Model-Based Systems Engineering Applied to the Detection and Correction of Object Slippage within a Dexterous Robotic Hand

Charles A. Meehan
Department of Mechanical Engineering
University of Maryland
College Park, USA
cmeehan2@umd.edu

John S. Baras
Department of Electrical and Computer Engineering
and the Institute for Systems Research
University of Maryland
College Park, USA
baras@umd.edu

2021 IEEE International Systems Conference (SysCon) | 978-1-6654-4439-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/SysCon48628.2021.9447100

***Abstract*—Slip detection and correction plays a very important role in robotic manipulation tasks, and it has long been a challenging problem in the robotic community. Further, the advantage of using systems engineering tools and framework to approach a solution and/or modeling of robotic tasks is not often pursued. In this paper, we use Model-Based Systems Engineering techniques to verify system requirements and validate stakeholder requirements for the problem of detecting and correcting for object slippage within a dexterous five-fingered robotic hand. We will discuss how the work accomplished in our laboratory was transferred to a simulated environment and how this simulated environment built in CoppeliaSim was connected to a systems engineering software, Cameo Systems Modeler. Measures of effectiveness were created from the stakeholder requirements for the slippage problem which allowed us to validate the robotic simulation that was built. Structural diagrams of the robotic system and environment were built along with behavioral diagrams of the simulation. Further, we used the connection of Cameo Systems Modeler and CoppeliaSim to track the measures of effectiveness for our robotic task which provided us with a complete systems engineering framework for the problem from the requirements phase through the implementation phase. Our main goal is to show the advantages of following a systems engineering framework to complete a robotic task through the connection of Cameo Systems Modeler and CoppeliaSim.**

***Index Terms*—MBSE, Robotics**

I. Introduction

In many cases, the approach to completing a robotic task deals with trial and error experiments completed in the laboratory. One reason for working out problems this way is due to the difficulty of fully simulating robotic tasks and accommodating all the uncertainties of the real world in simulation such as equipment failure or even the use of a proper model of friction during a grasping task. Another issue is being able to verify that the robotic system does what it is supposed to or validate that the robotic system does what the user first wanted it to do. This is why it is important to approach robotic tasks by first writing and keeping track of the requirements of the user (what the user wants the robotic system to do), building structure and behavior models of the robotic system, and being able to verify and validate the robotic system against stakeholder (user) and system requirements. This approach forms the basis of Model-Based Systems Engineering, and this paper will show how this was applied to the robotic problem of slippage detection and correction of object slippage within a five-fingered robotic hand.

The contributions of this work are as follows. First, the measures of effectiveness were built from the stakeholder requirements for the problem of detection and correction of object slippage within a robotic hand. Second, structure and behavior diagrams were constructed in Cameo Systems Modeler in order to ensure that the robotic system and the steps in solving the task in a robotic simulation were well defined. We were able to connect Cameo Systems Modeler to a popular robotic simulation software, CoppeliaSim, which allowed us to track output metrics and record them within Cameo Systems Modeler. Finally, we were able to verify the simulation requirements and validate the stakeholder requirements for the robotic task and show the advantage of having the systems engineering software tool connected to a robotic simulation software.

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III, we describe the Model-Based Systems Engineering (MBSE) approach to solving robotic problems. The system architecture is displayed in Section IV. In Section V, we discuss the process and reason for moving from experimentation in the laboratory to experimentation in simulation. Section VI deals with the verification of the simulation against the simulation requirements and the validation of the robotic system against the stakeholder requirements. Lastly in Section VII, we conclude the paper.

II. Related Work

Formal verification and validation of autonomous systems such as the robotic system used in this paper is necessary to guarantee system safety and robustness. In [1], formal models of verification and validation such as metric temporal logic and Timed Automata are referenced as ways to verify that the robotic system does not fail the system requirements and

Research supported in part by ONR grant N00014-17-1-2622.

validates that the system does what the user wants the system to do. However, in order to deploy these verification and validation methods, it is necessary to have direct feedback during system testing so that the failures or passes found during verification and validation are fed back into the system modeling software.

As stated in [1], the Robotic Operating System (ROS) does not come with support of verification and validation models. There have been some efforts by [2], [3], [4], but the overall nature of the ad hoc writing of ROS nodes to run a robotic task does not lend itself to applying formal verification and validation methods. There are some systems that authors built with verification of properties on the top of ROS during runtime, [5] and [6], but they do not rely on ROS itself [1]. This is why it was necessary to build the connection from CSM and CoppeliaSim where the verification and validation methods are built in with CSM and used through the connection to the robotic simulation in CoppeliaSim.

Further in order to follow the MBSE framework, it is key to have the models and requirements built for the system of interest connect to a robust simulation environment. As stated in [7], the problem with the holistic approach of MBSE is the fact that many of the powerful simulation tools are not seamlessly connected to the systems engineering tools which does not allow the user to perform verification or validation of their system all from one software environment. In [7], the authors discuss the notion of "Digital Twin" which can be thought of as a virtual entity that is the same as a real world entity. They created experimentable digital twins (EDTs) in order to combine the ideas of Digital Twins with simulation technology so that they can connect the systems models with simulation environments.

Ultimately, their entire framework is built on what they call a Versatile Simulation Database which allows them to integrate various data sources, simulation systems, visualization, interaction and feedback devices of the EDTs [7]. In this paper, we have focused on creating the connection between a particular systems engineering software, Cameo Systems Modeler, and a popular robotic simulation software, CoppeliaSim. This follows the efforts of Schluse, Atorf, and Rossman [7] where they are creating a database of various applications that can connect systems engineering models with powerful simulation software.

III. MBSE Approach to Solving Robotics Problems

In 2007, the INCOSE Systems Engineering Vision 2020 defined MBSE as "the formalized application of modeling to support system requirements, design analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [8]. MBSE promotes the use of going from Stakeholder requirements to the verification and validation stages by building and using system models or a set of models which are usually built using a Systems Engineering software tool such as Cameo Systems Modeler (CSM). Using the MBSE approach to system development can result in "significant improvements in systems requirements, architecture, and design quality; lower the risk and cost of system development by surfacing issues early in the system definition; enhance productivity through reuse of system artifacts; and improve communications among the system development team" [8]. These possible improvements to the system modeling process such as lowering the risk and cost of surfacing issues aligns well with the goals of using simulation for robotic tasks.

For building the robotic simulation, the process started with developing stakeholder requirements. The stakeholder requirements that were developed apply to the development and use of a robotic system that solves the slippage detection and correction problem. Measures of Effectiveness (MOEs) and the simulation requirements were derived from the stakeholder requirements. From the simulation requirements, it was possible to build the simulation architecture and the simulation design through structural and behavioral diagrams. These diagrams were programmed using the Systems Modeling Language (SysML) which is a graphical modeling language used by MBSE practitioners when developing system models [9]. These diagrams were visualized in Cameo Systems Modeler. The robotic simulation was implemented in a robotic simulator program and connected to CSM in order to perform verification and validation. Throughout the systems engineering process, the simulation was validated against the stakeholder requirements and verified against the simulation requirements.

