ABSTRACT

Title of Thesis: MODEL-BASED SYSTEMS ENGINEERING APPLIED TO THE DETECTION AND CORRECTION OF OBJECT SLIPPAGE WITHIN A DEXTEROUS ROBOTIC HAND FROM THE LABORATORY TO SIMULATION

Charles Anthony Meehan,
Master of Science, 2020

Thesis Directed By: Professor John S. Baras,
Institute for Systems Research

Now more than ever, it is important to have the ability to replicate robotic tasks in simulation and be able to validate the simulation against stakeholder requirements and verify the simulation against simulation requirements. In a previous study, a five-fingered robotic hand, the Shadow Dexterous Hand, with haptic BioTac SP sensors attached was used to detect the moment of slip of an object from the robotic hand while weight was continuously being added and stop the object from falling from the grasp while not overcorrecting. This work was accomplished by Dr. Zhenyu Lin, Dr. John S. Baras, and the author in the Autonomy Robotics Cognition Laboratory at the University of Maryland. This thesis will present the use of Model-Based System Engineering techniques to replicate the detection and correction of object slippage by a five-fingered robotic hand using force feedback control in simulation.

MODEL-BASED SYSTEMS ENGINEERING APPLIED TO THE DETECTION AND CORRECTION OF OBJECT SLIPPAGE WITHIN A DEXTEROUS ROBOTIC HAND FROM THE LABORATORY TO SIMULATION

by

Charles Anthony Meehan

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2020

Advisory Committee:
Professor John S. Baras, Chair/Advisor
Professor Yiannis Aloimonos
Dr. Cornelia Fermuller

Dedication

This thesis is dedicated to my parents, Charles and Maria Meehan, and my sister and brother-in-law, Lauren and Stuart Keefe.

Acknowledgments

I would like to thank my advisor, Dr. John S. Baras, for his continuous support throughout the Master of Science program. I am very grateful for the opportunity to conduct research in robotics that he made possible. His expertise and enthusiasm for the robotic research conducted in the Autonomy Robotics Cognition laboratory pushed myself and others to accomplish and solve robotic tasks and research problems. I look forward to continue to work under his advisement in the future.

I appreciate the support of my committee members, Dr. Yiannis Aloimonos and Dr. Cornelia Fermuller. Also, I am thankful for their students willingness to help in the Autonomy Robotics Cognition laboratory.

I am grateful for the help and guidance provided to me during this program from Dr. John MacCarthy. He was always made sure to be available if there was something that I needed help with especially during the transition from the Master of Engineering to the Master of Science program. Also, I would like to thank Dr. Jeffrey Herrmann and Ms. Emily Irwin for their help during the last semester to properly fill out and turn in all the necessary paperwork for the program.

I would like to acknowledge and thank the continuous support and help from Mrs. Kim Edwards. I appreciate the attention and care that she has always provided throughout this program.

Lastly, this thesis was based upon work supported by the Office of Naval Research award #N000141712622 to the University of Maryland College Park.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ARC	Autonomy Robotics Cognition
API	Application Programming Interface
BDD	Block Definition Diagram
BO	BlueZero
CSM	Cameo Systems Modeler
CAD	Computer-aided Design
INCOSE	International Council on Systems Engineering
MBSE	Model-Based Systems Engineering
MOEs	Measures of Effectiveness
ODE	Open Dynamics Engine
ROS	Robot Operating System
RTM	Requirements Trace Matrix
RVM	Requirements Verification Matrix
SHR	Stakeholder Requirements
SysML	Systems Modeling Language
UR	Universal Robots

Chapter 1: Overview

1.1 Introduction

Over the years, one major research interest in the field of robotics has been the creation of anthropomorphic robots and the ability for these robots to replicate the senses and actions of humans. With regard to robotic grasping, it is desirable to have a robotic hand that has the same dexterity and control of a human hand. There has been various end effectors or grippers that have been created in order to grasp objects from hands with two to five fingers to end effectors that use suction cups. However, the optimal design for manipulability of objects is a five fingered robotic hand. The Shadow Dexterous Hand which is used in this thesis is an example of a anthropomorphic design of a robotic hand. Although this robotic hand has similar kinematics and dexterity to a human hand, it lacks the level of sensing that a human hand has through the sense of touch. Therefore, BioTac SP sensors were added to the Shadow Dexterous Hand so that experiments could be performed that would mimic human capabilities.

In the Autonomy Robotics Cognition laboratory at the University of Maryland, the Shadow Dexterous Hand with BioTac SP sensors attached was used to detect when an object slips from the hand based on statistical correlation of tactile

sensor data. The Shadow Hand also corrected for that motion by increasing the grasping force on the object but did not overcorrect and deform the held object. The objects being grasped were a rigid plastic cup, a soft plastic cup, and a paper cup. The weight of the cup was changed during each experiment. The goal of the experiment was to replicate the human ability to hold a cup and be able to maintain a proper grasp of the cup even if the weight of the cup changes over time such as when holding a cup being filled with water.

The focus of this work is to extend the previous work using the Shadow Dexterous Hand and BioTac SP sensors by applying Model-Based Systems Engineering methods to model the problem of replicating the laboratory work in simulation. In today's working and research environment, the need for robotic simulation of real world tasks is necessary and will help reduce the costs of errors in real world application. Therefore, this thesis will apply systems engineering practices in order to build a robotic simulation that uses force feedback control in order to detect and correct for a cup slipping in the grasp of the Shadow Dexterous Hand with BioTac SP sensors attached.

1.2 Relevant Work

Over the years, there has been research that focused on using haptic data to detect the moment an object starts slipping from a robotic end effector. The need of haptic sensors that mimic the sensory capabilities of the human hand has been well documented by N. Wettles, Jeremy A. Fishel and Gerald E. Loeb in [1] with their

discussion of their creation of the first BioTac sensor. Since that creation, there has been many experiments employing some version of the BioTac sensors in order to detect and correct for object slippage.

In [2], researchers use the Pac (vibrational pressure) readings from BioTac sensors attached to a three finger manipulator to detect micro-vibrations in order to detect slippage. If 11 out of the 22 pressure samples in a time window were above a certain threshold then a slippage was detected. Other papers that have used BioTac sensors usually use a neural network to decipher the BioTac sensor readings in order to build their detection and correction algorithms. In [3], M. Abd et al. constructed their slippage detection algorithm using data from BioTac SP sensors and the application of an artificial neural network classifier. However, their approach focused more on classifying the direction of an object slipping from a grasp than just detecting slippage by itself.

The approach of N. Wettels et al. [4] focused on using the BioTac electrode readings to estimate the tangential and normal forces applied by the Otto Bock M2 hand to a styrofoam cup. The cup was filled with water during their grasp control experiments in order to see if the robotic fingers could correct for slippage caused by the continuous increased weight of the flowing water which is a similar idea to our experiment which adds bb-pellets at a constant rate. Using their grasp control algorithm, it was possible to correct for the slippage of the soft cup, but as noted by the authors, their robotic hand had issues with correcting for slippage caused by a rapid fill rate. For our laboratory experiment, our robotic hand is able to correct for the slippage of rigid and soft cups regardless of the rate of fill.

In [5], the authors created a fuzzy sliding mode controller to detect and stop slippage. The robustness of this controller was evaluated in simulation and experimentation. The simulation was run in the Simulink environment where their gripper was simulated picking up an object. The Lugre friction model was used which is a continuous friction model that is able to describe the effects of stick and slip [5]. Their control scheme was tested by simulating the gripper catching various items with different masses, stiffness, and friction coefficients. Their fuzzy sliding mode controller outperformed a linear feedback controller and was found to be robust in simulation. However, their robotic gripper used for the simulation was based on two fingers acting as pinchers and not five-fingered grasp.

The work presented here draws inspiration from previous work in robotic grasping using haptic feedback and furthers the previous work completed in the ARC laboratory at the University of Maryland.

1.3 Problem Statement

The main objective of this work is to create a robotic simulation of the cup slipping experiment that was completed in the laboratory. The robotic simulation will simulate the same robotic equipment used in the prior experiment which includes the Universal Robots UR10, the Shadow Dexterous Hand, and the BioTac SP sensors. The simulation software will calculate the kinematics and dynamics for the robotic equipment and objects used during the simulation. The robotic simulation will connect to external software programs in order to receive external inputs

and output data to the external software programs. The simulation will be controlled by scripts internal to the simulation software and scripts written in external programming languages.

Stakeholder requirements for this problem will be developed. Measures of effectiveness will be derived from these requirements. Simulation requirements will be created in order to ensure that the robotic simulation created satisfies the needs of the stakeholder which will be demonstrated in a traceability matrix. Structural models of the equipment will be modeled by Cameo Systems Modeler (CSM) which is a Systems Engineering software tool. The behavior of the simulation will also be modeled in diagrams built in Cameo Systems Modeler. Components listed in the structural diagrams will be allocated to the simulation requirements. Instances of the simulation system block will be used in order to send inputs to the external robotic simulation and receive outputs from the simulation after completion.

This simulation will be initiated from Cameo Systems Modeler. The robotic simulation will run in a well known and popular robotic simulator so that the simulation can be used by other researchers. The simulation will simulate the UR10 moving the Shadow Hand to a cup placed at a known location. The Shadow Hand will then grasp the cup, and the UR10 will lift the cup from the table. Force will then be added to the center of mass of that cup in the negative z direction in order to simulate weight being added. Once the cup starts slipping from the grasp, the hand will automatically sense the slippage and correct by closing the fingers until the slipping stops. After these steps have completed, the simulation will output metrics of interest to Cameo Systems Modeler. Lastly, the robotic simulation will

be validated against the stakeholder requirements and verified against the simulation requirements. The simulation will also be used to select an optimal value for the rotational velocity parameter of the finger and thumb joints used during the correction stage of simulation.

1.4 Contribution of Thesis

The main contribution of this thesis is the development of a robotic simulation architecture written in SysML and displayed by a system's engineering software tool, Cameo Systems Modeler. The ability to connect Cameo Systems Modeler to a robotic simulation software such as CoppeliaSim is a novel contribution. The scripts written in Lua and Python in order to control the simulation were developed by the author as well as the MATLAB function used in the process of the connection between Cameo Systems Modeler and the robotic simulation software. The choice of using a popular robotic simulator to simulate this problem allows other users to further this simulation or use it for their own work dealing with a five-fingered robotic hand grasping in simulation.

The robotic simulation created was validated to meet the stakeholder needs of a robotic simulation that replicated the solution to the slippage detection and correction problem as seen in the laboratory. Another contribution is the simulation of stable grasping of a cup by the Shadow Dexterous Hand. This is further improved by the use of force feedback control in order to stop the cup from slipping from the Shadow Hand's grasp in simulation. The ability to use the simulation to optimize

the rotational velocity parameter of the finger and thumb joints was realized. With the simulation, it was possible to run 180 tests in under 3 hours which would not be capable in the lab.

This work presented the application of Model-Based Systems engineering methods to robotic simulation of the slippage detection and correction by a five-fingered robotic hand. Overall, the problem as seen in the laboratory was replicated in simulation which allows for future work and problems dealing with the use of five fingered grasping and manipulation. This will provide the ability to test algorithms before applying them in the real world which will prevent damage of expensive equipment.

1.5 Outline of Thesis

This section provides an outline of the thesis. Chapter 1 provides an overview of the slippage detection and correction problem and why the simulation of this problem was necessary. Chapter 2 reviews the previous laboratory slippage detection and correction experiment completed by the author along with Dr. Zhenyu Lin and Dr. John Baras. Chapter 3 explains the approach of Model-Based Systems Engineering to the creation of a robotic simulation of the slippage detection and correction problem. Chapter 4 details the model development in Cameo Systems Modeler by creating structural and behavior diagrams. Chapter 5 discusses the robotic simulation software, CoppeliaSim, and how the dynamics are calculated for the simulation. Chapter 6 presents the simulation of the robotic task and use of the robotic simulation software to simulate the slippage detection and correction

problem. Chapter 7 goes through the simulation validation and verification as well as using the simulation to optimize the rotational velocity parameter of the finger and thumb joints used during the correction stage. Chapter 8 concludes this work and describes future work.

Chapter 2: Tactile Based Slippage Detection and Correction Problem

This chapter will review the laboratory work involving the Universal Robots UR10 and the Shadow Dexterous Hand with BioTac SP sensors attached which were used to create a statistics based slippage detection and correction algorithm based on object classification. This work was conducted by the author, Dr. Zhenyu Lin, and Dr. John Baras in the Autonomy Robotics Cognition (ARC) laboratory at the University of Maryland.

2.1 Experimental Set Up and Problem Description

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 [6]. The dimensions of the Shadow hand are shown in Figure 2.1.

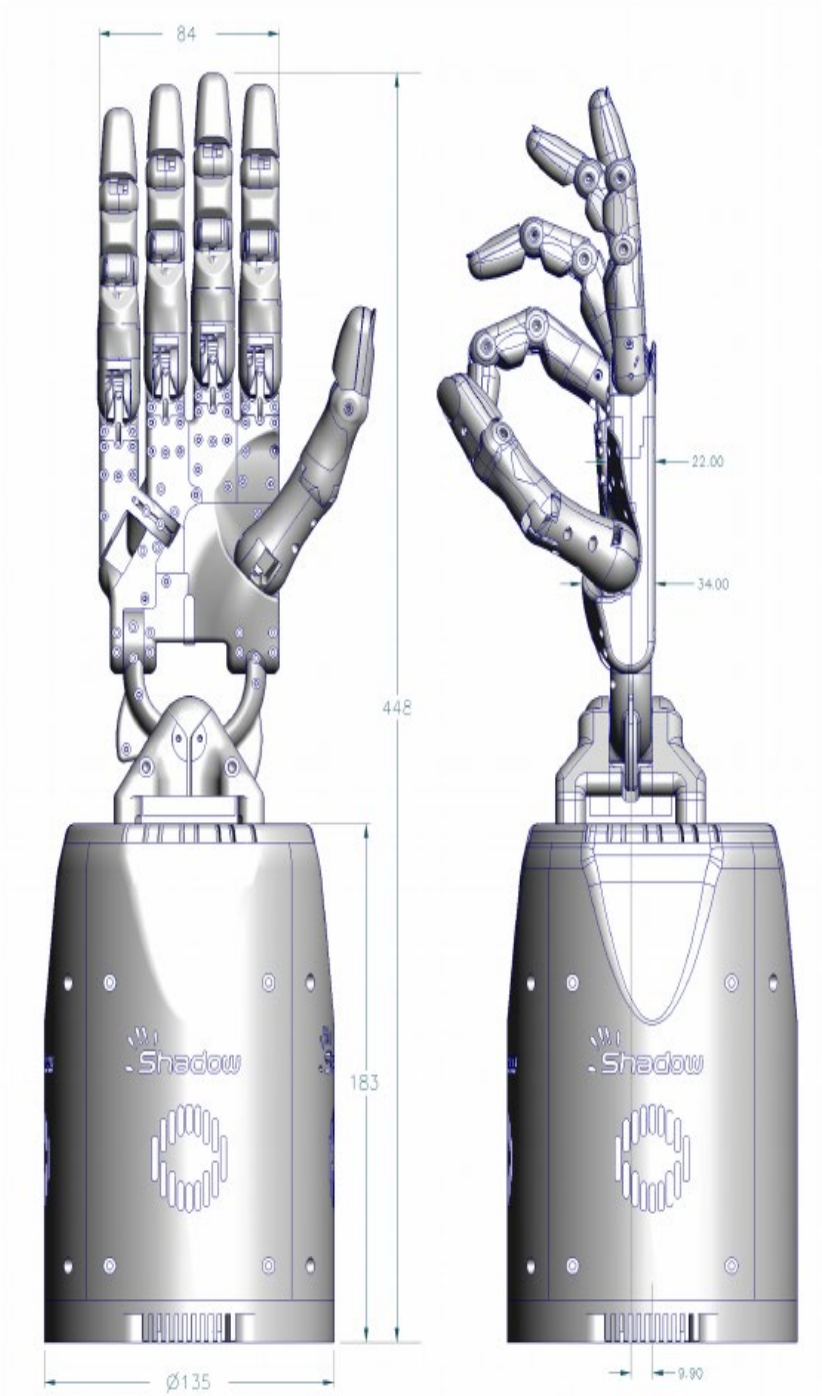

Figure 2.1: The Dimensions of the Shadow Dexterous Hand [6].

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 [7]. When these sensors come into contact with an object, the electrode data provides the contact location through impedance sensing. The location of the electrodes are shown in Figure 2.2. The electrodes are mounted on a rigid core inside of the BioTac SP sensor.

