continuously swinging stick. Both Adams and the PS platform share the same motion model, namely the swinging stick mentioned earlier, where Adams handles the physical simulation as the active agent, while the PS platform acts as the recipient of the motion state. Simultaneously, OpenCV processes real-world information about the swinging stick and helps correct the angle values transmitted to the PS platform. The PS platform also returns feedback information to Adams. Once the temporal and angular data are determined, the pre-trained robot grasping model proceeds to execute the grasping actions step by step. The multiple stages of the process and platform interface are shown in Figure 7.

Specifically, the system includes a server that connects the three components. Unlike a typical TCP/IP server, this server can also transmit FMU data. The key to achieving this functionality is the accompanying JSON configuration file, which is broadly similar to the one described in Section 3.2 but includes an additional key element: linkNum and linkLists. In this system environment, there are three different ports, so the number of links is 3—two for the FMUs in the software and one for the OpenCV

connection. With the aid of this configuration file, the server can facilitate information exchange with the FMUs.

Once the server is activated, the robot control logic, OpenCV video feed, Adams motion simulation, and PS platform simulation are sequentially launched. When all components are connected, the platforms synchronise their current states, and the robot in the PS platform executes and outputs the robot's grasping motion path.

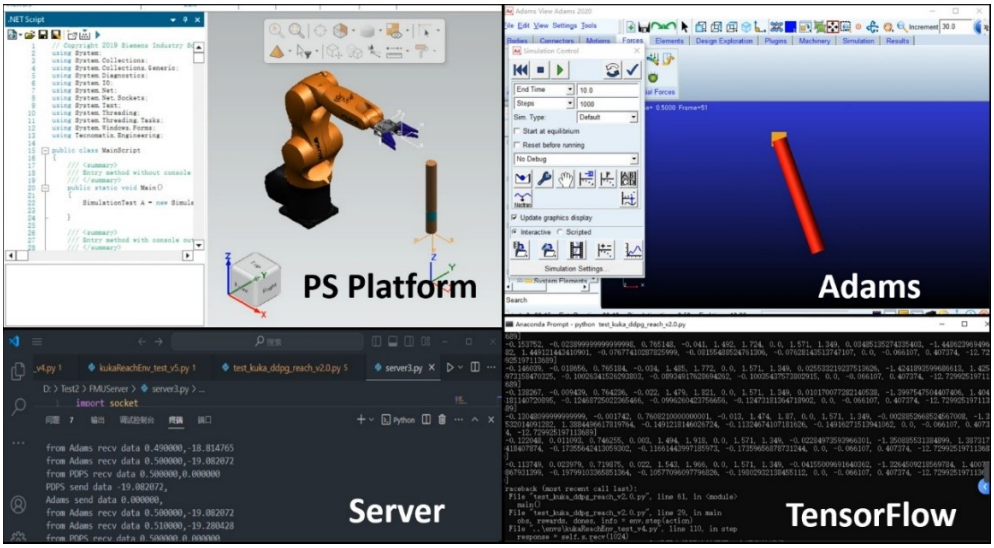


Figure 7: Interface between multiple parts of the Co-simulation process and the platform's operation

3.5 Reality Check and Results

The robot's motion path, obtained from the PS platform, is exported and encapsulated into an XML file, which is then transmitted to the actual robot's program files for reception. By mapping the relationship between the world coordinate system or the six axes of the robot, the robot can follow the grasping path displayed on the PS platform. The pseudocode for the robot path execution program, based on six-axis information transmission, is shown in Table 1.

Table 1. Pseudocode: KUKA Robot Motion

Define six axes A1~A6 and initialize the array XP1 to represent the initial coordinate positions.
Move the robot point-to-point to the initial position to reset. Initialize and execute the internal XML file to receive TCP signals. LOOP: IF the received data for XP1.A1 is empty: Exit the loop. END IF Sequentially receive data for the coordinate axes A1~A6 from the XML file. Update the XP1.A1~XP1.A6 data. Perform point-to-point movement to the position indicated by XP1. END LOOP End of program.

When the initial states of the real swinging stick and the stick model in Adams are similar, the motion of the stick obtained from the simulation software and the robot's grasping path align well with the real stick and robot, enabling the robot to perform the actual grasping task. The real-world robot's grasping process, viewed from two perspectives, is shown in Figure 8.