A. Stakeholder Requirements

The development of the stakeholder needs for the robotic system was driven by the following objective: to autonomously detect and correct for object slippage within a robotic dexterous hand while weight is being added to the object. This led to the following Table I of stakeholder requirements.

B. Measures of Effectiveness

The measures of effectiveness (MOEs) for the robotic system which were important output metrics to track from the slippage detection and correction problem were derived from the stakeholder requirements. MOEs are important metrics that can be used to validate the system. They are "operational measures of success that are closely related to the achievement of the mission or operational objective being evaluated, in the intended operational environment under a specified set of conditions" [8]. The MOEs are listed in Table II.

C. Simulation Requirements

A major part of this work was to create a robotic simulation of the slippage correction and detection problem that replicated the experiment which was worked on in the lab. Therefore, there was a need to create simulation requirements so that the elements built for the simulation could be verified. This way items that were necessary to include in the simulation were included, and elements are not built that are not needed. These requirements were derived from the stakeholder needs and requirements. The simulation requirements are listed in Tables III and IV where Table IV is shown in Appendix A.

Table I: Stakeholder Requirements Table.

ID	Stakeholder Requirements	Stakeholder Requirement Description
SHR.1.1	Performance Requirements and Constraints	
SHR.1.1.1	Detect Object Slippage	The Robotic System shall use haptic sensors to detect object slippage.
SHR.1.1.2	Use Anthropomorphic Robotic Hand	The Robotic System shall use a five-fingered dexterous robotic hand.
SHR.1.1.3	Autonomous Detection and Correction	The Robotic System shall detect and correct object slippage autonomously.
SHR.1.1.4	Stop Object From Falling	The Robotic System shall stop a grasped object from falling.
SHR.1.1.5	Accurate Slippage Detection	The Robotic System shall detect object slippage with a probability $> 0.97\%$.
SHR.1.1.6	Time of Slippage Detection	The Robotic System shall detect object slippage within 5 seconds.
SHR.1.1.7	Time of Slippage Correction	The Robotic System shall correct object slippage within 200 ms.
SHR.1.1.8	Object Correction Displacement	The Robotic System shall maintain an object displacement < 0.05m during correction stage.
SHR.1.1.9	Maintain Object Orientation	The Robotic System shall not rotate more than 45 degrees about its center of mass during correction.
SHR.1.1.10	Max Force Applied	The Robotic System shall apply a correction force less than the safety margin of the object.
SHR.1.1.11	Replicate Laboratory Experiment in Simulation	The Robotic System shall be able to replicate the slippage detection and correction problem in simulation.

Table II: Measures of Effectiveness Table.

Metric	Definition	Threshold Value	Objective Value
Probability of Detection of Object Slip (PDS)	The probability of detecting object slippage from the robotic hand given the object does slip.	$> 97\%$	$> 99\%$
Probability of False Detection of Object Slip (PFS)	The probability of detecting object slippage from the robotic hand given the object does not slip.	$< 3\%$	$< 1\%$
Time of Object Slip Detection (TOS)	The time it takes to detect object slippage from the robotic hand.	$<$ 5secs	$<$ 2secs
Time of Object Slip Correction (TOC)	The time it takes to correct for object slippage from the robotic hand.	$<$ 200ms	$<$ 100ms
Object Displacement (OD)	The total displacement of the object during correction.	$<$ 0.1m	$<$ 0.05m
Object Rotation (OR)	The rotation of the object about its center of mass relative to the world frame during correction.	$<$ 45degs	$<$ 20degs
Max Force Applied To the Grasped Object (MGF)	The maximum force applied by the robotic hand to the grasped object during correction.	$<$ Safety Margin	$<$ Crushing Threshold
Object Held (OH)	Boolean response which equals 1 if the object did not fall from the grasp during the experiment or 0 if not.	1	1

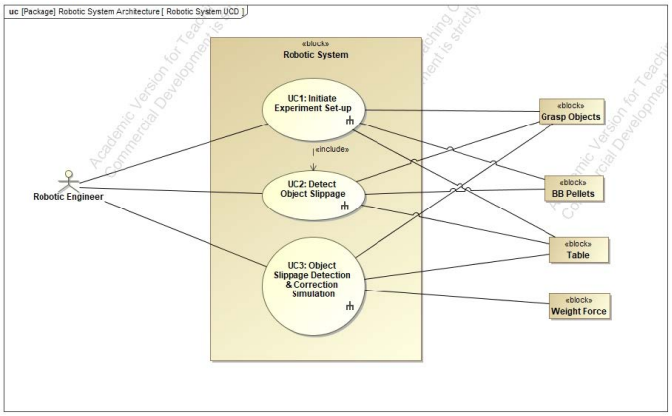

Figure 1: Use Case Diagram developed for the slippage detection and correction problem.

IV. System Architecture

While the system structural diagrams model the robotic system that was used for the slippage detection and correction problem, the behavior diagrams modeled the behavior of the laboratory experiments and simulation.

A. Slippage Detection and Correction Use Cases

A use case diagram was created in order to present the list of use cases that were needed in order to model the behavior of the robotic system in solving the slippage detection and correction problem. In general, the purpose of a use case diagram is to convey externally visible services that a system provides and the actors plus environment that are involved with those use cases [9]. The use case diagram built for this problem is presented in Figure 1.

The first and second use cases deal with the set up and actions of the robotic system to solve the slippage detection and correction problem in the laboratory. The third use case which is the focus of this work deals with the robotic simulation built to replicate the slippage detection and correction problem. The only actor involved in these three use cases was the robotic engineer who is responsible for running the experiment and

Table III: Simulation Requirements Table.

ID	Simulation Requirements	Simulation Requirement Description
S.1.1	System Capability Requirements	
S.1.1.1	System Functional Requirements	
S.1.1.1.1	Simulate Robotic Equipment	The Robotic System Simulation (Sim) shall simulate the Universal Robots UR10, the Shadow Dexterous Hand, and the BioTac SP sensors.
S.1.1.1.2	Simulate Grasp Objects	The Robotic System Sim shall simulate a cup as the object to grasp.
S.1.1.1.3	Simulate Laboratory Environment	The Robotic System Sim shall simulate an environment similar to the ARC laboratory.
S.1.1.1.4	Import Unified Robot Description Format Files	The Robotic System Sim shall be able to import robotic URDFs.
S.1.1.1.5	Calculate Robot Kinematics	The Robotic System Sim shall calculate the kinematics for the UR10 and the Shadow Hand Robot.
S.1.1.1.6	Calculate Dynamics	The Robotic System Sim shall be able to calculate the dynamics for all the objects in the simulation.
S.1.1.1.7	Use Physical Engines	The Robotic System Sim shall use physical engines to calculate the dynamics in the simulation.
S.1.1.1.8	Robotic Arm Movement	The Robotic System Sim shall control the movement of the UR10.
S.1.1.1.9	Robotic Hand Movement	The Robotic System Sim shall control the movement of the Shadow Hand Robot.
S.1.1.1.10	Add Weight Force	The Robotic System Sim shall add force to the center of mass of the object in the negative Z direction.

simulations. The environment on the right side of the diagram is composed of equipment that was involved with the system of interest but not a part of the system of interest such as the bb pellets or the simulated weight force.