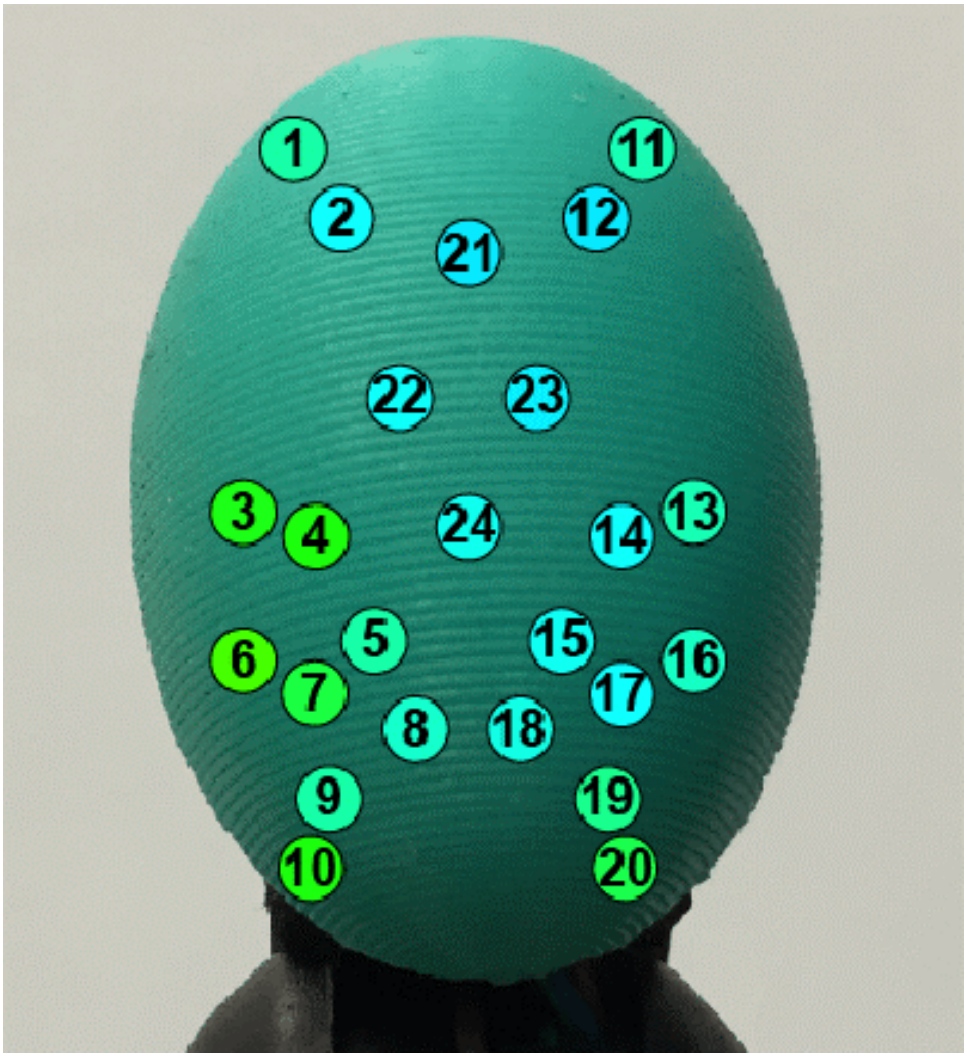

Figure 2.2: Locations of the electrodes in the BioTac SP sensor.

The Pdc sensor uses changes in the fluid pressure inside the BioTac SP sensor to measure the overall pressure exerted by the fingertip [7]. 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 [7]. The summary of the performance of these sensors is listed in Figure 2.3.

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 2.3: Summary of the Performance of the BioTac SP Sensors [7].

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 synched with visual data that was recorded by a high resolution camera. The experimental set up is shown in the following Figure 2.4.

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

Figure 2.4: Experimental Set Up for Data Collection.

2.2 Statistics Based Slippage Detection

The main objective of collecting 64 sets of haptic data was to find the moment of correlation in the electrode data which would indicate the moment the cup starting slipping from the hand. In order to detect slippage, it is important to understand the statistics of the sensor data before, during and after the moment of slip. Dr. Zhenyu Lin took the sensor data from the 64 experiments in order to find the moment of slippage. He first used a median flow tracking algorithm [8] to determine

the time that the cup started to slip for each experiment from the visual data. The median flow tracker tracked the cup in forward and backward directions in time and measured the discrepancies between these two trajectories. Minimizing this Forward-Backward error enabled the tracker to reliably track the cup. We used the tracker to track the cup grasped in the hand and detect if slippage occurs based on the velocity of the cup.

As shown in figure 2.5, the blue bounding box indicates that we are tracking the bottom of the cup. The text on the top left corner indicates the status of the experiment, whether slippage is detected or not, and the first frame that slippage is detected.

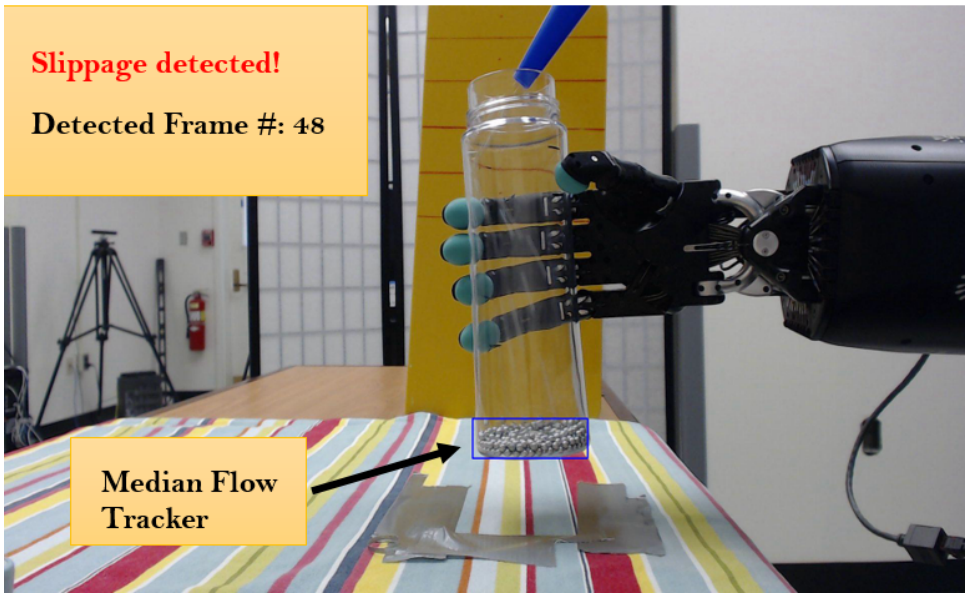

Figure 2.5: Median Flow Tracker is used to determine t_j^*. In the experiment pictured, first slippage is detected at frame 48 and t_j^*=1.6s.

Slippage occurred if the velocity of the object was greater than some small threshold ϵ. t_j^* denoted the first moment that slippage occurred for the jth experi-

ment. Once slippage time t_j^* was found for each experiment $j \in \{1 \cdots M\}$, where M was the total number of experiments in the dataset, the data was aligned based on t_j^* since the statistics around the moment of slip was key. The electrode data was denoted as $x_{i,k}^j(t)$, where j was the index for the experiment number, $i \in \{1 \cdots n_f\}$, was the index of the finger , $k \in \{1 \cdots n_e\}$ was the electrode index, and t was the time sample index. n_f was the number of fingers which is equal to 5 for the Shadow Hand, n_e was the number of electrodes which is equal to 24 for the BioTac SP sensors attached to the Shadow Hand. Finger index 1 to 5 corresponded to first finger (FF), middle finger (MF), ring finger (RF), little finger (LF) and thumb (TH), respectively.

In order to detect the moment of slippage from the sensor data, a statistical based method was used to find the correlation between the sensor data around the moment of slippage. This method calculated the correlation between the realtime haptic data sequence X_R and the pre-slip haptic data sequence X_H. The dimension of X_H is $X_H \in I \times n_f \times n_e$, where I is the window size $I = (t_b + t_a + 1)$. A single pre-slip haptic data sequence X_S for the jth experiment can be defined as follows:

$$X_S(t, j, i, k) = x_{i,k}^j(t) \tag{2.1}$$

for $t \in [t_j^* - t_a, t_j^* + t_b]$, $i \in [1, n_f]$, $k \in [1, n_e]$. Then, the average pre-slip haptic data sequence of the dataset was calculated as follows:

$$X_H(t, i, k) = \frac{1}{N_e} \sum_{j=1}^{N_e} X_S(t, j, i, k) = \frac{1}{N_e} \sum_{j=1}^{N_e} x_{i,k}^j(t) \tag{2.2}$$

for $t \in [t_j^* - t_a, t_j^* + t_b]$, $i \in [1, n_f]$, $k \in [1, n_e]$, $N_e = 64$ was the total number of experiments in the dataset.

The realtime haptic data sequence X_R has the same dimension as X_H, and it was updated with every new sample. Let t_{now} denote the current time index, then

$$X_R(t, i, k) = x_{i,k}^j(t) \tag{2.3}$$

for $t \in [t_{now} - (t_a + t_b), t_{now}]$, $i \in [1, n_f]$, $k \in [1, n_e]$.

The correlation between the realtime sequence X_R and the pre-slip sequence X_H for electrode k on finger i at time t was calculated as follows.

$$\rho_{i,k}(X_R, X_H) = \frac{1}{N-1} \sum_{j=1}^{N} (\frac{X_{R_j} - \mu_{X_R}}{\sigma_{X_R}})(\frac{X_{H_j} - \mu_{X_H}}{\sigma_{X_H}}) \tag{2.4}$$

where μ and σ are the mean and standard deviation of the haptic data sequence for electrode k on finger i. $N = (t_a + t_b + 1)$ is the window size or number of observations in the sequence. We select $t_a = t_b = 75$ (which is equivalently 0.75 seconds) in order to understand the statistics around the moment of slippage. $(N - 1)$ here is the Bessel's correction, which uses $(N - 1)$ instead of N in the formula for the sample variance and sample standard deviation. This method corrects the bias in the estimation of the population variance. The total correlation is a sum of all the electrode correlations.

$$\rho(X_R, X_H) = \sum_{i=1}^{5} \sum_{k=1}^{24} \rho_{i,k}(X_R, X_H) \tag{2.5}$$

During the testing phase, the cross-correlation between the real-time haptic data and the haptic data that we collected for the interval around the moment of slippage was calculated. From the experiments, it was noticed that at the moment of slippage t^*, there was always a peak outlier appearing in the cross-correlation sequence. Therefore, our slippage prediction problem was transferred into a peak outlier detection problem. A peak in the cross-correlation suggested a possible slip, and the forces applied on the object by the fingers should be increased to prevent slippage.

2.3 Object Classification and Slippage Correction

After the slippage detection algorithm was working, the next piece was to create a correction algorithm that stopped the cup from slipping from the Shadow Hand but not deform the cup. This was a necessary piece so that the robotic hand could apply this correction when handling soft plastics or even paper cups. In order to know the force required to stop the cup from slipping from the hand, an estimation of the changing weight was used. To estimate the weight of the grasped cup, the vibration sensor data, Pac, from the BioTac SP sensors was used. The vibration sensor readings were significant only when the bb pellets started to enter the cup. As shown in Figure 2.4(b), the narrow funnel ensures that the pouring speed is almost always constant. Therefore, the weight increase from time t_1 to t_2 could be estimated as:

$$\Delta W_{t_1,t_2} = I_{P_{ac}(t)} \cdot \sum_{t=t_1}^{t=t_2} R \tag{2.6}$$

where I is an indicator function and takes a value 1 when the vibration sensor has a significant reading and 0 otherwise. R is a constant which decides the rate of the weight increase and depends on the size of the funnel.

Additionally, the container was classified into two different classes, i.e. rigid containers or soft containers. As mentioned previously, it is important to classify the container so that the applied grasping force will not destroy the container if the container is soft or fragile. As shown in Figures 2.6 and 2.7, the vibration sensor readings behave differently for soft containers as compared to rigid containers.

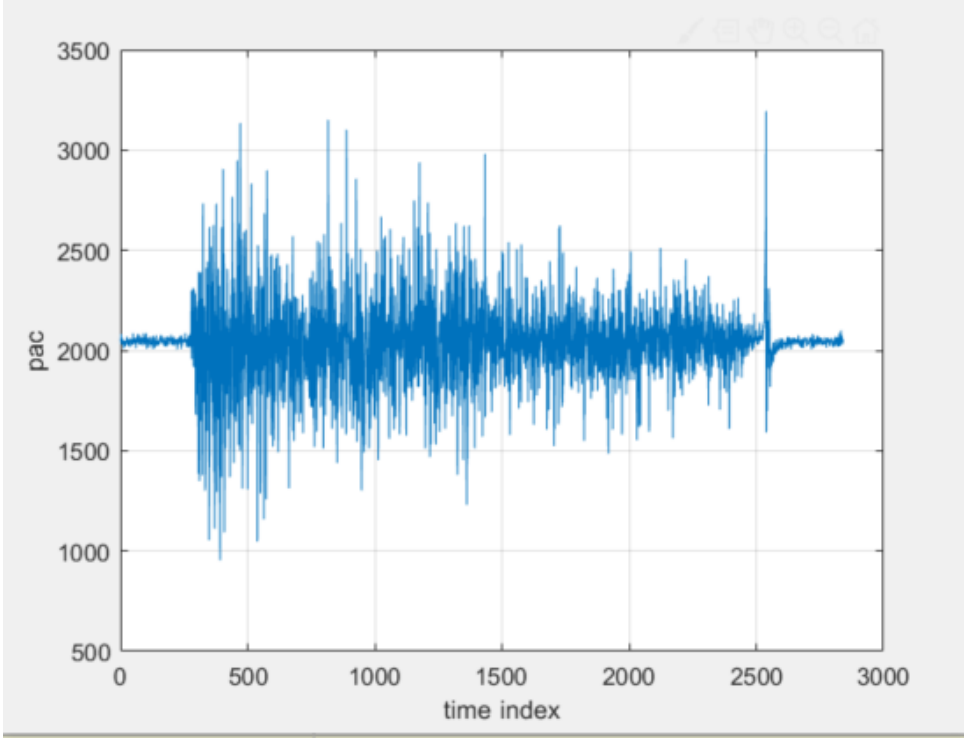

Figure 2.6: Pac for a rigid container.

Once the container was detected as a soft container, a grasping force threshold

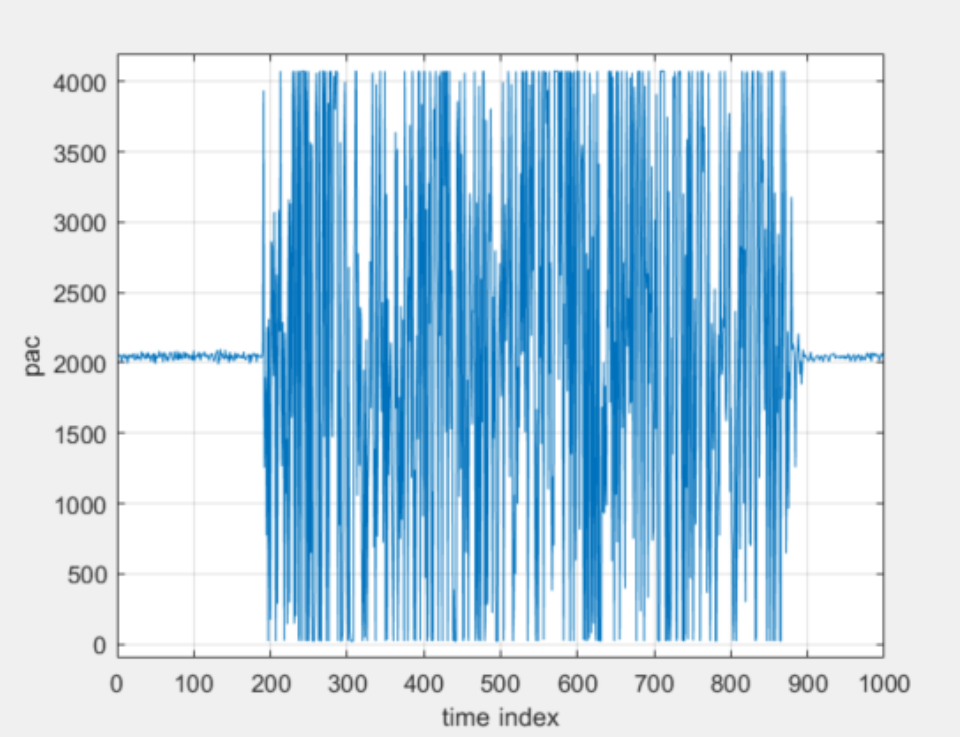

Figure 2.7: Pac for a soft container.

was added to the correction algorithm so that the Shadow Hand can not apply more force than the grasping force threshold during correction. There was a warning message displayed once this threshold was met, and the correction algorithm was stopped. This threshold can be adjusted based on user preferences such as deforming the cup past the crushing threshold of the cup in order to stop the cup from slipping from the hand.