To evaluate the accuracy and effectiveness of the framework, Table 2 summarizes the results obtained from multiple grasping trials, covering various initial angles, mean grasping distance errors, average execution time, and success rate. For each initial angle, we conducted 20 trials.

Table 2. Summary of Robot Grasping Results

Initial Angles	Grasping Distance Errors(mm)	Execution Time (s)	Grasping Success Rate (%)
30°	3.2	14.8	95
40°	3.8	15.6	95
50°	5.0	15.4	90
60°	5.8	16.6	80

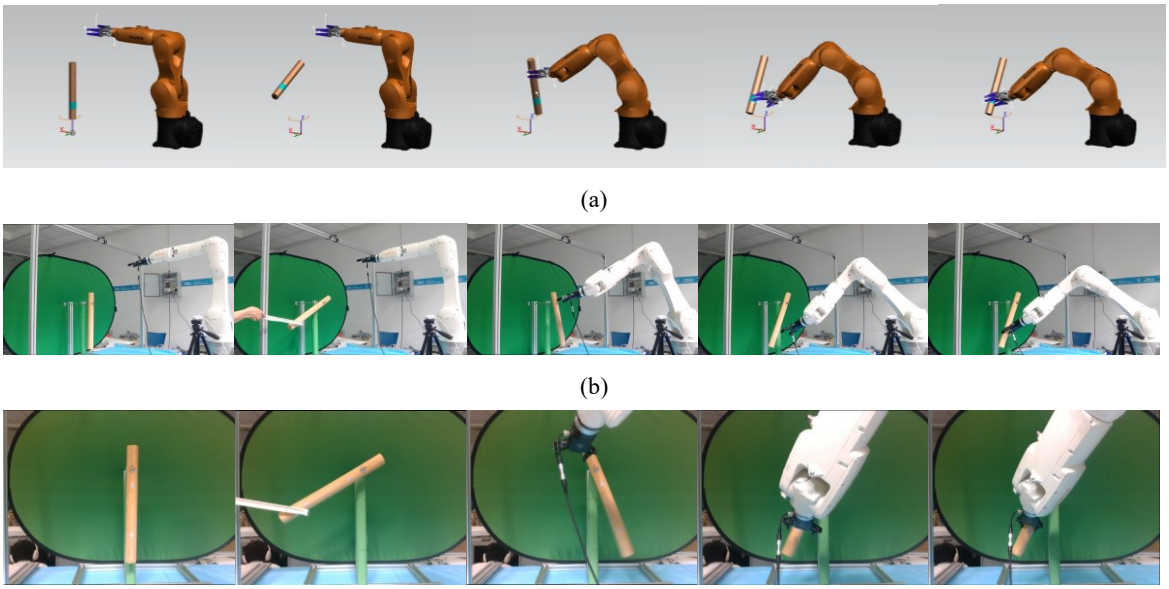
The experimental results highlight the framework's effectiveness in enhancing robotic grasping performance under varied initial conditions, demonstrating its adaptability and robustness. As shown in Table 2, the average grasping distance error remained low, with only slight increases observed at larger initial angles. Execution times remained efficient and consistent,

while success rates were high overall, achieving up to 95% at 30° and 40° initial angles. Although a minor decline in success rate was observed at a 60° angle, the hybrid time-event-driven mechanism successfully maintained synchronization and coordination across platforms, even in challenging scenarios.

4. Conclusions

This paper proposes an innovative time-event hybrid-driven co-simulation framework based on the FMI standard. By integrating platforms such as TensorFlow, OpenCV, and Adams, the framework addresses key limitations in robotic systems, including data synchronization, model portability, and time stability across platforms.

The use of FMI enhances model portability and modularity, enabling seamless integration between diverse simulation tools and supporting scalable system design. The hybrid time-event mechanism combines the advantages of both time-driven and event-driven methods, effectively capturing dynamic robotic behaviors and improving real-time simulation performance in complex environments. Experimental results validate the framework's ability to enhance path planning and grasping accuracy, demonstrating its potential in multi-platform collaborative tasks. Future work will further optimize synchronization mechanisms and expand the framework's application in increasingly complex scenarios.