B. The Robotic System Context-Level and System-Level Block Definition Diagram

Structural diagrams were created to model the robotic system context-level and system-level architecture. These diagrams show the structure decomposition of the robotic system and include the components used in the lab and in simulation. Block definition diagrams (BDDs) are used by systems engineers to show structural components and how they are connected. The purpose of BDDs is to convey system decomposition and the structural relationships between the elements of definition which are model elements displayed on BDDs such as blocks, actors, value types, constraint blocks, flow specification, and interfaces [9]. The context-level BDD for the robotic system used in the slippage detection and correction problem is shown in Figure 2.

The left side of the context-level diagram is the structural decomposition of the robotic system which includes blocks for the Shadow Hand and UR10 as well as blocks that represent the simulation software under the computer block. The user is listed under the center of the diagram. The right side of the context-level diagram is the structural decomposition of the environment which includes blocks for the grasp objects and other items that interact with the system of interest. Also, this includes items that are different for the simulation versus in the real lab.

The system-level BDDs go further into the decomposition of the system of interest by showing the structural decomposition of the Shadow Hand Robot and UR10 robot. For example, the BDD for the Shadow Hand Robot is shown in Figure 3. The BDD for the Shadow Hand Robot displays the joints and links of the robot as well as a block for the BioTac SP sensors which are attached to the Shadow Hand fingertips. This structural decomposition came from the URDF provided by the Shadow Hand Robot Company. The BDD created for the UR10 also displayed the joints and links of the robot and came from a URDF provided by the Universal Robots Company. These structural BDDs provide the user or stakeholder with a way to understand and visualize the components involved in the system of interest. Elements from these BDDs will be used in the behavioral diagrams that describe the behavior of this robotic system.

C. Simulation Activity and Sequence Diagrams

Behavior diagrams were built for the laboratory work, but the focus of this section will be on the behavior diagrams created for the robotic simulation which was Use Case 3. First, a activity diagram was created to model the actions taken in order to accomplish the robotic simulation. The purpose of an activity diagram is to provide a dynamic view of the system through the use of a sequence of event occurrences or sequence of behaviors over time [9]. The activity diagram for the robotic simulation is shown in Figure 4. The activity diagram further shows what systems and users are involved with the actions of the activity of Use Case 3.

The steps as shown in the activity diagram start with the robotic engineer initiating the robotic simulation from CSM. The engineer starts the simulation from an instance in CSM which triggers a MATLAB script. This script triggers a python script which starts the robotic simulation in CoppeliaSim. The robotic simulation starts with moving the UR10 to a specified position where the Shadow Hand can grasp a cup. Once at the pre-grasp location, the Shadow Hand grasps the cup, and the UR10 lifts the Shadow Hand and cup. A force is added to the center of mass of the grasped cup in order to simulate weight being added similar to when bb pellets were added

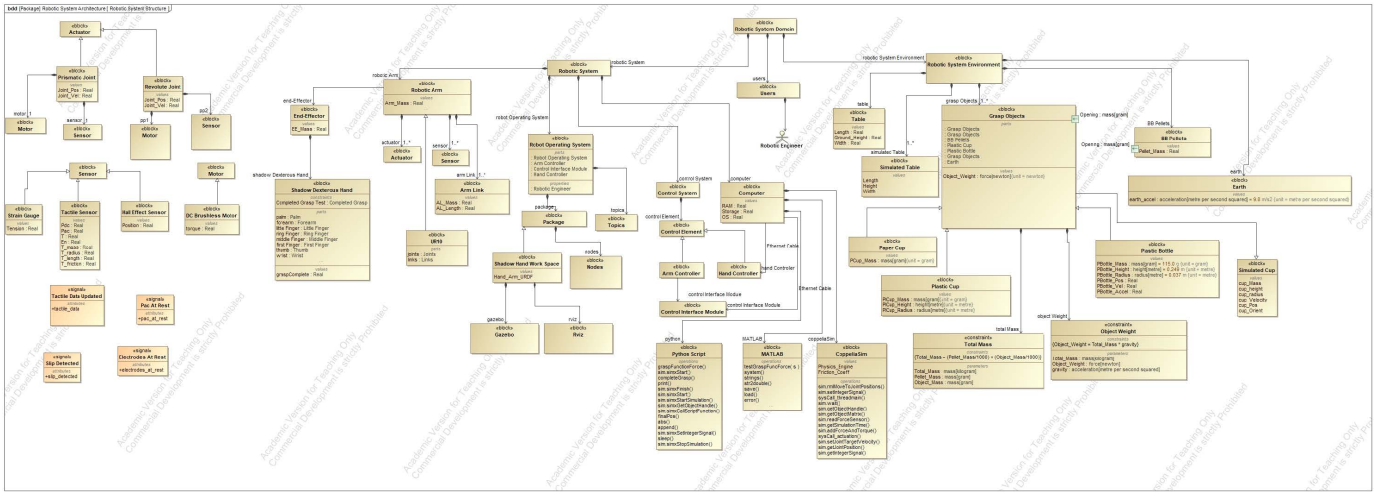

Figure 2: Context-Level BDD of Robotic System.

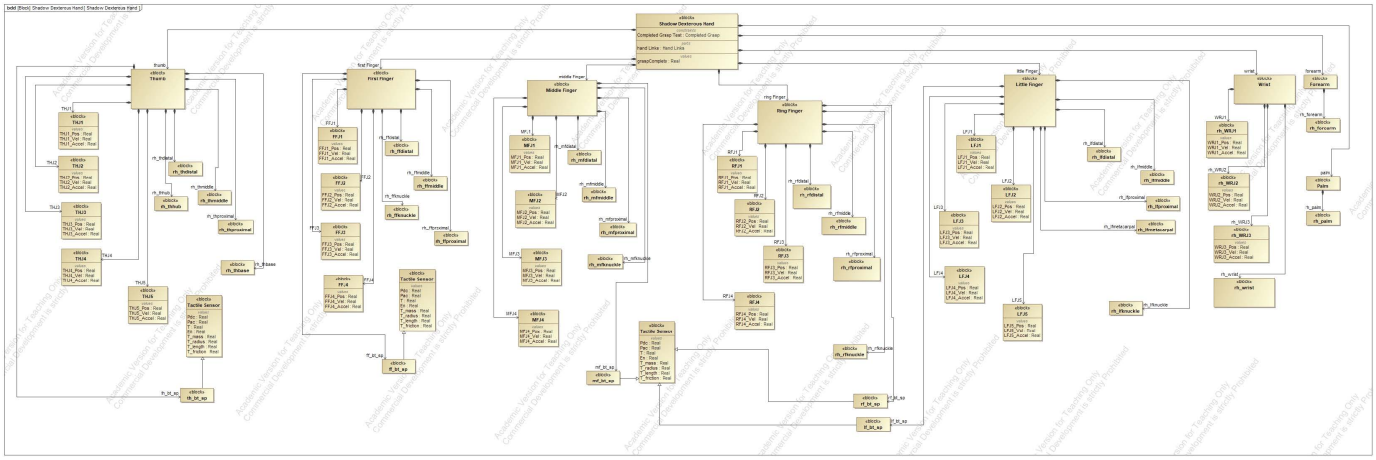

Figure 3: System-Level BDD of the Shadow Hand Robot.

in the lab. Force sensors on the fingertips are used to sense slippage of the cup, and the second joint of the fingers and thumb is rotated to correct for that slippage. Once these tasks are completed, the simulation is shut down and returns output metrics to CSM.