2.4 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 was never 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 thesis which used Model-Based Systems Engineering methods to build a simulation that replicated the robotic task of solving the slippage detection and correction problem as detailed in this Chapter.

Chapter 3: Model-Based Systems Engineering Approach to Simulation

This chapter reviews the reasons to use Model-Based Systems Engineering (MBSE) and how those methods and practices are used to build a robotic simulation that can be verified and validated.

3.1 Use of Model-Based Systems Engineering

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" [9]. 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 communica-

tions among the system development team" [9]. 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.

When using Systems Engineering techniques to develop a system, these techniques can be visualized by the Vee model, shown in Figure 3.1.

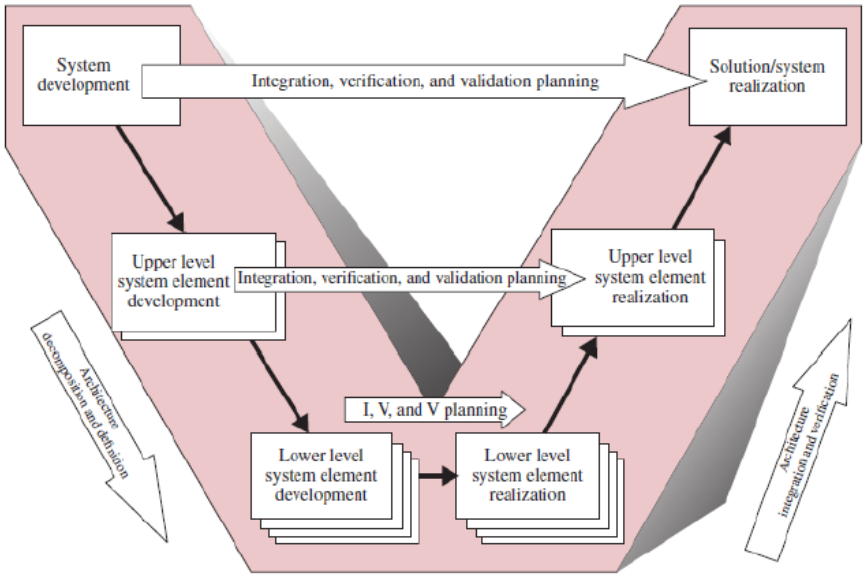

Figure 3.1: Vee Model Diagram as developed by Frosberg et al. (2005) [9].

The left hand side of the Vee model deals with the design and development of the system of interest while the right hand side deals with testing and realization of the system of interest.

For building the robotic simulation, the Vee model was followed. Therefore, 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 Effective-

ness (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 behaviorial 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 [10]. 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 process of following the Vee model, the simulation was validated against the stakeholder requirements and verified against the simulation requirements.

3.2 Stakeholder Requirements

Following the Vee model as shown in 3.1, stakeholder needs or requirements for the robotic system that solves the slippage detection and correction problem were developed. As defined in the INCOSE Systems Engineering Handbook, "the purpose of the Stakeholder Needs and Requirements Definition process is to define the stakeholder requirements for a system that can provide the capabilities needed by the users and other stakeholders in a defined environment" [9]. 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 3.1 of stakeholder requirements.

ID	Stakeholder Requirement	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 3.1: Stakeholder Requirements Table.

3.3 Measures of Effectiveness

The measures of effectiveness for the robotic system that will solve 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" [9]. The MOEs are listed in Table 3.2.

3.4 Simulation Requirements

The main focus of this work is to create a robotic simulation of the slippage correction and detection problem that was worked on in the lab. Therefore, there is a need for simulation requirements so that the elements built for the simulation can be verified. This way items that needs to be included in the simulation are included and elements are not built that are not needed. These requirements are derived from the stakeholder needs and requirements. The simulation requirements are listed in Tables 3.3, 3.4, and 3.5.

Metric	Definition	Threshold Value	Objective Value
Probability of Detection of Object Slip (PDS)	The probability of detecting object slippage from the robotic 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 robotic hand given the object does not slip.	$< 3\%$	$< 1\%$
Time of Object Slip Detection (TOS)	The time is takes to detect object slippage from the robotic hand.	$<$ 5secs	$<$ 2secs
Time of Object Slip Correction (TOC)	The time is 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 Grasped Object (MGF)	The maximum force applied by the robotic hand to the 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

Table 3.2: Measures of Effectiveness Table.

ID	Simulation Requirement	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 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 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.

Table 3.3: Simulation Requirements Table.

ID	Simulation Requirement	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 grasp 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.

Table 3.4: Simulation Requirements Table Continued.

ID	Simulation Requirement	Simulation Requirement Description
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.

Table 3.5: Simulation Requirements Table Continued.

Chapter 4: System Description

The focus of this chapter is on the creation of the system architecture which was modeled by structural and behavioral diagrams. While the system structural diagrams model the robotic system that is used for the slippage detection and correction problem, the behavior diagrams modeled the behavior of the laboratory experiments and simulation.

4.1 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 [10]. The use case diagram built for this problem is presented in Figure 4.1.

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 and correction problem. The only actor involved in

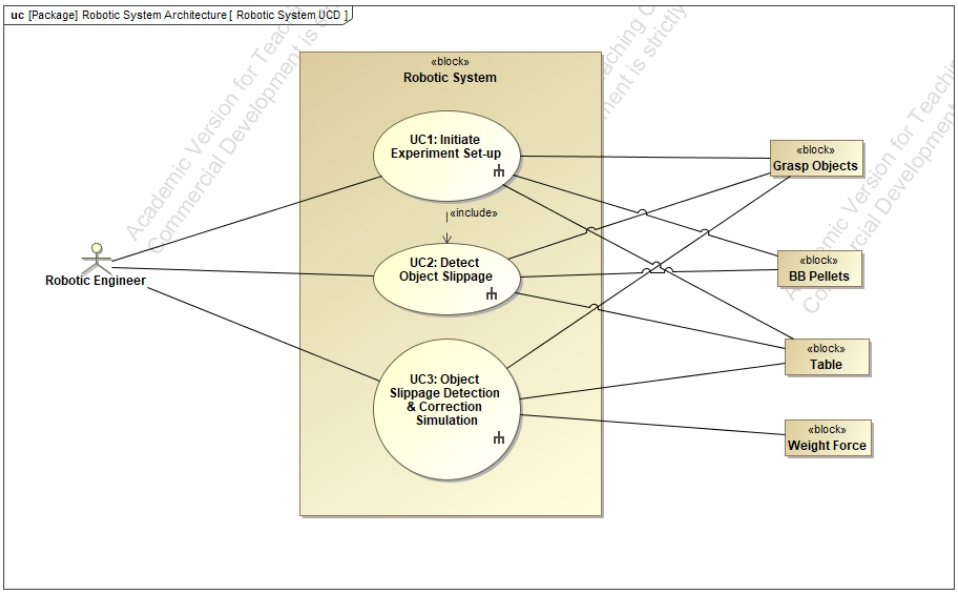

Figure 4.1: Use Case Diagram developed for the slippage detection and correction problem.

these three use cases was the robotic engineer who is responsible for running the experiment and 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.

4.2 The Robotic System Context and System Level Block Definition Diagrams

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 [10]. The context-level BDD for the robotic system used in the slippage detection and correction problem is shown in Figure 4.2.

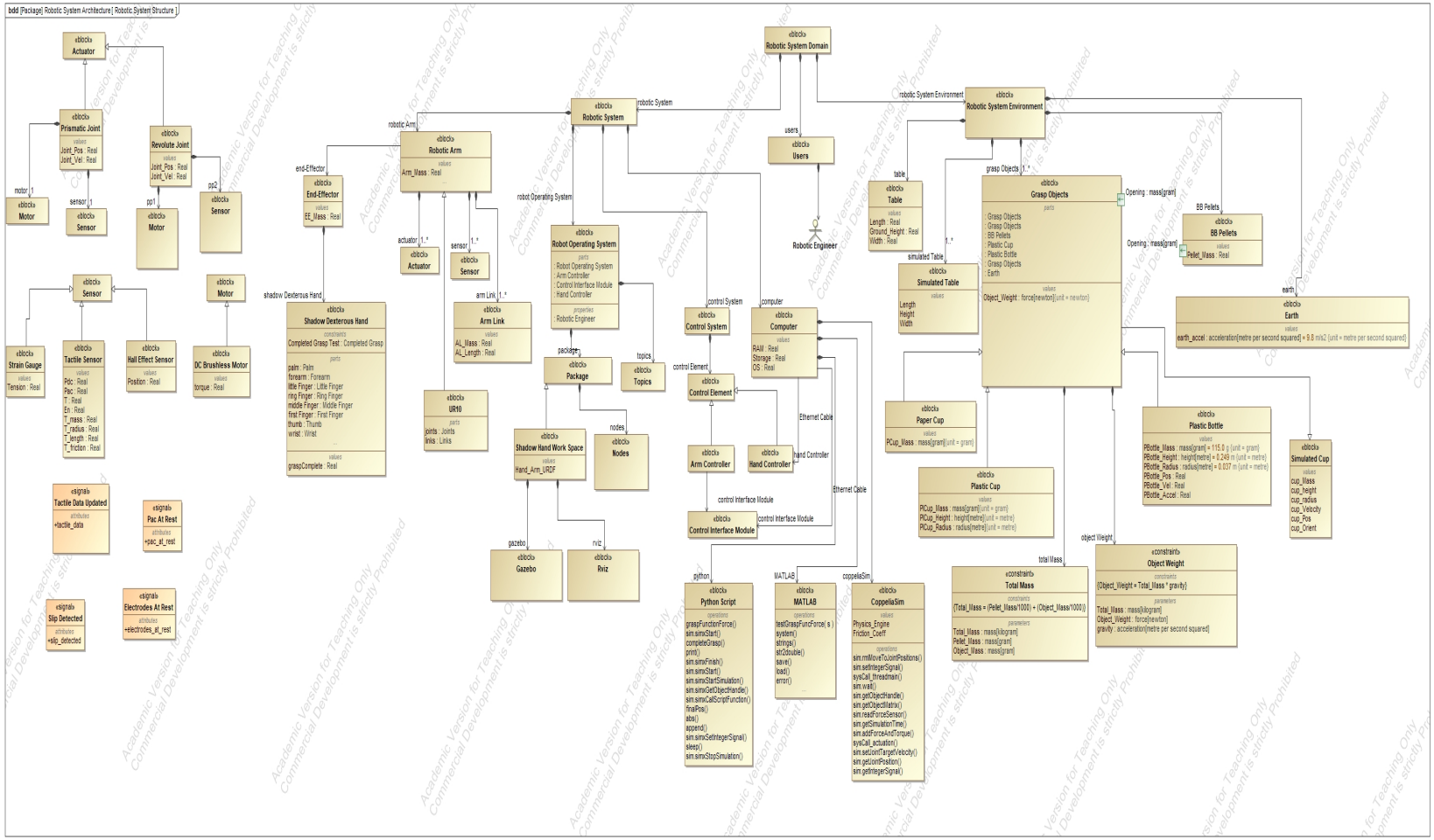

Figure 4.2: Context-Level BDD of Robotic System.

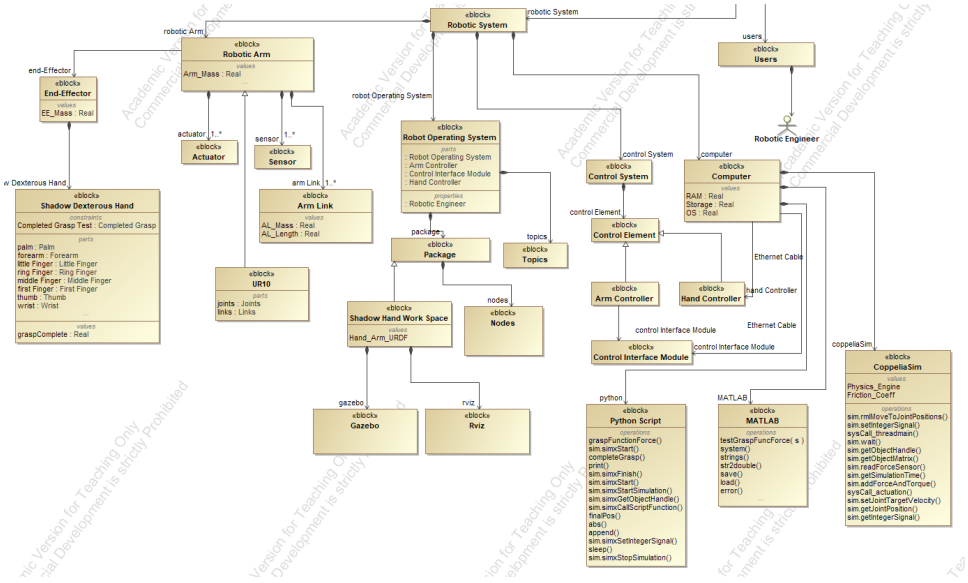

Figure 4.3: Left side view of the context-level BDD.

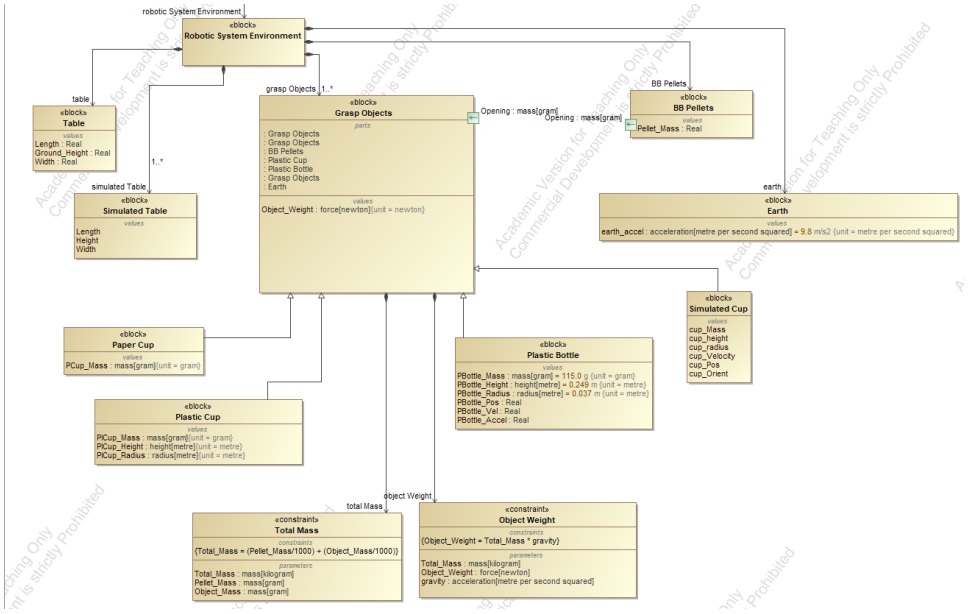

Figure 4.4: Right side view of the context-level BDD.

In Figure 4.3, 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. In Figure 4.4, 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. The BDD for the Shadow Hand is shown in Figure 4.5, and the BDD for the UR10 is shown in Figure 4.6. 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 for the UR10 displays the joints and links of the robot and this structural decomposition 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.