(a)

(b)

(c)

Figure 8: Real Robot Grasping Test (a. Virtual test, b. Side view, c. Front view)

References

[1] LI W, DENG Z, GE J, et al. Research progress of robot joint space trajectory planning[J]. Mechanical design and manufacturing engineering, 2022, 51(10): 15-23.

[2] Desai A, Saha I, Yang J, et al. DRONA: a framework for safe distributed mobile robotics[C]//Proceedings of the 8th International Conference on Cyber-Physical Systems. 2017: 239-248.

[3] Wang L, Liu M, Meng M Q H. Real-time multisensor data retrieval for cloud robotic systems[J]. IEEE Transactions on Automation Science and Engineering, 2015, 12(2): 507-518.

[4] Gomes C, Thule C, Larsen P G, et al. Co-simulation of continuous systems: a tutorial [J]. arXiv preprint arXiv:1809.08463, 2018.

[5] Gomes C, Thule C, Broman D, et al. Co-simulation: a survey[J]. ACM Computing Surveys (CSUR), 2018, 51(3): 1-33.

[6] Gomes C, Thule C, Broman D, et al. Co-simulation: state of the art[J]. arXiv preprint arXiv:1702.00686, 2017.

[7] Camus B, Galtier V, Caujolle M. Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO[C]//2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS). IEEE, 2016: 1-8.

[8] Mahler J, Liang J, Niyaz S, et al. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics[J]. arXiv preprint arXiv:1703.09312, 2017.

[9] LU J, PING X. An optimal time-impact trajectory optimization algorithm for manipulator [J]. Mechanical Science and Technology, 2019, 38(10): 1548-1554.

[10] Koopman P, Wagner M. Challenges in autonomous vehicle testing and validation[J]. SAE International Journal of Transportation Safety, 2016, 4(1): 15-24.

[11] Aslam K, Chen Y, Butt M, et al. Cross-platform real-time collaborative modeling: An architecture and a prototype implementation via emf. cloud[J]. IEEE Access, 2023, 11: 49241-49260.

[12] Paulavičius R, Grigaitis S, Filatovas E. A systematic review and empirical analysis of blockchain simulators[J]. IEEE access, 2021, 9: 38010-38028.

[13] Fan C, Ghaemi S, Khazaei H, et al. Performance evaluation of blockchain systems: A systematic survey[J]. IEEE Access, 2020, 8: 126927-126950.

[14] Marlier N, Brüls O, Louppe G. Simulation-based Bayesian inference for robotic grasping[J]. arXiv preprint arXiv:2303.05873, 2023.

[15] FMI Project Group. FMI Tools [EB/OL]. https://fmi-standard.org (2023-12-01).

[16] Blochwitz T, Otter M, Arnold M, et al. The functional mockup interface for tool independent exchange of simulation models[C]//Proceedings of the 8th international modelica conference. 2011: 105-114.

[17] Eguillon Y, Lacabanne B, Tromeur-Dervout D. IFOSMONDI co-simulation algorithm with Jacobian-free methods in PETSc[J]. Engineering with computers, 2022, 38(5): 4423-4449.

[18] Liu W, Zhao Z, Shi B, et al. Hybrid Time and Event Co-simulation Framework for Power Electronics Systems[C]//2023 IEEE 14th International Symposium on Power Electronics for Distributed Generation Systems (PEDG). IEEE, 2023: 1055-1058.

[19] Al-Hammouri A T. A comprehensive co-simulation platform for cyber-physical systems[J]. Computer Communications, 2012, 36(1): 8-19.

[20] Benedikt M, Holzinger F R. Automated configuration for non-iterative co-simulation[C]//2016 17th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE). IEEE, 2016: 1-7.

[21] Völker L. Untersuchung des Kommunikationsintervalls bei der gekoppelten Simulation [M]. KIT Scientific Publishing, 2014.

[22] KUKA Roboter GmbH, KR QUANTEC ultra[EB/OL].(2022-09-23)[2023-05-13].https://www.kuka.com/-/media/kuka-downloads/imported/8350ff3ca11642998dbdc81dcc2ed44 c/0000210361_zh.pdf

[23] Lillicrap T P. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.