The other behavior diagram created to model the behavior of the robotic simulation at a lower level where messages are being sent between elements of the system is a sequence diagram. The purpose of sequence diagrams like activity diagrams express the sequence of behaviors of a system, but a sequence diagram can be used to focus on how elements of blocks interact with other elements via operation calls and asynchronous signals [9]. The messages displayed in the sequence diagram for the robotic simulation are the messages that were used in the scripts that control the robotic simulation. This diagram shown in Appendix B; which is split up into Figures 11, 12, 13, 14, 15, 16, and 17; provides a deeper view of how the robotic simulation is able to replicate the slippage detection and correction problem. Further, one major advantage of the sequence diagram is the ability to be able to write software based on this diagram that can be implementable on a real world robot or robotic simulation directly.

V. From the Laboratory to Simulation

A. Laboratory Set-up and Experiment

The problem scenario that was investigated involved the case where a human can pick up a cup from a table and is able to keep adjusting the grasp as needed in order to avoid dropping the cup while the weight of the cup is being changed such as when pouring water into the cup. Our group was interested to see if we could replicate this scenario by using the Shadow Dexterous Hand with BioTac SP sensors attached. The Shadow Dexterous Hand is a five fingered manipulator that is the size of an average human male's hand and has 20 actuated degrees of freedom [10].

The BioTac SP sensors mimic the sense of touch and compliance of a human fingertip. The BioTac SP sensor has 24 electrodes, 2 pressure sensors (Pac and Pdc), and 1 temperature sensor built into the rigid core which is surrounded by fluid and wrapped in an elastic skin [11]. The BioTac SP sensors are the

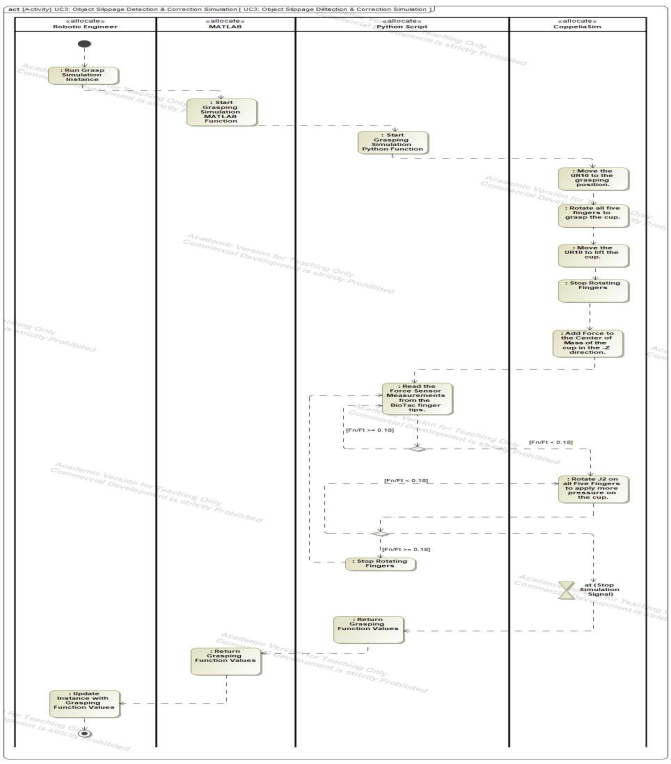

Figure 4: Activity Diagram for the robotic simulation.

green sensors attached to the Shadow Hand shown in Figure 6. When these sensors come into contact with an object, the electrode data provides the contact location through impedance sensing. The electrodes are mounted on a rigid core inside of the BioTac SP sensor.

The Pdc sensor uses changes in the fluid pressure inside the BioTac SP sensor to measure the overall pressure exerted by the fingertip [11]. The Pac sensor measures microvibrations detected when the skin moves across an object, and the temperature sensor measures the difference in temperature between the BioTac SP sensor and the object in contact [11]. The summary of the performance of these sensors is listed in Figure 5.

Sensory Modality	Symbol	Range	Resolution	Frequency Response
Impedance	E_n	0 - 3.3V	3.2 mV	0 - 100 Hz
Fluid Pressure	P_{DC}	0 - 100 kPa	36.5 Pa	0 - 1040 Hz
Microvibration	P_{AC}	+/-0.76 kPa	0.37 Pa	10 - 1040 Hz
Temperature	T_{DC}	0 - 75 C	0.1 C	0 – 22.6 Hz
Thermal Flux	T_{AC}	0 - 1 C/s	0.001 C/s	0.45 – 22.6 Hz

Figure 5: Summary of the Performance of the BioTac SP Sensors.

For each experiment, the Shadow Dexterous Hand was connected to a Universal Robots UR10 robotic arm, and the arm was moved into a designated starting position. Once the Shadow Hand was moved into the proper pre-grasping position, the fingers were moved to grasp the object which for all experiments was a 22 ounce plastic bottle. The movement of the Shadow Hand fingers was controlled by ROS (Robotic Operating System) messages that were sent by a computer in the laboratory that had the Shadow Hand software loaded into a Docker container. The weight of the bottle was measured before the start of experiments. Prior to lifting the bottle off the table, it was held securely by the BioTac SP sensors that are attached to the Shadow Hand fingers. The UR10 was moved to lift the bottle from the table, and the bottle was held at the same height at the beginning of each experiment. During the experiment, weight was added to the bottle at a consistent rate by pouring bb pellets through a funnel and into the bottle. The experiment was run until the bottle slipped completely from the hand. Sixty four experiments were conducted with each experiment collecting raw sensor data which was synced with visual data that was recorded by a high resolution camera. The experimental set up is shown in the following Figure 6.

(a) Experiment Configuration (b) Funnel and Bb pellets

Figure 6: Experimental Set Up for Data Collection. Bb pellets are used as the filler, and we use a funnel to ensure a constant pouring rate

B. Move to Simulation

A slippage detection and correction algorithm was created for the Shadow Hand to be able to autonomously detect and stop a soft or rigid cup from slipping from the hand while weight was being added. However, a simulation of this robotic task had not previously been created. It is necessary to be able to simulate robotic tasks so that problems can be resolved before using the robotic hardware. This will reduce the risk of failures caused by improper use of the equipment or accidental errors during use. Further, during times when using real hardware is difficult such as during a pandemic, research can still continue as long as the robotic task run in simulation can be replicated by hardware. This was the motivation behind the main contribution of this paper which used Model-Based Systems Engineering methods to build a simulation that replicated the robotic task of solving the slippage detection and correction problem.