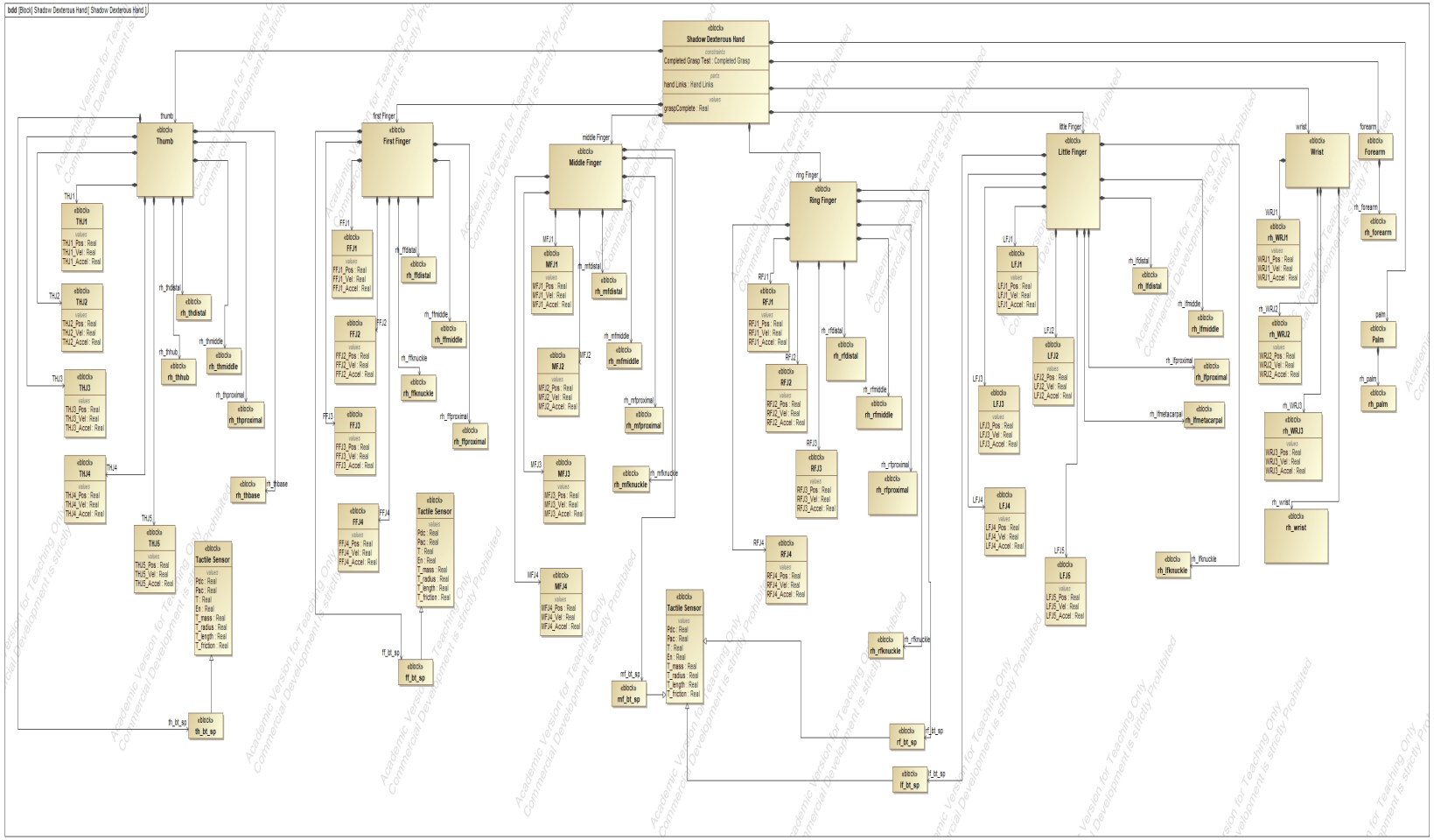

Figure 4.5: System-Level BDD of the Shadow Hand Robot.

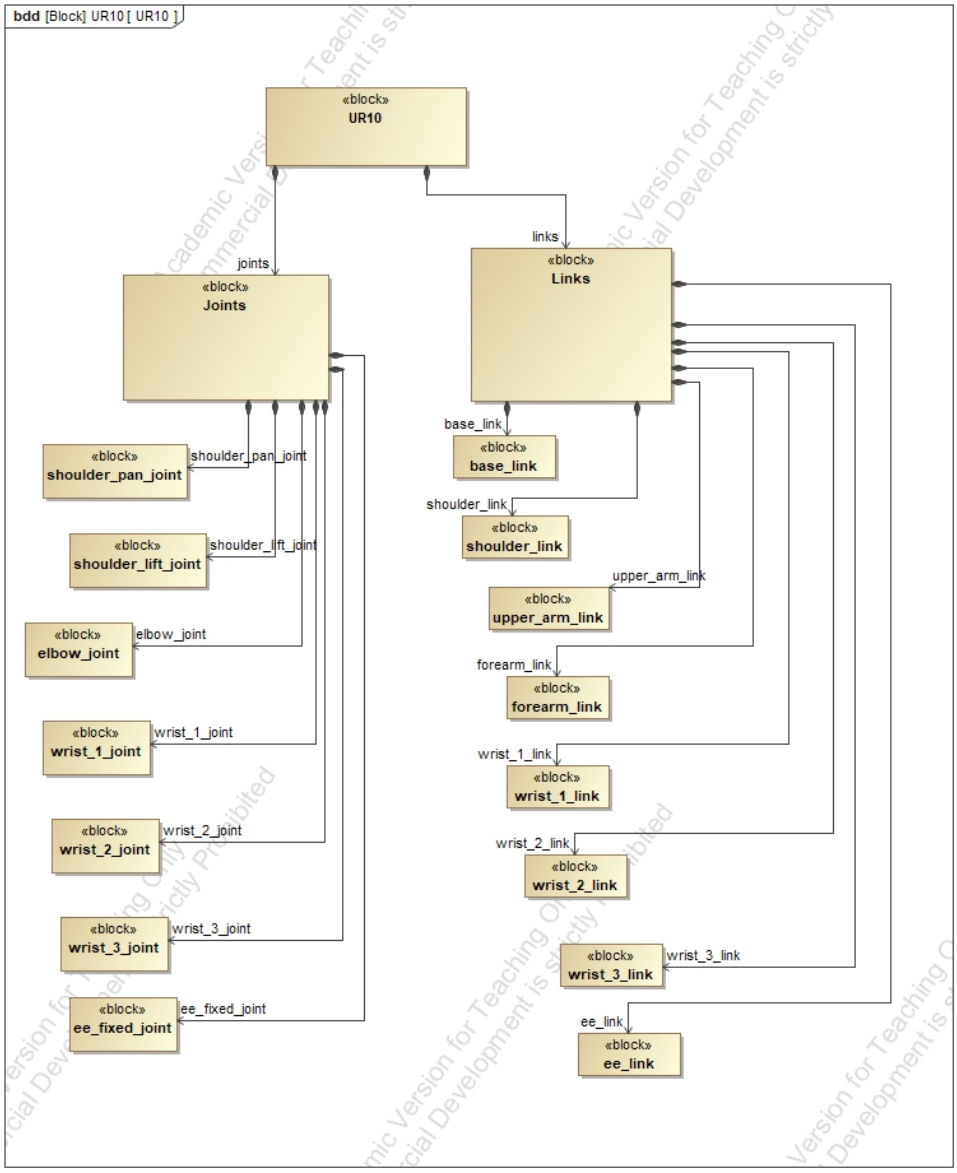

Figure 4.6: System-Level BDD of the UR10 robot.

4.3 Simulation Activity and Sequence Diagrams

Behavior diagrams were built for the laboratory work as described in Chapter 2, 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 [10]. The activity diagram for the robotic simulation is shown in Figure 4.7. 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, a robotic simulation software. 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 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 [10]. 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; which is split up into Figures 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, and 4.14; 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 will provide the steps and control messages necessary to control a robotic simulation to use the UR10 and Shadow Hand robot in simulation to detect and correct for object slippage.

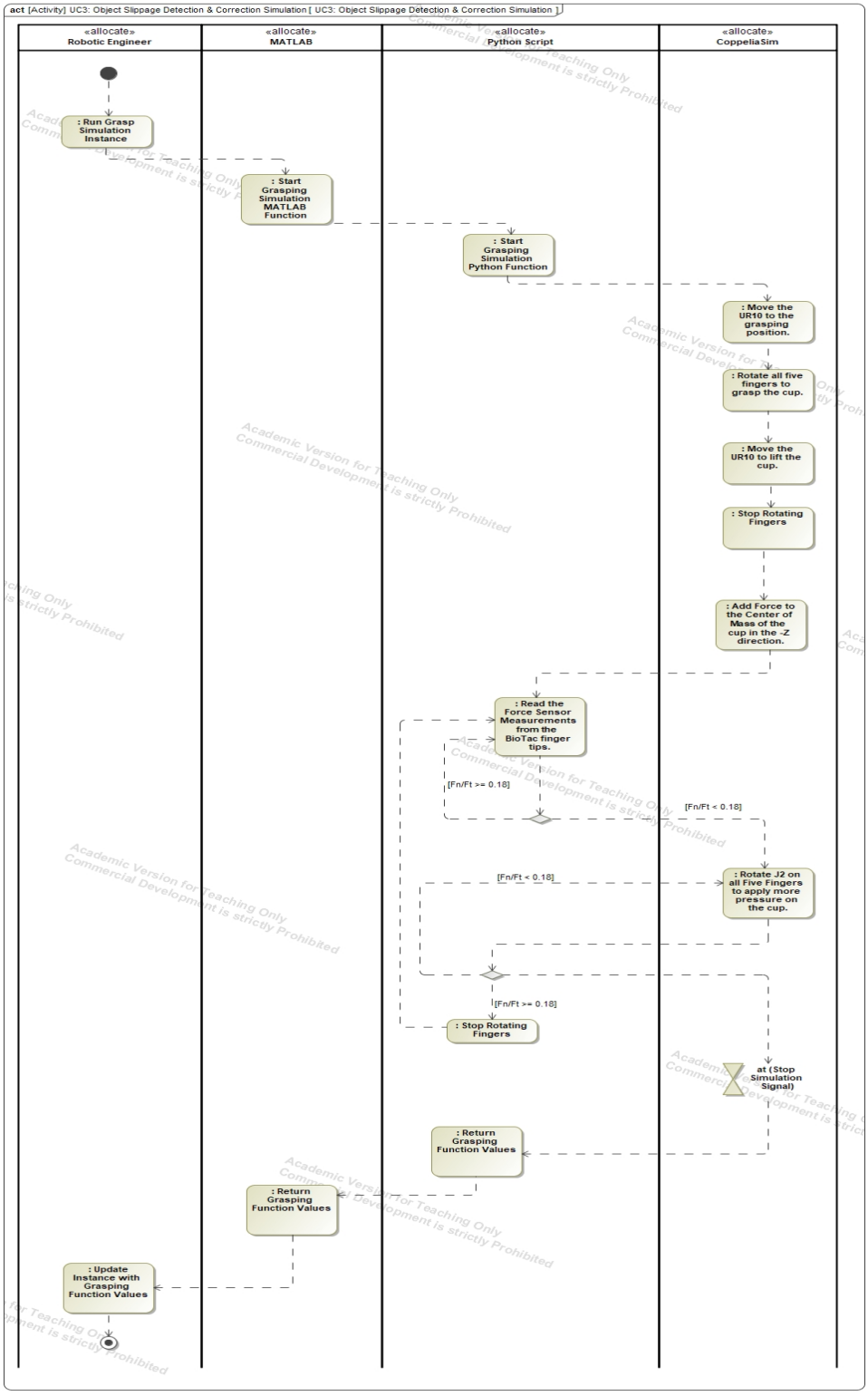

Figure 4.7: Activity Diagram for the robotic simulation.

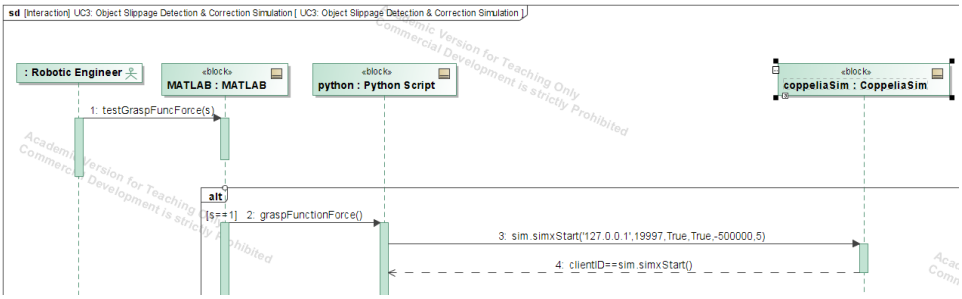

Figure 4.8: Section 1 of sequence diagram for the robotic simulation.

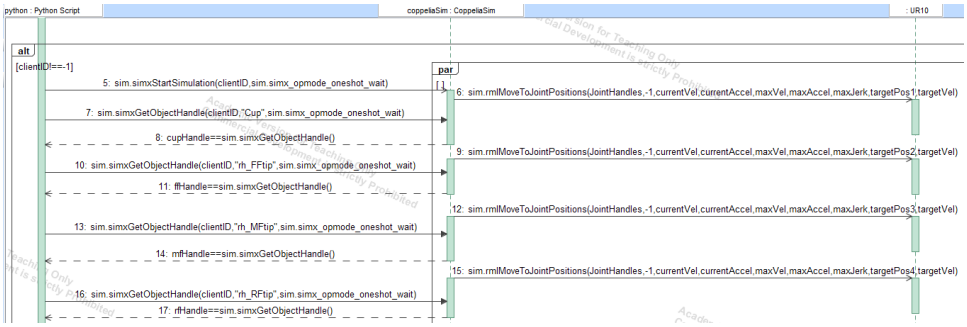

Figure 4.9: Section 2 of sequence diagram for the robotic simulation.

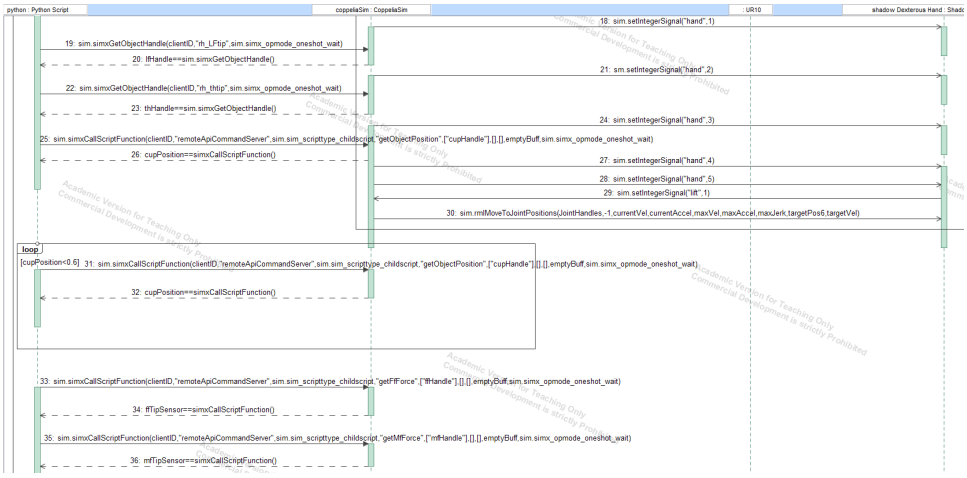

Figure 4.10: Section 3 of sequence diagram for the robotic simulation.

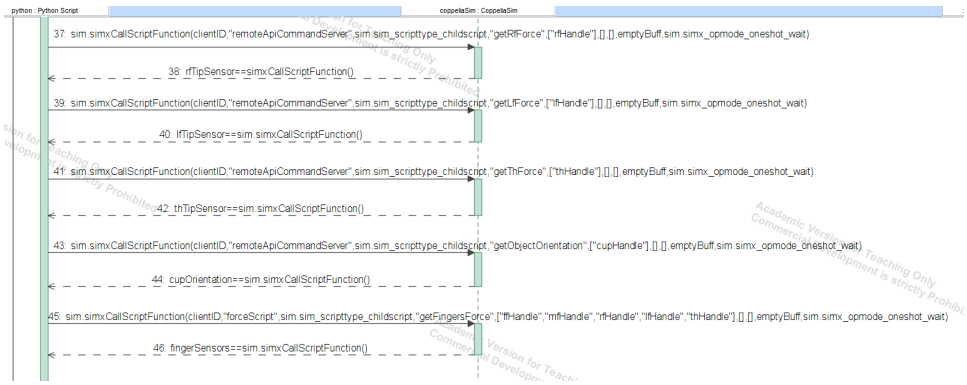

Figure 4.11: Section 4 of sequence diagram for the robotic simulation.

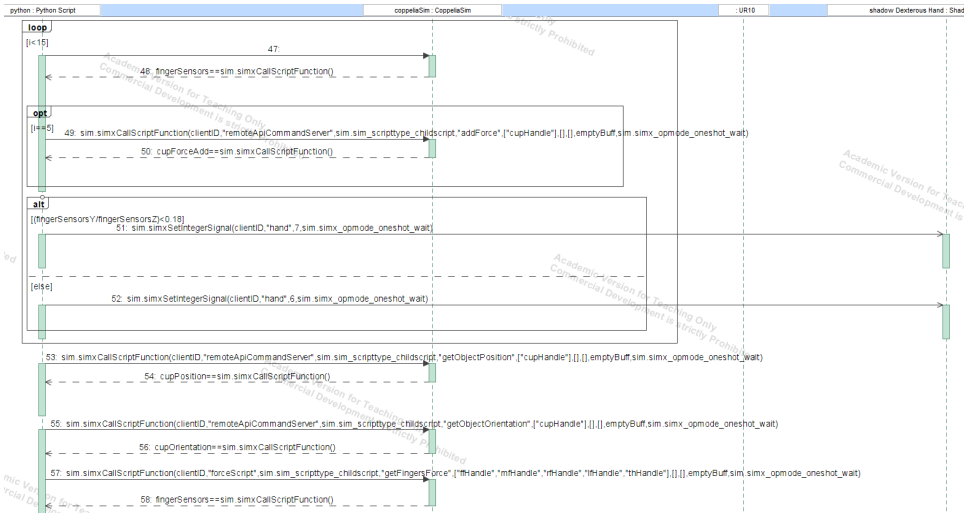

Figure 4.12: Section 5 of sequence diagram for the robotic simulation.

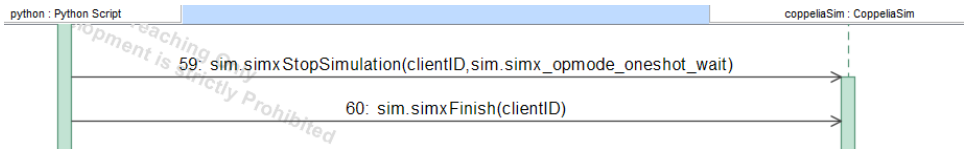

Figure 4.13: Section 6 of sequence diagram for the robotic simulation.

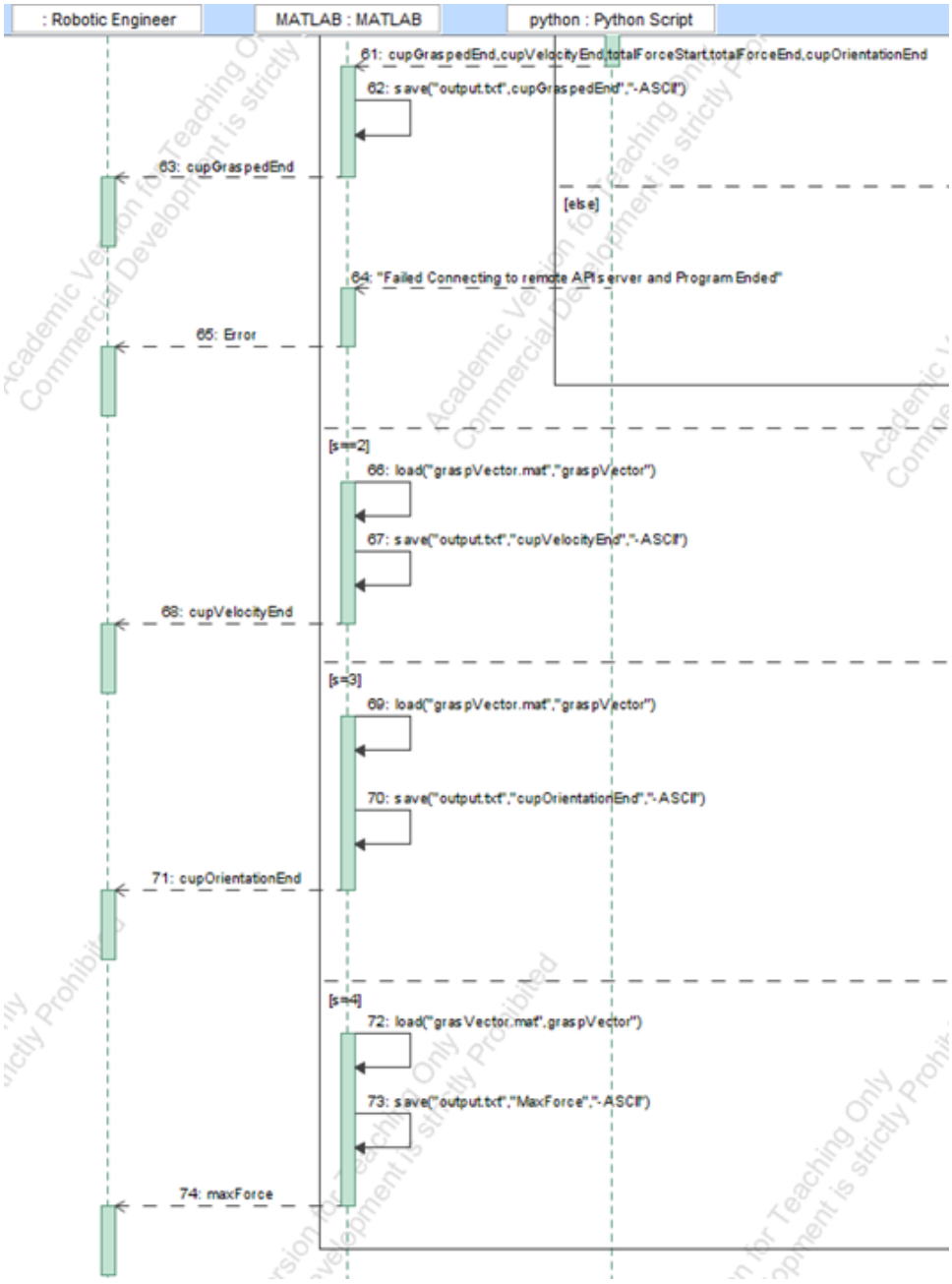

Figure 4.14: Section 7 of sequence diagram for the robotic simulation.

Chapter 5: Robotic Simulation in CoppeliaSim

There are a variety of choices of software that can simulate robotic tasks from custom software created by companies for their own use to open source software that anyone can use. Two of the most popular choices for robotic simulation software are Gazebo and CoppeliaSim (formerly V-REP). Both of these simulators work well with the ROS (Robotic Operating System) which has also become a popular robotic platform for academia and industry. While Gazebo is more closely integrated with ROS, CoppeliaSim has provided a more stable simulation of the Shadow Hand grasping a cup for the slippage detection and correction problem.

Therefore, this chapter will review CoppeliaSim which was chosen to simulate the slippage detection and correction problem. An overview of the features and capabilities of CoppeliaSim will be presented as well as how the CoppeliaSim simulation is externally controlled. Further, the chapter will conclude with how multibody dynamics is calculated in CoppeliaSim, and the physics engine chosen for this simulation.

5.1 Overview of CoppeliaSim

CoppeliaSim is a robotic simulation software that provides distributed control of models and elements of the simulation to the user. CoppeliaSim evolved from V-REP which both were created by Coppelia Robotics. The simulation can be controlled through ROS nodes, BlueZero nodes, embedded scripts, plugins, remote API clients or custom solutions [11]. The user also has the choice to program controllers in multiple languages such as Octave, Lua, Python, C/C++, Java, MATLAB, or Urbi which makes CoppeliaSim very open and user friendly [11]. Further, the same code that is written to control the simulation can be used to control hardware. The many features and capabilities of CoppeliaSim are summarized in the next section.

5.1.1 Features and Capabilities

The main features of CoppeliaSim include six programming approaches, ability to program in six languages, remote Application Programming Interface (API), access to four physics engines, collision detection, inverse and forward kinematic calculations, path and motion planning, vision and force sensors, convenient model browser, and many other features are available [11]. These features allow the user to create complex and powerful robotic simulations. The user interface for CoppeliaSim EDU which is the free educational licensed version of CoppeliaSim is shown in Figure 5.1.

The model browser as shown on the left in Figure 5.1 allows the user to select models of components and drag them into the main simulation area in the

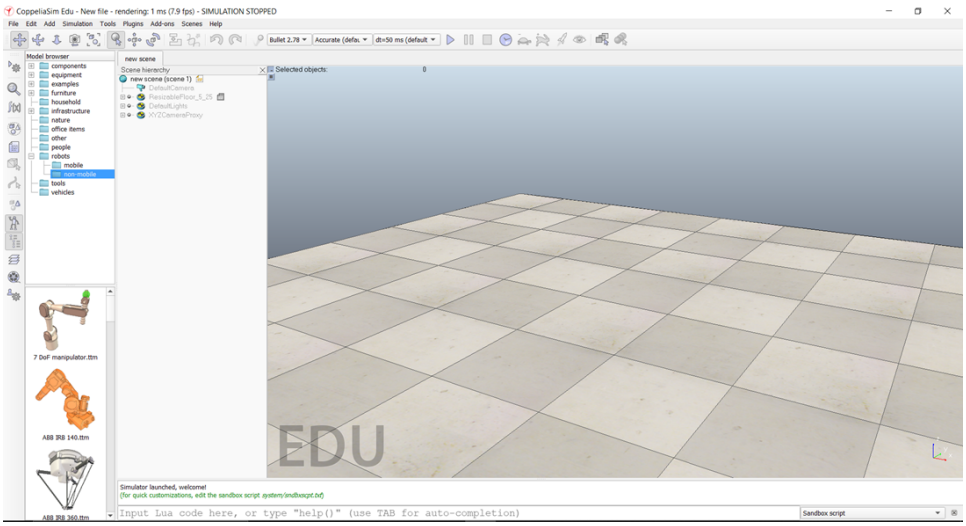

Figure 5.1: CoppeliaSim user interface.

center. This includes robot models that are pre-installed with the simulation software and ones that are saved by the user. The user can build robot models within CoppeliaSim or import robot models from their URDF provided by other robotic companies or researchers. Users can also save their own specific layout of their models for their simulation, embedded customized scripts, and other customizations as a CoppeliaSim scene. This allows the user to create specific scenes with various different versions of one simulation for instance. As well as, this scene can be sent to other users of CoppeliaSim without those users needing to download separate custom files to run or change the simulation.

The ability to write a simulation in multiple languages allows the user more flexibility and does not force the user to learn a tailored language that only works for this simulator. Further, there are over 100 CoppeliaSim functions that allow the user to control a simulation, the simulator, or a robot remotely as part of the

remote API feature [11]. The remote API feature is shown by the graphic in Figure 5.2 and discussed further in Chapter 5.2.

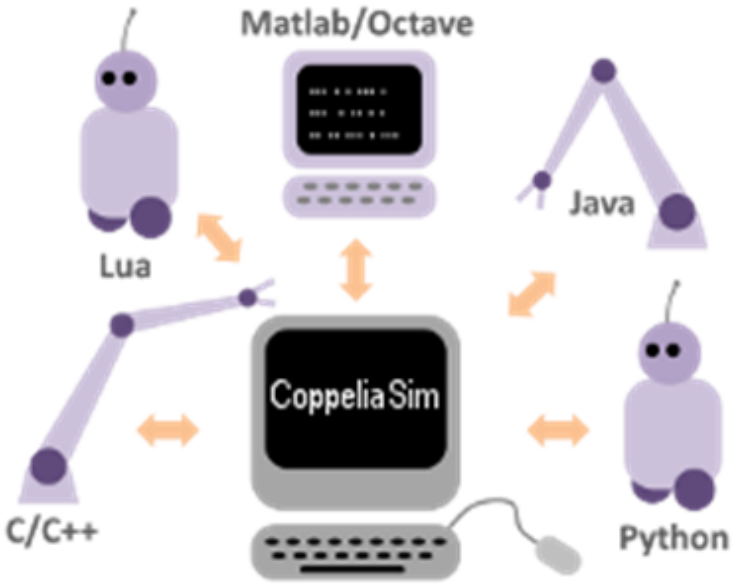

Figure 5.2: Remote API connections from CoppeliaSim [11].

Other powerful features include access to four physics engines; Bullet Physics, Open Dynamics Engine (ODE), Vortex Studio and Newton Dynamics; which provides the ability to select which physics engine will calculate the dynamics during the simulation [11]. This is necessary since some physics engines provide more stability in simulation which is needed for grasping simulations. In order to go from one physics engine to using another is made simple by selecting the physics engine from a drop down bar as shown in Figure 5.3.

Overall, CoppeliaSim provides many capabilities and features that can support

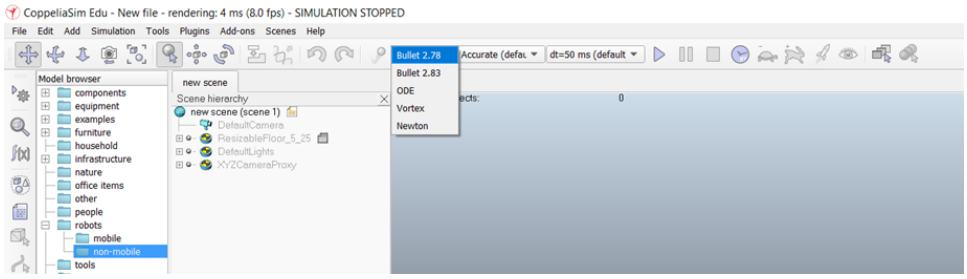

Figure 5.3: Picture of physics engines toolbar drop down list.

the creation of powerful robotic simulations which is why CoppeliaSim was selected as the robotic simulation software for this work.

5.2 External Control of CoppeliaSim Simulations

There are multiple ways to control CoppeliaSim simulations. For the slippage detection and correction problem, it is important to be able to control the robotic simulation externally. This was accomplished through the use of the remote API feature which allows communication between CoppeliaSim and external applications. The remote API feature used is based on the BlueZero (BO) middleware and its interface plugin to CoppeliaSim. The BO-based remote API allows the user to control CoppeliaSim simulations from an external application such as a script written in Python, C/C++, or a program written in MATLAB [11]. The user can run multiple external applications to control the same simulation at the same time, and these calls are hidden from the user during the simulation.

The messages sent from the external applications to control or interact with the CoppeliaSim simulation can be synchronous or asynchronous. Synchronous mes-

sages are used when the external application is running in sync with each pass of the simulation where asynchronous is used when they are not in sync [11]. These messages are sent between the client and server sides of the BO-based remote API. The client side is the external application, and the server side is implemented via a CoppeliaSim plugin and a embedded Lua script in CoppeliaSim [11]. This BO-base remote API feature was used for this work in order to run the robotic simulation externally which was listed as a simulation requirement.

5.3 Simulated Dynamics in CoppeliaSim

It is important to use a simulation software that calculates the dynamics of the models or elements used within the simulation especially if the purpose of the simulation is to mirror a robotic task completed with real hardware. As stated previously, CoppeliaSim allows the user to choose between four different physics engines in order to calculate the dynamics of the multibodies in the simulation. The four options are Bullet physics, ODE, the Vortex Studio, and the Newton Dynamics Engine. The reason for providing different options is due to the fact that physics simulation is a complex task and each engine offers different performance measures such as precision or speed of simulation, and each engine might have different features to choose from depending on the task being simulated such as how friction is calculated [11]. The physics engine selected for the robotic simulation of the slippage detection and correction problem was the Vortex Studio since it provided a very stable grasping simulation.

5.3.1 Vortex Studio Software's Multibody Dynamics Engine

Vortex Studio's physics engine has a unique blend of speed, accuracy, and stability that is used to provide high-fidelity and interactive simulations [12]. Vortex Studio's simulations involve rigid body calculations with constraints. The rigid bodies represent moving components of a mechanism such as robotic links and joints, and the constraints create restrictions on the movement of the rigid bodies such as a robotic joint between two robotic links [12]. Vortex's simulation is advanced in discrete time steps where the workflow followed during a single time step is shown in Figure 5.4.