C. Cameo Systems Modeler to CoppeliaSim

As listed under the S.1.2: System External Interface Requirements, it was necessary to connect Cameo Systems Modeler to a robotic simulation software so that the models built in

CSM that represent the structure and behavior of the system of interest can connect to simulation. This will provide the system engineer with the ability to test parameters or conduct trade off studies by running simulations from CSM with different inputs and recording the metrics of interest as the output. In order to connect CSM to CoppeliaSim, the ®ParaMagic plugin was used in Cameo Systems Modeler. ®ParaMagic is a third-party plugin application that can connect the simulation engine from CSM to run models or functions in programs such as MATLAB, Modelica, or Mathematica [12]. A MATLAB function was created that would receive inputs from CSM and would then connect to a Python script which would externally control the robotic simulation in CoppeliaSim. These steps were modeled by the activity diagram for the slippage correction and detection algorithm use case. The portion of the activity diagram that deals with these software connections is shown in Figure 7.

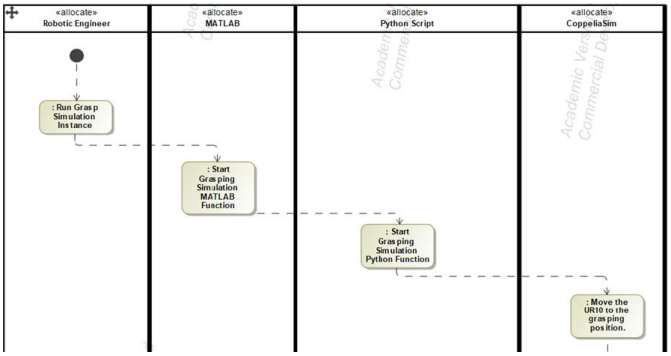

Figure 7: This is part of the activity diagram created for the slippage detection and correction use case which shows the connection from CSM to CoppeliaSim.

As shown in the first step of the workflow in Figure 7, the simulation process starts with running an instance BDD which was created in CSM. The instance BDD allows the systems engineer to choose the parameters that will be input into the simulation. An example of an instance used for this work is shown in Figure 8.

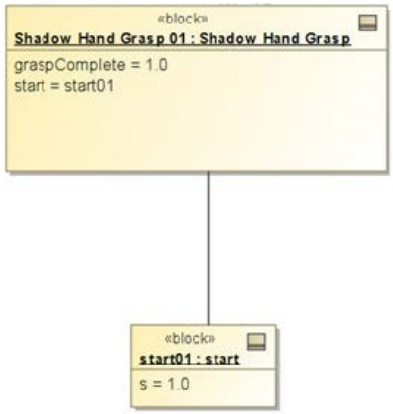

Figure 8: Example of an instance diagram built in CSM.

This is a simple instance created to input a parameter named start known as variable s to the connected MATLAB function. The connection to the MATLAB function was modeled by a parametric diagram. The MATLAB function was input into a constraint block. The input to the constraint block is the s parameter, and the output is the output of the MATLAB function. This output is then fed back into the graspComplete variable in the instance block. The parametric diagram is shown in Figure 9.

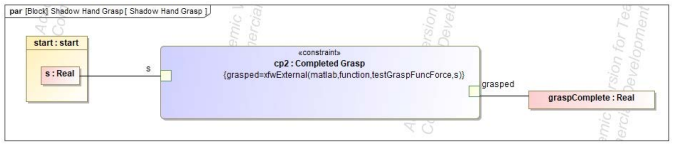

Figure 9: This parametric diagram shows the input, constraint block, and output used during the connection of CSM to MATLAB.

In order to start the simulation of the slippage detection and correction problem from CSM, an instance as shown in Figure 8 and a parametric diagram as shown in Figure 9 are created, and the following ®ParaMagic application window shown in Figure 10 is used to start the simulation.

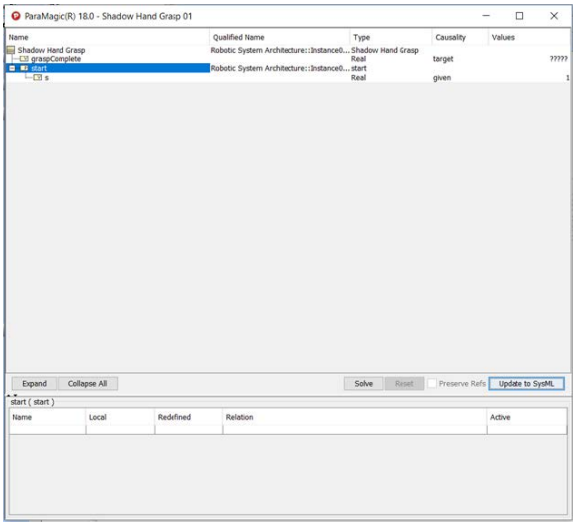

Figure 10: The ®ParaMagic application window used to initiate the robotic simulation from CSM.

The process is started by clicking the solve button. This will initiate the ®ParaMagic plugin which connects CSM and MATLAB. A MATLAB function was written by the author to accept the input of the variable s from CSM and trigger a Python script which starts the robotic simulation in CoppeliaSim. The functions used during this process were shown in the sequence diagram created in CSM. Once the robotic simulation has terminated, the output from this simulation is sent in the reverse direction back to CSM. This is what fills in the question marks shown next to graspComplete variable as seen in Figure 10. This allows the system engineer to

test different inputs by using various instances and record outputs of interest that are fed back to CSM at the end of the simulation.

VI. Verification and Validation

In order to satisfy the stakeholder requirements, it is first necessary to build a system that meets all of the stakeholder requirements. This is why it is important that the system or simulation requirements can trace back to stakeholder requirements. If every system requirement can trace back to a stakeholder requirement then the system should satisfy the stakeholder needs. One way to view this relationship is by creating a requirements trace matrix (RTM). A RTM provides the ability to trace down from the stakeholder requirements to see if at least one system requirement satisfies a stakeholder requirement. This ensures that all the stakeholder requirements are met. Further, a RTM provides the ability to trace upward to see if all system requirements can be traced back to at least one stakeholder requirement. This ensures there are no additional system requirements created that are not necessary which would indicate additional system elements built that are not needed.

The RTM created for the slippage detection and correction problem is shown in Appendix C in Figure 18. As seen in the RTM, the sum check column is used to check the upward trace from the system requirements to the stakeholder requirements. The sum check row is used to check the downward trace from the stakeholder requirements to the system requirements. A zero in the sum check row or column indicates an error that needs to be addressed.

One way to verify that the system of interest meets the system or simulation requirements is done by creating a requirements verification matrix (RVM). This is a matrix of data that identifies and records the verification methods used to verify each simulation requirement and the verification results [8]. The RVM created for the slippage detection and correction problem is shown in Appendix C in Figures 19, 20, and 21.

As shown in the RVM, the method for verifying each simulation requirement is detailed under the verification method column. The demonstration method is usually done by simulating the system and is used to show that the process works as intended [8]. The inspection method is a technique that is based on verification through the use of human senses or uses simple methods of measurement and handling [8]. The verification method description column describes the processes used to verify the simulation requirement in that row. If the verification method required a metric to be recorded or calculated, this metric is shown under the metric result column. A y in the verified column indicates that the simulation requirement was verified. All simulation requirements were verified.