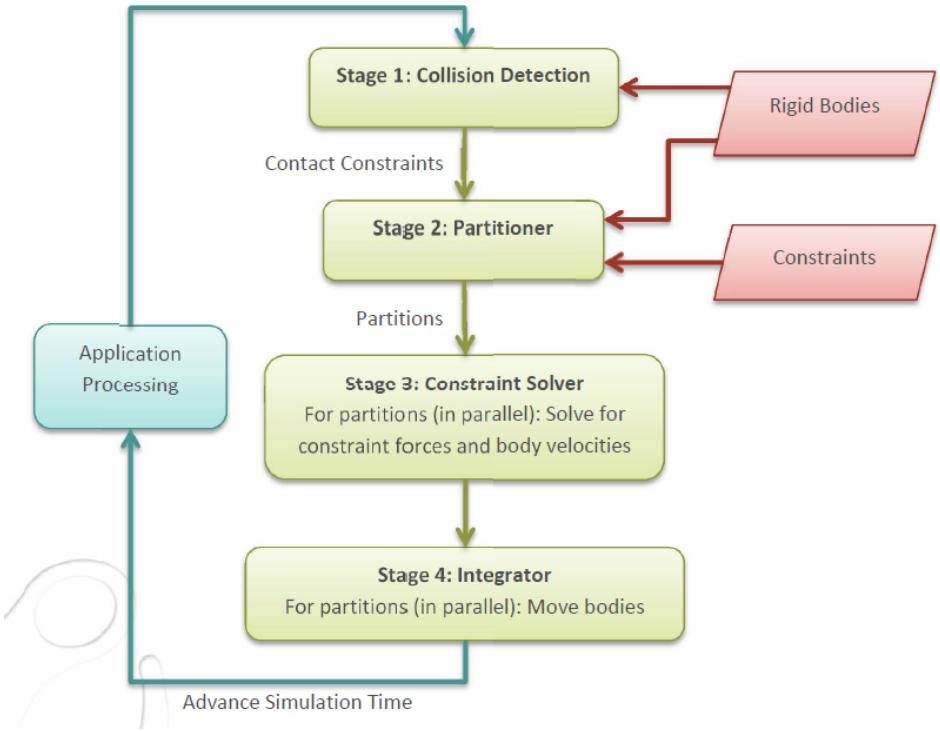

Figure 5.4: Vortex Engine's Simulation Loop [12].

During the collision detection stage, contact constraints are generated and added to the simulation in order to model the physical interactions between the collision of rigid bodies [12]. In order to detect collisions between simulated bodies, each model has certain attached collision geometries which are used to define the bodies shape. This software then uses a collision detection algorithm that performs fast and efficient overlap tests on bounding volumes of the collision geometries and then progressively more accurate and less conservative but slower tests [12]. This results in a small set of pairs of overlapping geometries. Intersection tests are then performed to find a final set of overlapping collision geometries. This process is more computationally efficient than a following a brute force method to figure out the collision points between colliding bodies [12].

During the partitioner stage, the partitioner organizes rigid bodies into partitions where every partition has a subset of rigid bodies with their related joints [12]. These partitions contain rigid bodies that only affect each other through constraints and are automatically computed. Partitions can be coupled or not coupled where partitions are coupled if there is a single constraint between any rigid bodies in a single partition or across partitions [12]. Coupled partitions will affect each other in simulation, and not coupled partitions are solved in parallel and independently during simulation.

During the constraint solver stage, partitions are received and processed to compute the forces applied by the constraints [12]. Partitions that are not coupled are processed in parallel which saves time during computation. The coupled partitions are initially solved isolated and in parallel with interaction forces that are

applied to the rigid bodies at interaction boundaries computed as well [12]. This is repeated iteratively, and once the constraint forces are calculated, the rigid bodies can be moved to their new location at the end of the simulation step which is done in the integrate stage for each partition in parallel [12]. Lastly, these simulations are highly customizable and can be tailored to the users needs.

Chapter 6: Simulation of the Slippage Detection and Correction Problem

In Chapter 5, the simulation software, CoppeliaSim, was reviewed along with the choice of Vortex Studio as the physics engine used during simulation and reasons why this software and physics engine were chosen to simulate the slippage detection and correction problem. This chapter goes in to the details of the simulation set up and how the slippage detection and correction problem was replicated in simulation.

6.1 Simulation Set up and Overview

One of the first simulation requirements, S.1.1.1.1, was to simulate the robotic equipment that was used in the experiments in the laboratory. This required finding models to use for the Universal Robots UR10, the Shadow Hand Robot, and the BioTac SP sensors. Computer models for these robots were already created by the companies that produce these robots. These models are represented by URDF files which contain kinematic and dynamic data for the robots. The model for the UR10 comes preloaded into CoppeliaSim and can be found in the model browser. The model for Shadow Hand Robot was not included in the model browser. Therefore, it was necessary that CoppeliaSim had the ability to import the URDF file for

the Shadow Hand Robot so that creating the robot model from scratch could be avoided. This was completed through a URDF plugin that comes preloaded with the CoppeliaSim software. The URDF file was downloaded as part of a software package provided by the Shadow Robot Company.

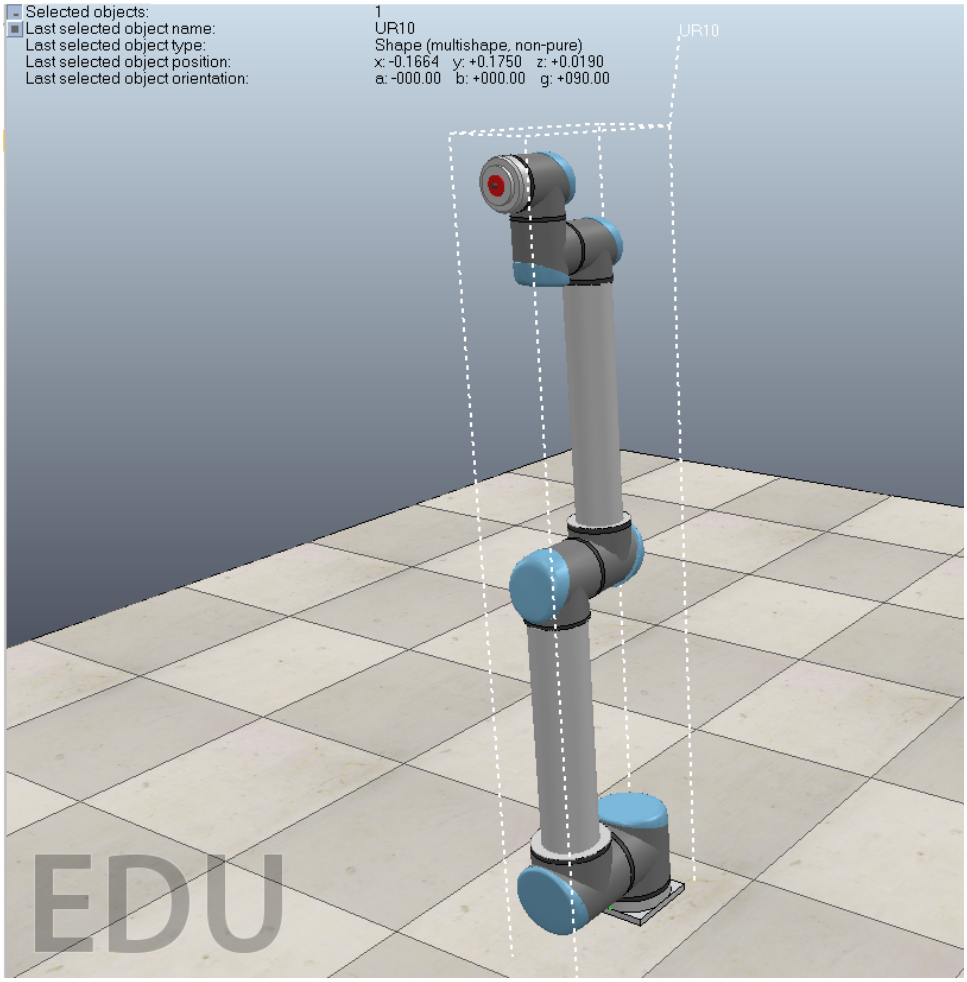

Figure 6.1: Robotic Model of the UR10 in CoppeliaSim.

The robot model for the UR10 in CoppeliaSim is shown Figure 6.1,and the robot model for the Shadow Hand Robot in CoppeliSim is shown in Figure 6.2. The BioTac SP sensors were also included in the Shadow Hand Robot model which is

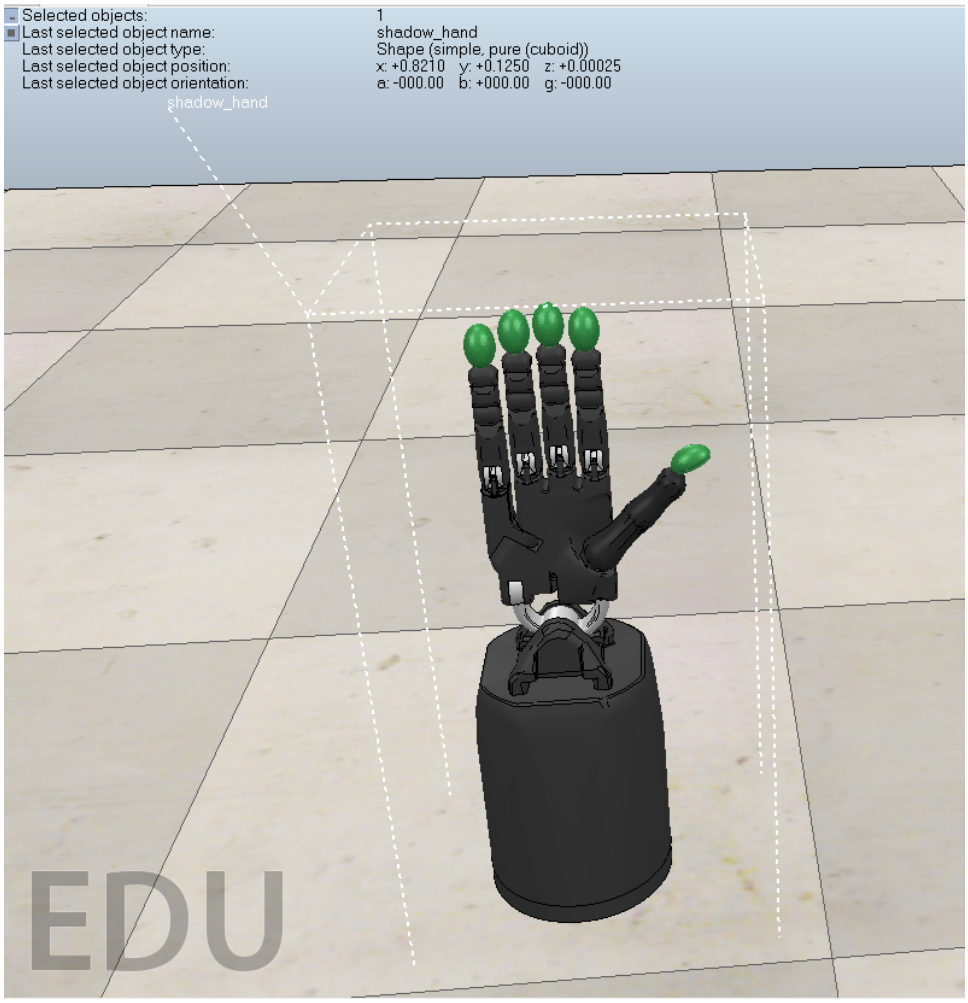

Figure 6.2: Robotic Model of the Shadow Hand Robot in CoppeliaSim.

seen as the green fingertips in Figure 6.2.

It was necessary to attach the Shadow Hand Robot model to the UR10 model through a force sensor which is the red circle in Figure 6.1 so that both models become one robot model for the UR10 with the Shadow Hand Robot attached as the end-effector as shown in Figure 6.3. In order to replicate the lab environment of the slippage detection and correction problem, the UR10 was lifted and attached to a table. Across from the UR10 and Shadow Hand robot, a model of a cup was placed

on a table and within the reach of the UR10 and Shadow Hand. Two simulated cameras were also added to the environment in order to provide different views of the simulation. This final environment, shown in Figure 6.4, remained constant throughout every simulation run.

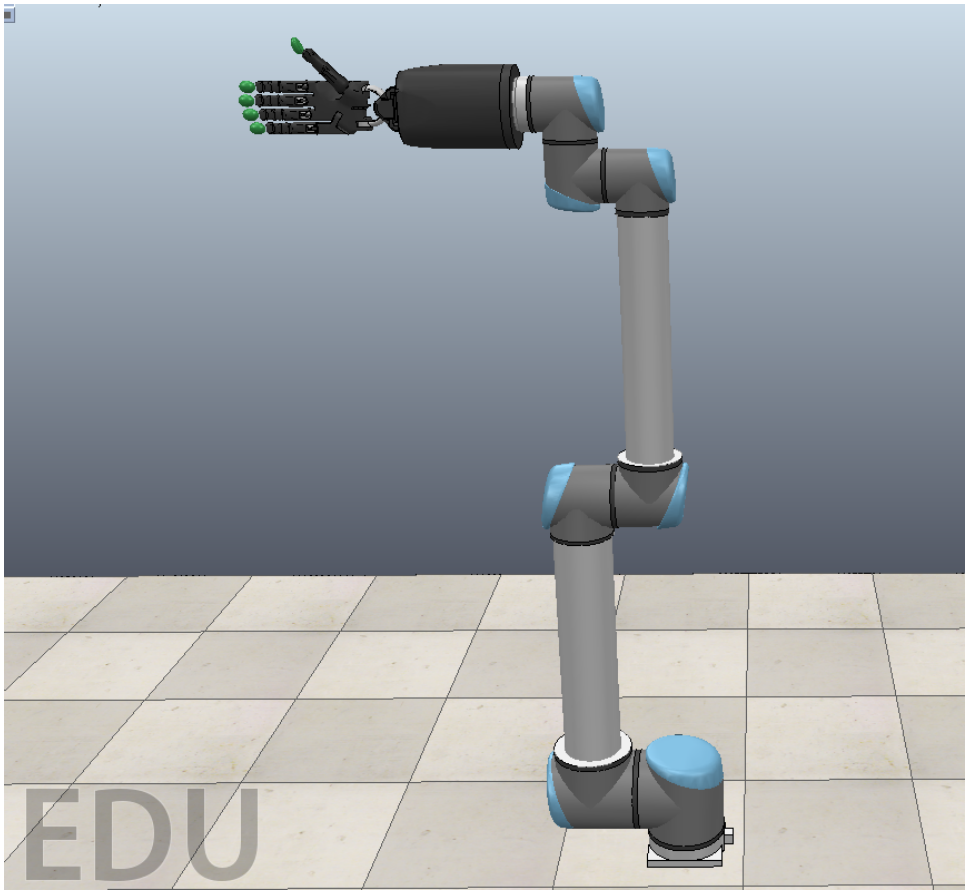

Figure 6.3: Robotic Model of the Shadow Hand Robot model attached to the UR10 robot model in CoppeliaSim.

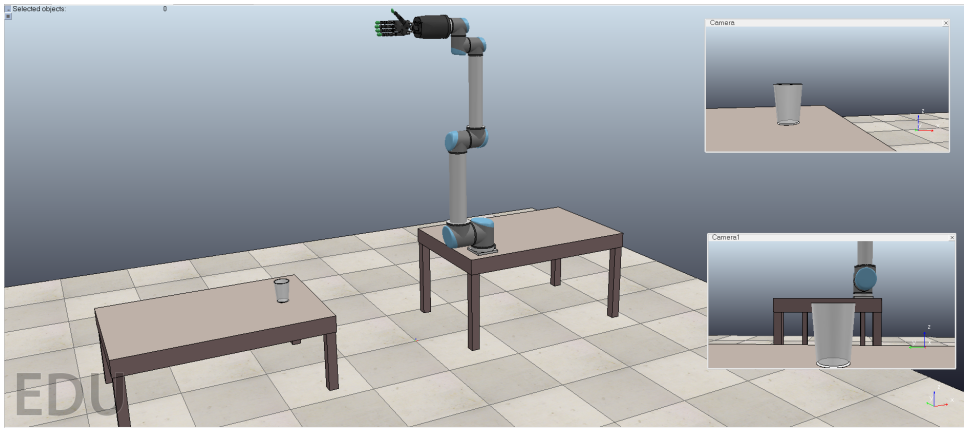

Figure 6.4: Simulation environment set-up used for every simulation run.

6.2 Slippage Detection and Correction in Simulation

The slippage detection problem as discussed in Chapter 2 requires the robotic hand to detect the moment a cup slips from the grasp as weight is being added. In the lab, this was accomplished by using a statistics-based detection algorithm that used tactile data from the BioTac SP sensors attached to the Shadow Hand Robot. While simulating the full set of raw data that comes from the BioTac SP sensors was not possible, a CAD (Computer-aided Design) model of the BioTac SP sensor was used as the fingertips for the Shadow Hand Robot. In order to detect the moment of slippage in simulation, force sensors were attached to the BioTac SP fingertips. CoppeliaSim force sensors are able to detect forces in three dimensions as well as torques in three dimensions between two rigid objects [11]. The forces and torques measured by the force sensor are shown on the left in Figure 6.5, and the force sensor inside of the BioTac SP fingertip is shown on the right in Figure

6.5.

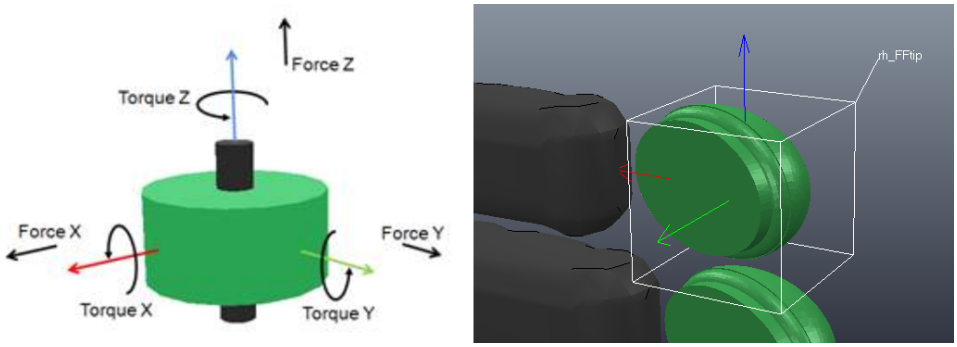

Figure 6.5: The forces and torques measured by the force sensor [11] on the left and force sensor within BioTac SP fingertip on the right. The blue axis is the z-axis, the red axis is the x-axis, and the green axis is the y-axis.

The force sensors are within all five BioTac SP fingertips. When the BioTac SP fingertips make contact with the cup in simulation, the force sensors will measure the amount of force that is applied by the fingertips to the grasped cup similar to the pressure sensor in the real BioTac SP sensors. The method chosen to detect the moment of slip and correct for slippage is based on work by Nicholas Wettels *et al.* where they used a BioTac prototype sensor to measure shear and normal forces at the fingertips and used this measurement to maintain perturbed objects within the force cone to prevent slip [4].

The Coulomb model of friction relates the tangential friction force magnitude to the normal force magnitude by

$$f_t \leq \mu \times f_n \tag{6.1}$$

where the normal force magnitude (f_n) is measured as the force in the y-axis, the

tangential friction force magnitude (f_t) is measured as the force in the z-axis by the force sensors in the BioTac SP sensors in simulation. μ is the coefficient of friction between the simulated cup and fingertips. This coefficient can be modified for each object in the simulation environment within the Vortex Studio physics engine settings as shown in Figure 6.6.

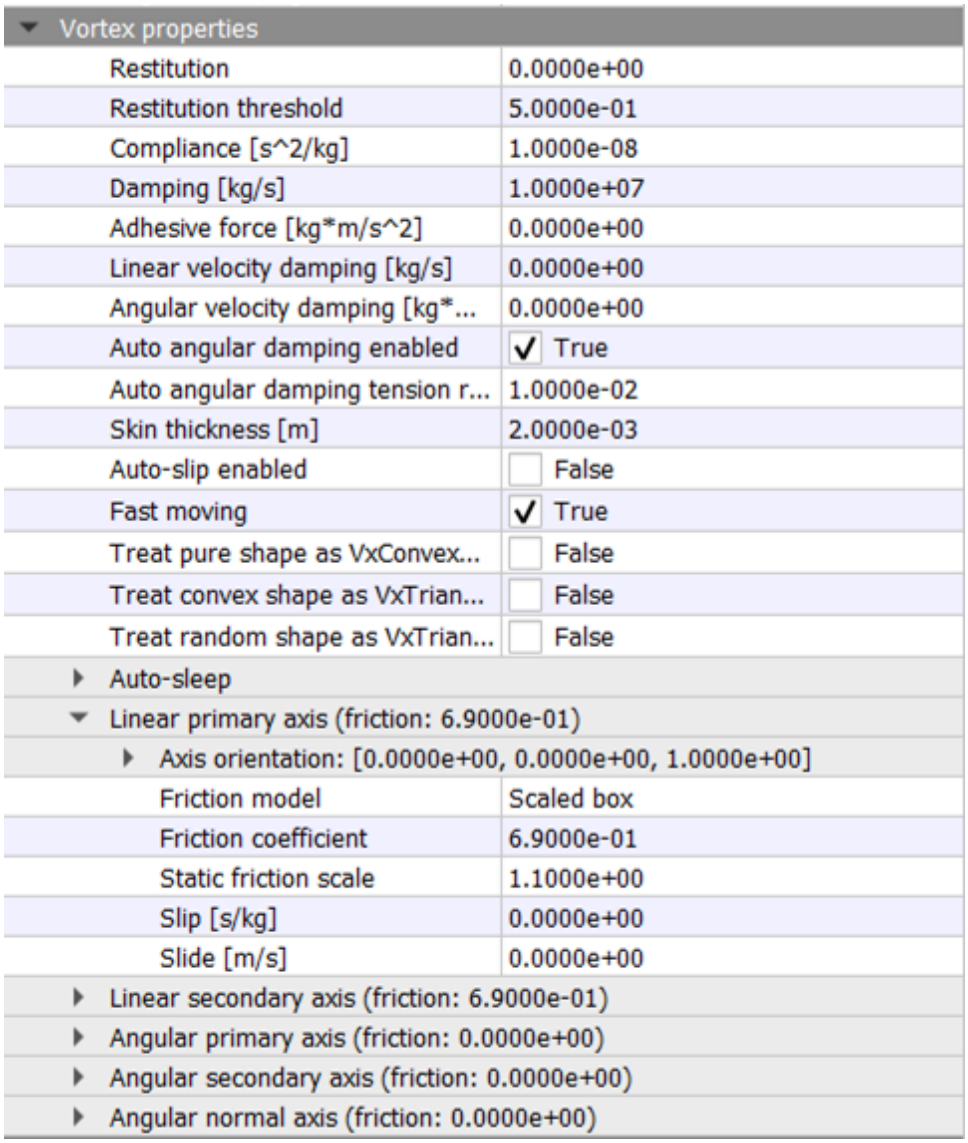

Figure 6.6: The Vortex Studio physics engine settings that can be modified for each model in the simulation environment which includes friction.

For the simulation to maintain a stable initial grasp, the friction coefficient for the materials were chosen such that the robotic fingertips were able to maintain a stable grasp of the cup while lifting the cup as shown in Figure 6.7.

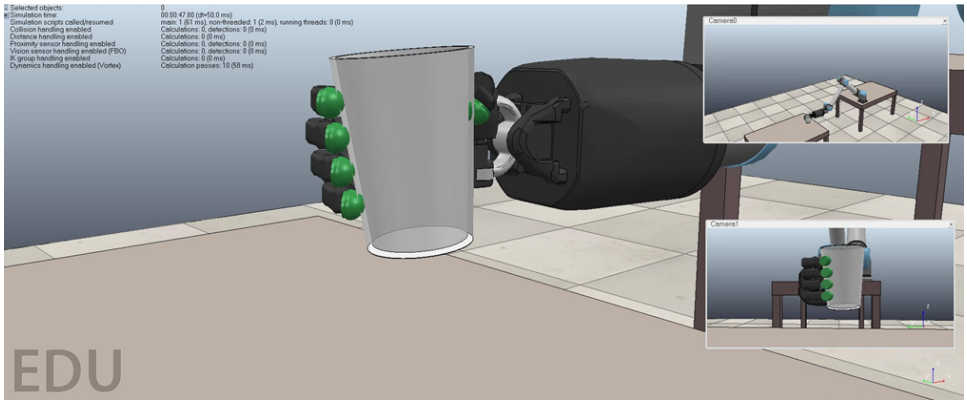

Figure 6.7: Friction coefficients for the materials were chosen to sustain a stable grasp of the cup while the cup was lifted.

Force was added to the center of mass of the grasped cup in the negative z-direction through the use of the sim.addForceAndTorque function which is an embedded function in CoppeliaSim. This force was added once the cup was held in the air as shown in Figure 6.7. The purpose of the slippage detection and correction algorithm created for this simulation was to detect when the cup starts to slip from the grasp after force was added to the cup such as weight force from adding bb pellets to the cup in the lab or the simulated force during simulation and stop that cup from falling from the grasp. In order to detect the moment of slippage in simulation, trials were run where force was added to the cup and data was collected from the fingertip force sensors to see what they measured when the cup slipped from the grasp. Figure 6.8 shows the total normal force and total tangential force measured

by the fingertips sensors during one of the slip simulation trials. The max force is the maximum applied force to the cup during the trial. The friction coefficients and maximum applied force remained the same for each trial. The results were the same for each trial, and the ratio of total normal force to total tangential force was calculated to find the moment when the cup slipped from the grasp. This moment is shown highlighted in yellow in Figure 6.8.

Max Force	0.42															
Force Step	0.001															
Friction Coeff Fingertips	0.7															
Friction Coeff Cup	0.9															
Time Steps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Total Force Normal	0.0085	0.0085	0.0085	0.0085	0.0085	0.0085	0.0085	0.0085	0.0076	0.0076	0.0076	0.0076	0.0076	0.0076	0.0076	0.0076
Total Force Tangential	0.046	0.046	0.046	0.046	0.046	0.046	0.046	0.046	0.046	0.0458	0.0458	0.0458	0.0458	0.0458	0.0458	0.0458
Force Normal/Force Tang	0.1843	0.1843	0.1843	0.1843	0.1843	0.1843	0.1843	0.1843	0.1645	0.1652	0.1654	0.1654	0.1654	0.1654	0.1654	0.1654

Figure 6.8: Example of trial data used to find the moment of slip during simulation.

From the simulation trials, the moment the cup was slipping from Shadow Hand's grasp was found to be when the ratio of the total normal force to total tangential force was less than 0.18 as shown in Equation 6.2 .

$$\frac{f_n}{f_t} < 0.18 \tag{6.2}$$

where f_n is the total normal force measured in the y-axis from the force sensors and f_t is the total tangential force measured in the z-axis from the force sensors. In order to stop the cup from slipping from the grasp, a command was sent to the Shadow Hand finger and thumb joints to rotate towards the cup to apply more pressure on the cup. A workflow was created to show the steps for the slippage detection and

correction algorithm for the simulation which was a part of the activity diagram developed in CSM. This section of the activity diagram is shown in Figure 6.9.

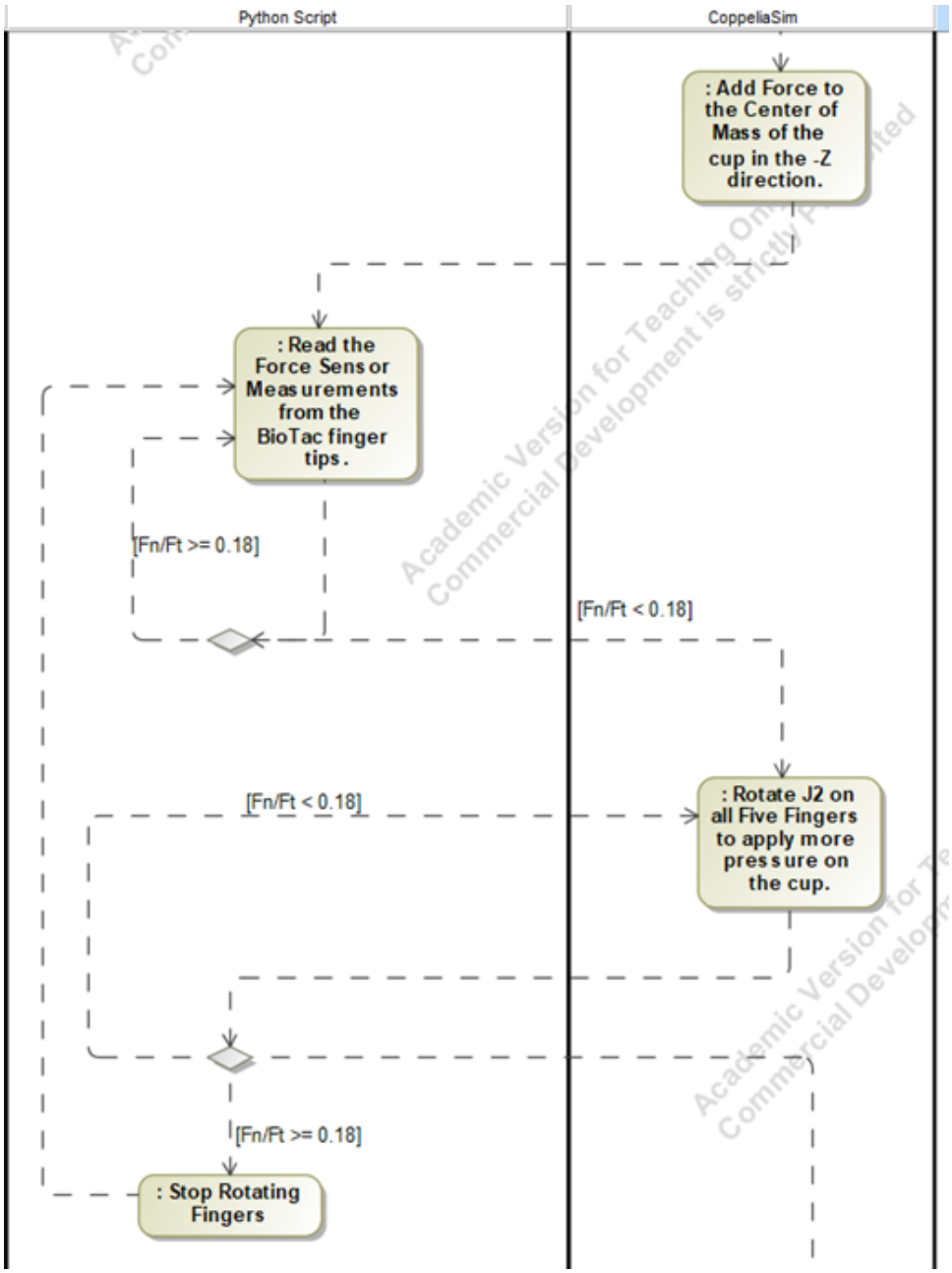

Figure 6.9: This is part of the activity diagram created for the slippage detection and correction use case which shows the slippage detection and correction algorithm workflow used for simulation.

As shown in the diagram, the detection and correction algorithm calculates the ratio of the total normal force to the total tangential force and compares this to the threshold ratio found earlier of 0.18. If the ratio is less than 0.18, the cup is slipping from the grasp and commands are sent to the finger and thumb joints (J2) to rotate in order to apply more pressure to the cup. This correction will stop once the ratio equals or passes 0.18. The rotational velocities for the finger and thumb joints are chosen so that the robotic hand stops the cup's motion before it falls from the grasp.

6.3 Robotic Simulation Connection to Cameo Systems Modeler

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 [13]. 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 6.10.

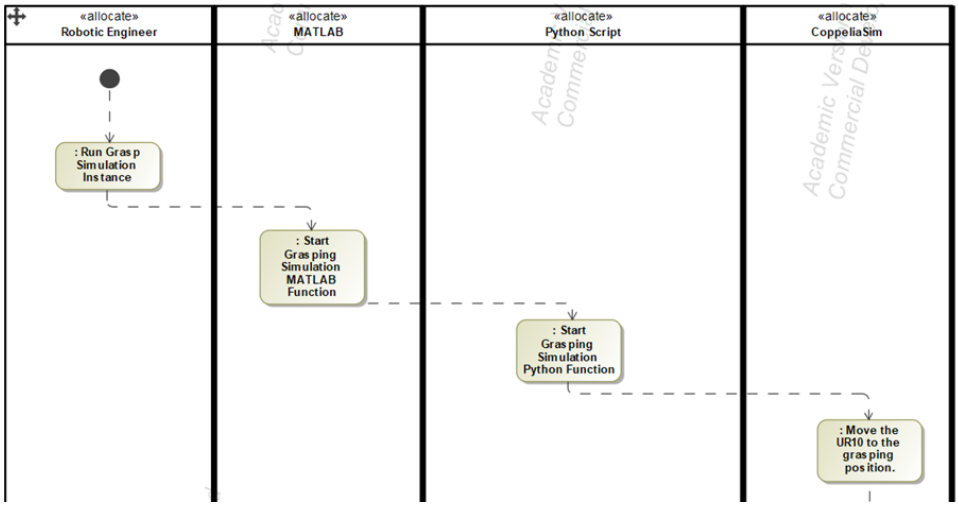

Figure 6.10: This is part of the activity diagram created for the slippage detection and correction use case which shows the connection from CSM to CoppeliaSim.

As shown in the first step of the workflow in Figure 6.10, 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 6.11.