Validation of the system or simulation involves providing objective evidence that the system or simulation when in use fulfills the stakeholder requirements [8]. The table in Appendix C in Figure 22 lists the stakeholder requirements that were created for the slippage detection and correction problem. These requirements apply to the experiment conducted in the lab and the replication of that experiment through simulation. As shown, all stakeholder requirements have been validated except for the accurate slip detection requirement which is explained in the comments column. Also, SHR.1.1.6 and SHR.1.1.7 were validated in simulation, but not yet tested for validation in the laboratory.

VII. Conclusion

In this paper, we followed the Model-Based System Engineering framework to solve the robotic task of detecting object slippage and correction while weight was being added. During this process, stakeholder requirements were created which allowed us to create measures of effectiveness for the robotic task. We were able to validate the robotic system and simulation against these requirements. Further, simulation requirements were derived from the stakeholder requirements. The system and simulation was verified against these requirements. During this work, we used the software engineering tool, Cameo Systems modeler, to create the necessary structural and behavioral diagrams for the robotic system and simulation. We then connected this software to a robust robotic simulation software, CoppeliaSim, and tracked the necessary measures of effectiveness from CSM to CoppeliaSim and back. This provided us with the ability to go from requirements creation to implementation all from one software tool. In future work, we are looking to connect our robotic simulation to the real hardware in order to fully establish a digital twin. Overall, we were able to follow MBSE techniques to solve the robotic task of slippage detection and correction within a five-fingered robotic hand from the laboratory to simulation.

References

[1] F. Ingrand, "Recent trends in formal validation and verification of autonomous robots software," *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pp. 321–328, 2019.

[2] D. Come, J. Brunel, and D. Doose, "Improving code quality in ros packages using a temporal extension of first-order logic," *ICRC*, 2018.

[3] W. Meng, J. Park, O. Sokolsky, S. Weirich, and I. Lee, "Verified ros-based deployment of platform-independent control systems," *NASA Formal Methods*, 2015.

[4] K. W. Wong and Kress-Gazit, "H. robot operating system (ros) introspective implementation of high-level task controllers," *JOSER*, 2017.

[5] J. Huang, C. Erdogan, Y. Zhang, B. Moore, and Q. Luo, "Rosrv: Runtime verification for robots," *Runtime Verification*, 2014.

[6] A. Sorin, L. Morten, J. Kjeld, and U. P. Schultz, "Rule-based dynamic safety monitoring for mobile robots," *Journal of Software Engineering in Robotics*, 2016.

[7] M. Schluse, L. Atorf, and J. Rossmann, "Experimentable digital twins for model-based systems engineering and simulation-based development," *2017 Annual IEEE International Systems Conference (SysCon)*, pp. 1–8, 2017.

[8] I. C. on Systems Engineering, *Systems Engineering Handbook*, 4th ed., D. D. Walden, G. J. Roedler, K. J. Forsberg, R. D. Hamelin, and T. M. Shortell, Eds. John Wiley and Sons, Inc., 2015.

[9] L. Delligatti, *SysML Distilled A Brief Guide to The Systems Modeling Language*. Pearson Education, Inc., 2014.

[10] *Shadow Dexterous Hand E Series Technical Specification*, Shadow Hand Company, February 2019.

[11] *BioTac SP Product Manual*, Syntouch, August 2018.

[12] ®*ParaMagic 18.0 User Guide*, InterCAX, 2014.

VIII. Appendix A

Table IV: Simulation Requirements Table Continued.

ID	Simulation Requirements	Simulation Requirement Description
S.1.1.1.11	Use Friction	The Robotic System Sim shall use friction between the grasp object and robotic fingertips to hold the object.
S.1.1.1.12	Calculate Friction Force	The Robotic System Sim shall be able to calculate the force of friction.
S.1.1.2	System Performance Requirements	
S.1.1.2.1	Slippage Detection	The Robotic System Sim shall detect object slippage within 2 secs.
S.1.1.2.2	Slippage Correction	The Robotic System Sim shall stop object slippage within 100ms.
S.1.1.2.3	Displacement of Grasp Object	The Robotic System Sim shall ensure total object displacement is < 0.05m.
S.1.1.2.4	Rotation of Grasp Object	The Robotic System Sim shall maintain the grasped object rotation about its center of mass relative to the world frame to be < 20 degrees.
S.1.1.2.5	Max Grasp Force	The Robotic System Sim shall apply less force than the crushing threshold of the grasped object.
S.1.2	System External Interface Requirements	
S.1.2.1	Start From CSM	The Robotic System Sim shall start from an instance in CSM.
S.1.2.2	Connect to Robotic Simulation Software	The Robotic System Sim shall connect CSM to a robotic simulator.
S.1.2.3	External Control	The Robotic System Sim shall be externally controlled by an external application.
S.1.2.4	Output Grasp Object Position	The Robotic System Sim shall output the grasp object position to external applications.
S.1.2.5	Output Grasp Object Orientation	The Robotic System Sim shall output the grasp object orientation to external applications.
S.1.2.6	Output Grasp Force	The Robotic System Sim shall output the grasp force to external applications.
S.1.2.7	Output Grasp Hold Performance Metric	The Robotic System Sim shall output a 1 if the grasped object was held during the correction stage of the simulation.
S.1.2.8	Output to CSM	The Robotic System Sim shall output metrics to CSM.

IX. Appendix B

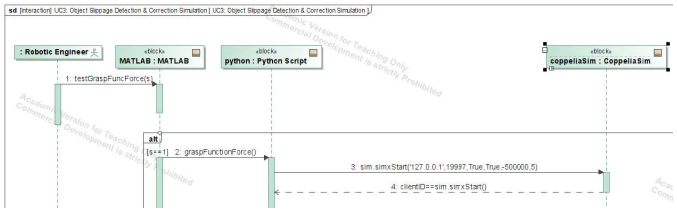

Figure 11: Section 1 of sequence diagram for the robotic simulation.

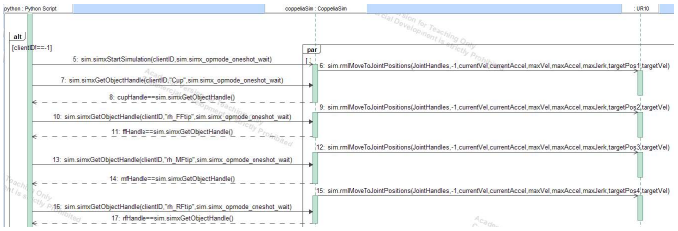

Figure 12: Section 2 of sequence diagram for the robotic simulation.

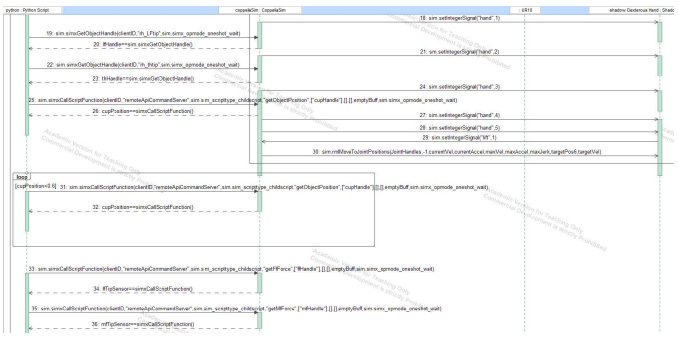

Figure 13: Section 3 of sequence diagram for the robotic simulation.