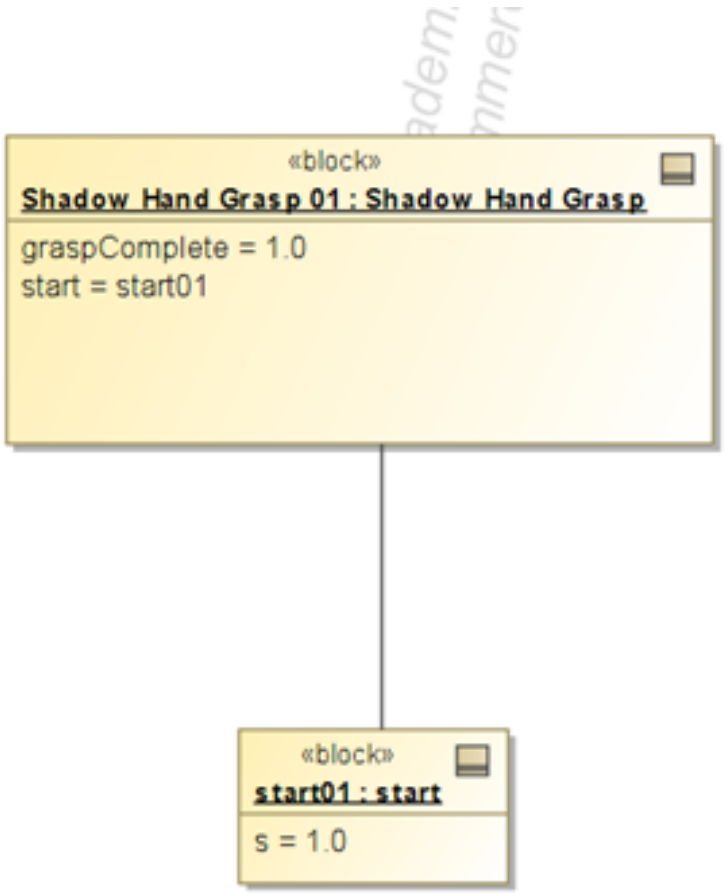

Figure 6.11: Example of an instance diagram built in CSM.

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 6.12.

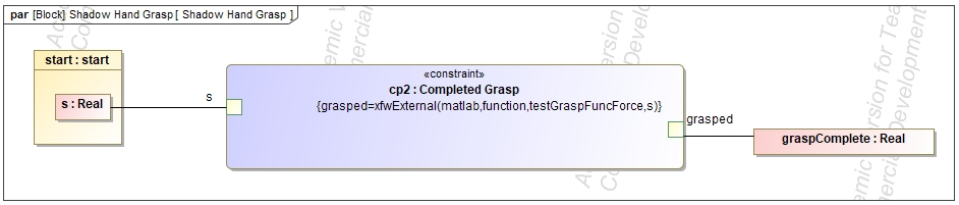

Figure 6.12: This parametric diagram shows the input, constraint block, and output used during the connection of CSM to MATLAB.

In order to start the simulation of the slippage detection and correction problem from CSM, an instance as shown in Figure 6.11 and a parametric diagram as shown in Figure 6.12 are created, and the following ®ParaMagic application window shown in Figure 6.13 is used to start the simulation.

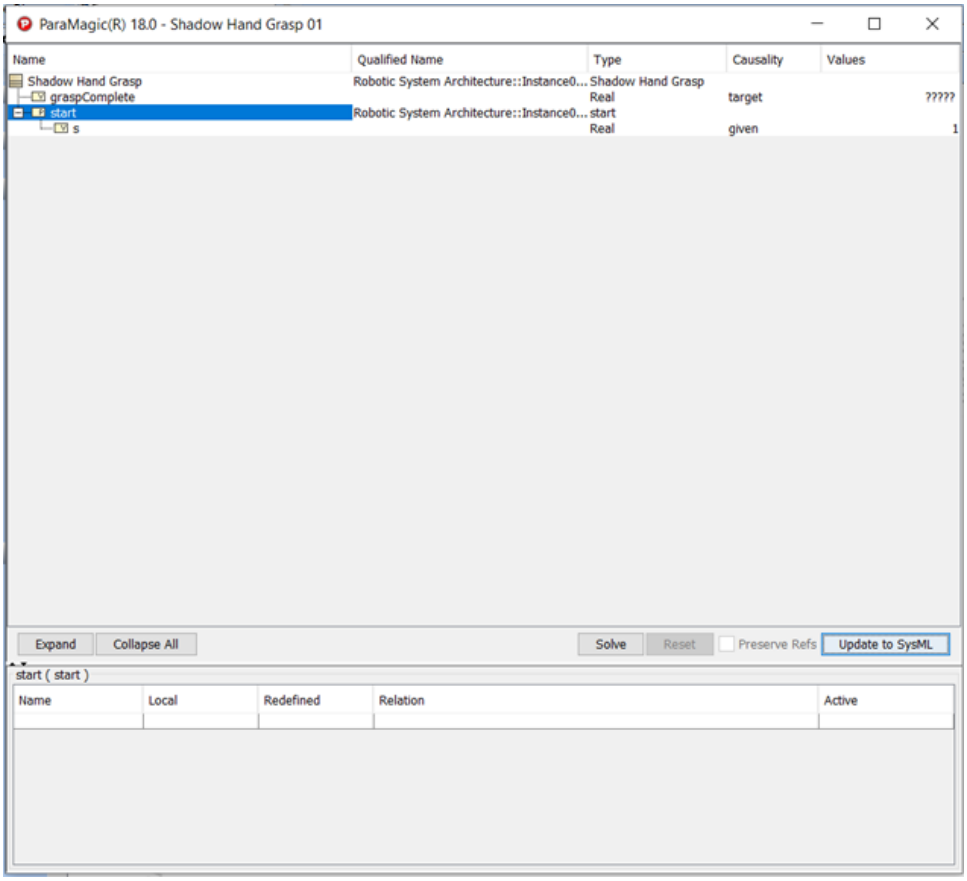

Figure 6.13: The ®ParaMagic application window used to initiate the robotic simulation from CSM.

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 and described in Chapter 4. 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 6.13.

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.

Chapter 7: Simulation Analysis and Results

This chapter will verify the slippage detection and correction simulation against the simulation requirements and validate the simulation against the stakeholder requirements. Further, the use of the simulation to select an optimal rotational velocity for the finger and thumbs joints used during correction will be presented.

7.1 Verification and Validation of the Simulation

In order to satisfy the stakeholder requirements, it is 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 nec-

essary which would indicate additional system elements built that are not needed.

The RTM created for the slippage detection and correction problem is shown in Figure 7.1. 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.

	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 Output Grasp Object Position					1				1				2
S.1.2.5 Output Grasp Object Orientation										1			1
S.1.2.6 Output Grasp Force											1		1
S.1.2.7 Output Grasp Hold Performance Metric				1									1
S.1.2.8 Output to CSM												1	1
Sum Check	N/A	2	3	9	5	1	1	3	5	4	2	7	

Figure 7.1: Requirements trace matrix for the slippage detection and correction problem.

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 [9]. The RVM created for the slippage detection and correction problem is shown in Figures 7.2, 7.3, and 7.4.

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 7.2: 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 7.3: 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 7.4: Third section of the requirements verification matrix for the slippage detection and correction problem.

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 [9]. 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 [9]. 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 [9]. The table in Figure 7.5 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 expect

for the accurate slip detection requirement. The reason for this is explained in the comments column. Overall, a slippage detection and correction algorithm was created for both the laboratory experiments and simulation experiments to solve 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 7.5: This table lists the stakeholder requirements for the slippage detection and correction problem and the validation of these requirements.

7.2 Model-Based Parameter Optimization

One advantage of building the simulation of the slippage detection and correction problem is having the ability to test parameters and ask questions that can be answered through the use of the simulation. This will save time and reduce the risk of experimentation if the tests are conducted in simulation before they are tested on real hardware. Also, it allows the user to run multiple tests faster than if done with the robotic equipment. As an example, a question that came up while solving

the slippage detection and correction problem was what the rotational velocity of the finger and thumb joints should be in order to stop the cup from slipping from the grasp of the Shadow Hand. The range of the choices for the velocities was from 0.1 radians per second to 0.95 radians per second with a step size of 0.5 radians per second.

The set up for the experiment to find an optimal velocity was to run the slippage detection and correction problem multiple times for each velocity and determine for which velocity the robotic hand successfully grasped the cup and maintained that grasp during the experiment. Even with using a small set of tests (10) per rotational velocity would require a total of 180 experiments. This would last days of continuous use of the robotic equipment if this was conducted in the lab. Instead with the use of the simulation built to replicate the slippage detection and correction problem, it took under three hours to complete all 180 tests with each simulation run lasting 55 seconds.

For each simulation, the position of the cup was tracked and if at the end of the simulation the cup was above or equal to a certain height than the cup was successfully held, and a 1 was recorded. If the cup was below that height, a 0 was recorded. After 10 simulations at the same rotational velocity, the percentage of times the cup was held was calculated. The raw data from the 180 experiments is shown in Figure 7.6. All simulation parameters were kept constant during each simulation. The only parameter changed between the sets of ten simulations was the rotational velocity of the finger and thumb joints involved in stopping the cup from slipping.

Correction Speed	1	2	3	4	5	6	7	8	9	10	Number of Successful Holds	Number of Failed Holds	Percentage of Success
0.1	0	0	0	1	1	0	0	0	0	0	2	8	20%
0.15	0	0	1	0	1	0	0	0	0	1	3	7	30%
0.2	0	0	1	1	0	0	0	1	0	1	4	6	40%
0.25	1	0	0	1	0	0	0	0	0	1	3	7	30%
0.3	1	0	0	0	1	0	0	0	1	1	4	6	40%
0.35	1	1	0	1	1	0	1	1	1	0	7	3	70%
0.4	0	1	0	0	1	1	1	0	1	0	5	5	50%
0.45	1	1	1	1	0	0	0	0	0	1	5	5	50%
0.5	0	1	1	0	1	0	0	0	1	1	5	5	50%
0.55	1	1	1	1	0	1	0	0	0	1	6	4	60%
0.6	1	1	0	1	1	1	1	1	1	1	9	1	90%
0.65	1	0	1	1	0	0	0	1	0	0	4	6	40%
0.7	0	0	0	1	0	0	1	0	1	1	4	6	40%
0.75	1	0	1	1	1	0	1	1	1	0	7	3	70%
0.8	0	1	1	0	1	1	1	0	1	1	7	3	70%
0.85	0	0	0	0	0	0	1	1	1	1	4	6	40%
0.9	1	0	1	0	1	0	0	1	1	1	6	4	60%
0.95	1	1	0	1	0	1	1	0	1	0	6	4	60%

Figure 7.6: Table of the simulation data from testing the range of rotational velocities to find the optimal rotational velocity.

As seen in Figure 7.7, the rotational velocity of 0.6 radians per second had the highest number of successful holds by scoring a 9 out of 10 or 90 percent. The flexibility of simulation allows the user to decide which parameters to change in order to tune the simulation to be similar to their problem. This solution can be transferred to the real equipment when setting the rotational velocity of the finger and thumb joints during correction.

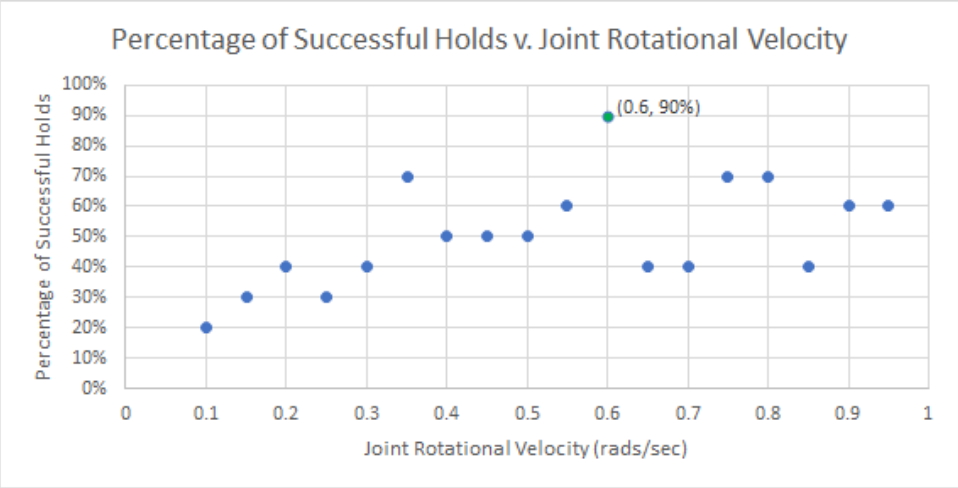

Figure 7.7: Chart of the percentage of successful holds versus the finger and thumb joint rotational velocity. The optimal velocity is highlighted in green.

Chapter 8: Conclusions and Future Work

In order for service robots or robotic hands to have the same level of dexterity and response as human hands, it is necessary to conduct research focused on increasing the haptic and dexterous ability of robotic hands. This interest led this group of researchers at the University of Maryland to use the five-fingered Shadow Robot Hand with BioTac SP sensors attached to investigate how to detect and correct for object slippage while weight or force is being added to the grasped object. Experiments were conducted in the ARC laboratory, and a statistics-based algorithm for slippage detection was created. This algorithm provided the Shadow Hand the ability to autonomously detect and correct for object slippage. Further, the haptic sensors were also used to classify the object held in the hand so that the crushing threshold of the object was not passed.

A simulation was never created for this experiment, and it is necessary to create a simulation that would replicate the slippage detection and correction problem. A simulation of the problem would allow for more tests to be conducted without the risk of equipment failure. However, the simulation would have to be a replication of the equipment used in the lab so that research conducted in simulation could be migrated to the real hardware. This led to using model-based systems engineering

techniques in order to build a replication of the slippage detection and correction problem in simulation.

Stakeholder requirements were created and at the end validated. System or simulation requirements were created and verified. Structural diagrams of the robotic equipment and laboratory environment were created. Behavior diagrams were created that were used in building the slippage detection and correction algorithm for the simulation. The measures of effectiveness were input into CSM, and these metrics were tracked from CSM to the robotic simulation software. The connection from the systems engineering tool, CSM, to the robotic simulator provided the ability to conduct more powerful robotic simulations and still use the systems engineering resources of CSM.

CoppeliaSim was used as the main robotic simulation software and provided a stable environment to conduct the slippage detection and correction simulations. Scripts were written in Lua, Python, and MATLAB in order to externally and internally control the robotic simulation. Lastly, the simulation was used to optimize the rotational velocity parameter for the finger and thumb joints used during the correction stage of the simulation. The robotic simulation created met the stakeholder needs and simulation requirements. Overall, it successfully replicated the slippage detection and correction problem conducted in the lab.

In the future, the group is looking to integrate simulation into the process when conducting experiments in the laboratory. The robotic simulation created for this work would need to be augmented to be more similar to the laboratory environment. This way it would be possible to transfer the algorithms created for

and run during simulation to the computer that controls the real hardware. This would include creating a better model of the BioTac SP sensors for the simulation so that the raw data collected from the sensors during the simulation matches the raw data collected during an experiment in the lab.

Bibliography

[1] N. Wettels, J. A. Fishel, and G. E. Loeb. Multimodal tactile sensor. *The Human Hand as an Inspiration for Robot Hand Development*, pages 405–429, 2014.

[2] Zhe Su and et. al. Force estimation and slip detection/classification for grip control using a biomimetic tactile sensor. *2015 IEEE-RAS 15th International Conference on Humanoid Robots*, pages 297–303, November 2015.

[3] M. A. Abd, I. J. Gonzalez, T. C. Colestock, B. A. Kent, and E. D. Engeberg. Direction of slip detection for adaptive grasp force control with a dexterous robotic hand. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2018.

[4] Nicholas Wettels, Avinash R. Parnandi, Ji-Hyun Moon, Gerald E. Loeb, and Gaurav S. Sukhatme. Grip control using biomimetic tactile sensing systems. *IEEE/ASME Transactions on Mechatronics*, 14(6):718–723, December 2009.

[5] M O'Toole, K Bouazza-Marouf, D Kerr, and M Vloeberghs. Robust contact force controller for slip prevention in a robotic gripper. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 224(3):275–288, 2010.

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

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

[8] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. *International Conference on Pattern Recognition*, pages 2756–2759, 2010.

[9] International Council on Systems Engineering. *Systems Engineering Handbook*. John Wiley and Sons, Inc., fourth edition, 2015.

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

[11] Coppelia Robotics. *CoppeliaSim User Manual*, 4th edition, 2019.

[12] CM Labs Simulations Inc. *Theory Guide: Vortex Software's Multibody Dynamics Engine*, February 2020.

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