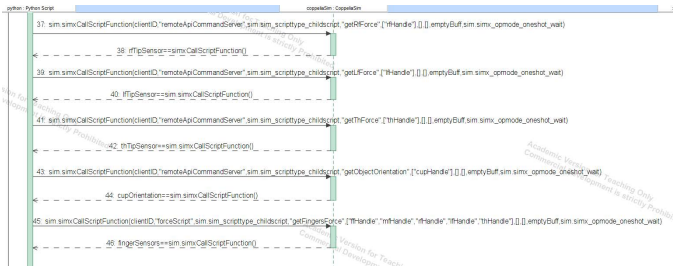

Figure 14: Section 4 of sequence diagram for the robotic simulation.

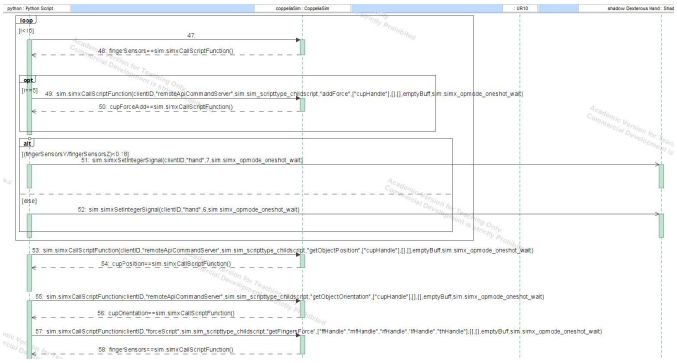

Figure 15: Section 5 of sequence diagram for the robotic simulation.

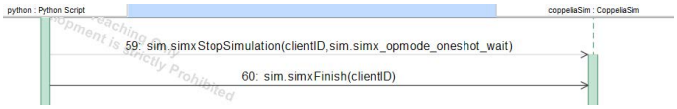

Figure 16: Section 6 of sequence diagram for the robotic simulation.

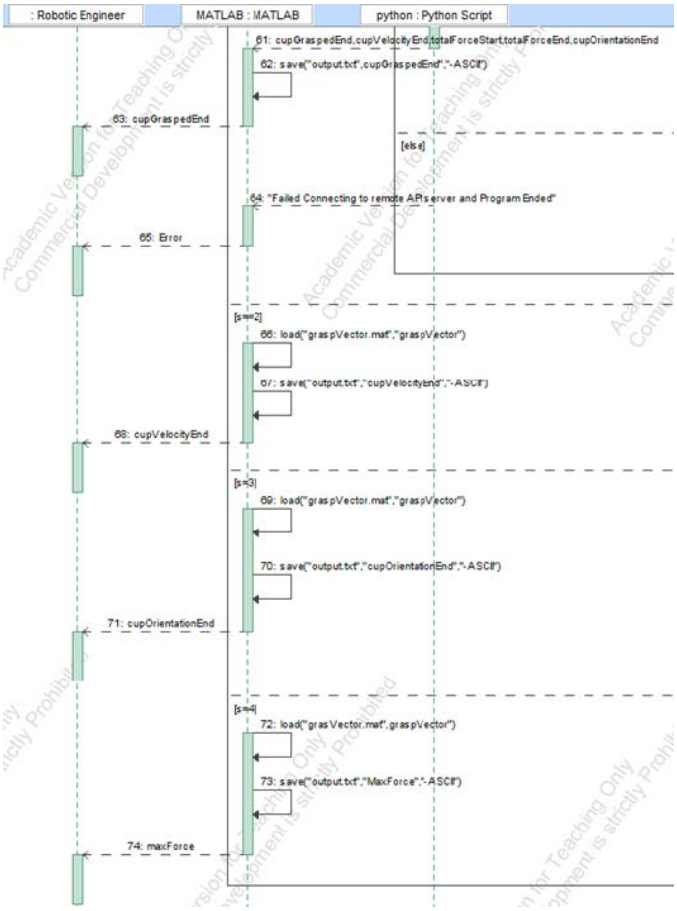

Figure 17: Section 7 of sequence diagram for the robotic simulation.

X. Appendix C

	Stakeholder Requirements												
System Requirements	SHR.1.1 Performance Requirements and Constraints	SHR.1.1.1 Detect Object Slippage	SHR.1.1.2 Use Anthropomorphic Robotic Hand	SHR.1.1.3 Autonomous Detection and Correction	SHR.1.1.4 Stop Object from Falling	SHR.1.1.5 Accurate Slippage Detection	SHR.1.1.6 Time of Slippage Detection	SHR.1.1.7 Time of Slippage Correction	SHR.1.1.8 Object Correction Displacement	SHR.1.1.9 Maintain Object Orientation	SHR.1.1.10 Max Force Applied	SHR.1.1.11 Replicate Laboratory Experiment in Simulation	Sum Check
S.1.1 System Capability Requirements													N/A
S.1.1.1 System Functional Requirements													N/A
S.1.1.1.1 Simulate Robotic Equipment			1									1	2
S.1.1.1.2 Simulate Grasp Objects												1	1
S.1.1.1.3 Simulate Laboratory Environment												1	1
S.1.1.1.4 Import Unified Robot Description Format Files			1									1	2
S.1.1.1.5 Calculate Robot Kinematics				1									1
S.1.1.1.6 Calculate Dynamics				1									1
S.1.1.1.7 Use Physical Engines				1									1
S.1.1.1.8 Robotic Arm Movement				1									1
S.1.1.1.9 Robotic Hand Movement			1	1	1								3
S.1.1.1.10 Add Weight Force		1											1
S.1.1.1.11 Use Friction					1			1	1	1			4
S.1.1.1.12 Calculate Friction Force					1			1	1	1			4
S.1.1.2 System Performance Requirements													N/A
S.1.1.2.1 Slippage Detection		1		1		1	1						4
S.1.1.2.2 Slippage Correction				1	1			1	1				4
S.1.1.2.3 Displacement of Grasp Object									1				1
S.1.1.2.4 Rotation of Grasp Object										1			1
S.1.1.2.5 Max Grasp Force											1		1
S.1.2 System External Interface Requirements													N/A
S.1.2.1 Start from CSM												1	1
S.1.2.2 Connect to Robotic Simulation Software												1	1
S.1.2.3 External Control				1									1
S.1.2.4 Cutput Grasp Object Position					1				1				2
S.1.2.5 Cutput Grasp Object Orientation										1			1
S.1.2.6 Cutput Grasp Force											1		1
S.1.2.7 Cutput Grasp Hold Performance Metric				1									1
S.1.2.8 Cutput to CSM												1	1
Sum Check	N/A	2	3	9	5	1	1	3	5	4	2	7	

Figure 18: Requirements trace matrix for the slippage detection and correction problem.

System Requirements	Verification Method	Verification Method Description	Verified?	Metric Result
S.1.1 System Capability Requirements	N/A	N/A	N/A	N/A
S.1.1.1 System Functional Requirements	N/A	N/A	N/A	N/A
S.1.1.1.1 Simulate Robotic Equipment	Demo	Run simulation of robotic equipment in CoppeliaSim.	Y	N/A
S.1.1.1.2 Simulate Grasp Objects	Demo	Simulate cup in CoppeliaSim.	Y	N/A
S.1.1.1.3 Simulate Laboratory Environment	Demo	Simulate laboratory environment in CoppeliaSim.	Y	N/A
S.1.1.1.4 Import Unified Robot Description Format Files	Demo	Import Shadow Robot Hand URDF into CoppeliaSim.	Y	N/A
S.1.1.1.5 Calculate Robot Kinematics	Inspection	Visualize the geometries of the UR10 and Shadow Hand Robot in CoppeliaSim.	Y	N/A
S.1.1.1.6 Calculate Dynamics	Inspection	Visually verify that CoppeliaSim can calculate the dynamics for all items that are being simulated.	Y	N/A
S.1.1.1.7 Use Physical Engines	Demo	Select a physical engine to use during simulation and test that gravity is being applied and objects in contact behave similar to in the real world.	Y	N/A
S.1.1.1.8 Robotic Arm Movement	Demo	Send commands to move UR10 to a specified location.	Y	N/A
S.1.1.1.9 Robotic Hand Movement	Demo	Send commands to rotate the Shadow Hand fingers by a specified rotational velocity.	Y	N/A
S.1.1.1.10 Add Weight Force	Demo	Use embedded CoppeliaSim function to add force to the center of mass of the grasp object along the negative Z axis.	Y	N/A
S.1.1.1.11 Use Friction	Test	Change friction coefficient parameters for the fingertips and cup and add the same force. Verify the changes visually when running the simulation.	Y	N/A
S.1.1.1.12 Calculate Friction Force	Demo	Use force sensors on the fingertips to collect force data in the normal and tangential directions and calculate friction force.	Y	N/A

Figure 19: First section of the requirements verification matrix for the slippage detection and correction problem.

System Requirements	Verification Method	Verification Method Description	Verified?	Metric Result
S.1.1.2 System Performance Requirements	N/A	N/A	N/A	N/A
S.1.1.2.1 Slippage Detection	Demo	Use force sensors on the fingertips to detect when there is a change in the ratio between the total force in the normal direction and the total force in the tangential direction changes below a measured threshold value. Measure slippage detection time.	Y	0.116 secs
S.1.1.2.2 Slippage Correction	Demo	Send a command to rotate Joint 2 for each finger and thumb at a certain rotational velocity to apply more pressure to the grasped object. Measure slippage correction time.	Y	0.117 secs
S.1.1.2.3 Displacement of Grasp Object	Demo	Record cup position before force is added to make it slip and record cup position at the end of the simulation when cup is still in hand. Measure difference between before and after.	Y	0.011 meters
S.1.1.2.4 Rotation of Grasp Object	Demo	Record cup orientation before force is added to make it slip and record cup orientation at the end of the simulation when cup is still in hand. Measure difference between before and after.	Y	Change in X: 0.217 degrees Change in Y: 6.92 degrees Change in Z: 0.098 degrees
S.1.1.2.5 Max Grasp Force	Demo	Track Grasp Force once cup is grasped by the Shadow Hand Robot	Y	Never Exceeded Grasp Crushing Threshold
S.1.2 System External Interface Requirements	N/A	N/A	N/A	N/A
S.1.2.1 Start from CSM	Demo	Start robotic simulation from CSM.	Y	N/A
S.1.2.2 Connect to Robotic Simulation Software	Demo	Connect CSM to CoppeliaSim.	Y	N/A
S.1.2.3 External Control	Demo	Control CoppeliaSim simulation by the script written in Python.	Y	N/A

Figure 20: Second section of the requirements verification matrix for the slippage detection and correction problem.

System Requirements	Verification Method	Verification Method Description	Verified?	Metric Result
S.1.2.4 Output Grasp Object Position	Demo	Output cup position to the Python function and then to the MATLAB function.	Y	N/A
S.1.2.5 Output Grasp Object Orientation	Demo	Output cup orientation to the Python function and then to the MATLAB function.	Y	N/A
S.1.2.6 Output Grasp Force	Demo	Output grasp force as measured by the force sensors attached to the fingertips to the Python function and then to the MATLAB function.	Y	N/A
S.1.2.7 Output Grasp Hold Performance Metric	Demo	Output grasp hold metric to the Python function and then to the MATLAB function.	Y	N/A
S.1.2.8 Output to CSM	Demo	Output requested metric back to CSM.	Y	N/A

Figure 21: Third section of the requirements verification matrix for the slippage detection and correction problem.

Stakeholder Requirements	Stakeholder Requirements Text	Validated ?	Comments
SHR.1.1 Performance Requirements and Constraints		N/A	None
SHR.1.1.1 Detect Object Slippage	The Robotic System shall use haptic sensors to detect object slippage.	Y	BioTac SP sensors used in lab. BioTac SP sensors goemetry used in simulation with force sensors attached.
SHR.1.1.2 Use Anthropomorphic Robotic Hand	The Robotic System shall use a five-fingered dexterous robotic hand.	Y	Shadow Robot Hand used in lab and simulation.
SHR.1.1.3 Autonomous Detection and Correction	The Robotic System shall detect and correct object slippage autonomously.	Y	Slippage detection and correction algorithm developed for both in lab and simulation.
SHR.1.1.4 Stop Object from Falling	The Robotic System shall stop a grasped object from falling.	Y	Object stopped in lab and in simulation.
SHR.1.1.5 Accurate Slippage Detection	The Robotic System shall detect object slippage with a probability greater than 0.97%.	N/A	Did not verify in lab, and always detected in simulation.
SHR.1.1.6 Time of Slippage Detection	The Robotic System shall detect object slippage within 5 seconds.	Y	Verified in simulation. Not verified in lab.
SHR.1.1.7 Time of Slippage Correction	The Robotic System shall correct object slippage within 200 milliseconds.	Y	Verified in simulation. Not verified in lab.
SHR.1.1.8 Object Correction Displacement	The Robotic System shall maintain an object displacement of less than 0.05 meters during the correction stage.	Y	Cup did not have a displacement greater than 0.05 meters in lab and simlation.
SHR.1.1.9 Maintain Object Orientation	The Robotic System shall not rotate more than 20 degrees about its center of mass during correction.	Y	Cup did not rotate more than 20 degrees in simulation or in the lab (for most tests).
SHR.1.1.10 Max Force Applied	The Robotic System shall apply a correction force less than the safety margin of the object.	Y	Grasp threshold used in lab and simulation.
SHR.1.1.11 Replicate Laboratory Experiment in Simulation	The Robotic System shall be able to replicate the slippage detection and correction problem in simulation.	Y	Simulation of the slippage detection and correction problem completed in CoppeliaSim

Figure 22: This table lists the stakeholder requirements for the slippage detection and correction problem and the validation of these requirements.