VIRTUAL SIMULATION OF ROBOTIC OPERATIONS USING GAMING SOFTWARE

An Undergraduate Research Scholars Thesis

by

ERIC LLOYD REYSON ROBLES

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Stavros Kalafatis

May 2020

Major: Electrical Engineering

TABLE OF CONTENTS

Page

ABSTRACT....1

ACRONYMS AND ABBREVIATIONS....2

CHAPTER

I. INTRODUCTION....3

Background....3
Overview....5

II. METHODS....8

CAD Modeling....8
Game Engine....9
VR Integration....17

III. SYSTEM CHARACTERISTICS....18

Functional / Performance Requirements....18
Physical Characteristics....18
Failure Propagation....19
Physical Interface....19
Virtual Interface....20

IV. RESULTS....21

Reach Validation....21
Pick and Place Validation....23
VR Integration and Operation....32

V. CONCLUSION....33

REFERENCES....35

APPENDIX....36

ABSTRACT

Virtual Simulation of Robotic Operations Using Gaming Software

Eric Robles
Department of Electrical and Computer Engineering
Texas A&M University

Research Advisor: Dr. Stavros Kalafatis
Department of Electrical and Computer Engineering
Texas A&M University

The main objective of the project is to provide a Virtual Reality (VR) representation for robotic simulations using a game engine software such as Unity. It is useful to an ongoing research on testing the compatibility of training collaborative robots entirely in VR to further enhance the use of human-robot collaboration in industry. The success of the research project implies that collaborative robots can be tested and trained via software running numerous simulations more efficiently than the traditional operation of robotic designs in industry.

The UR10e robot arm is the industrial robot to be tested in VR. To provide the simulation for the research, four objectives are set across a timeline. A CAD model of the UR10e will be imported into Unity without any loss of data. Next, the UR10e's degrees of freedom (DOF) will be added into the game engine such that its movements have the same constraints as its actual design. Then, a 3D environment is added onto the game engine simulation such that the game engine robot can interact with a virtual workspace. Finally, the game engine robot is assigned to perform a simple task such as a pick and place activity. The robot must be able to perform the task autonomously and a user interface (UI) must also be added such that an operator can manually direct the robot's movements for the task given.

ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
DOF	Degrees of Freedom
EE	End Effector
FOV	Field of View
GUI	Graphical User Interface
HMD	Head Mounted Display
HMI	Human Machine Interface
HRC	Human Robot Collaboration
SDK	Software Development Kit
UI	User Interface
VR	Virtual Reality

CHAPTER I

INTRODUCTION

Background

It was not until the early twenty-first century [4] that the compatibility between robotics and VR technology emerged in research and development area as the quality of the human-computer interface of VR technology has improved drastically through its applications in the gaming sector [3]. VR simulations for operating industrial robots augment the efficiency and safety in robotic training [4] through its advanced visual graphics and real-time controller feedback [3]. Virtual simulations can be recorded and reevaluated such that accurate measurements are easily collected, and readjustments can be made. In addition, using VR interface to interact with robotic machines, the operator can experience real-world scenarios and assess detailed processes remotely.

There is a research project that used the game engine Unity to develop a VR simulation on virtual model of the robot design called ABB IRB 120 robot manipulator [Figure 1] [2]. The VR simulation involves an operator wearing a Head Mounted Display (HMD) and using controllers to manipulate the movements of the virtual robot which is mimicked by the actual robot in real-time. The Unity software provides game development tools to implement control algorithms for the virtual model such that it can accurately animate the robotic design to the extent that its movements invokes almost instant response from the actual robot. This project is one of the motivations for my current URS research. The research project will enhance this system by using VR representation to simulate a robotic task for the UR10e robot arm entirely in software such that the performance of the virtual robot design can later be translated into the actual robot using reinforcement learning techniques outside the scope of this project.

Figure 1. ABB IRB 120 Robot in VR [2]

In addition, this project is directly under an ongoing graduate research conducted in Texas A&M University. This graduate research aims to prove the compatibility of designs of collaborative robots in industry by conducting simulations entirely in VR to gather data for applying machine learning onto the UR10e robot model. Collaborative robots are automated systems that operate in tandem with human operators such that both assist one another in completing tasks (see Appendix in page 36). Unlike fully autonomous machines, cobots are guided through complex tasks by human operators who possess critical thinking and specialized skills required for these processes. Operators must interfere in some cases for precise performance of the robotic machines [3]. The implementation of VR teleoperation is useful for operating cobots such that the operator can visualize an immersive, tridimensional view of the actual robot while controlling it remotely. This can help evaluate the compatibility and reliability of training collaborative robots entirely in VR to further enhance the use of human-robot collaboration in the industry.

Traditional Human-Machine Interface (HMI) requires barriers to protect human operators from dangerously massive forces generated from the robotic machines [3]. On the other hand, VR eliminates safety risks while it allows the operator to control these machines in real-time with realistic conditions. In fact, the training duration for skill acquisition from reinforcement learning is shorter when using VR teleoperation than the conventional console [8]. Compared to the traditional architecture of manual control of the robotic design through a console, the proposed architecture replaces the console with VR technology.

Overview

The scope of the project generally involves the familiarity of CAD modeling, C# programming, and utilizing game engine tools and VR device components. The correct dimensions of the UR10e arm [Figure 2] can be designed and measured in a CAD software. In addition, individual parts of the robot arm can also be assembled, and joint constraints added using the tools provided by the CAD software. Within the game engine, C# scripts can be attached to game objects (see Appendix in page 36) to provide them with various functions and other modules. An algorithm will be written onto a script attached to the UR10e to direct the rotation of each joint of the robot arm and lead the end effector (EE) to the path directed. The game engine also provides numerous built-in components that can help fulfill different needs such as animating, adding laws of physics, adjusting camera perspective, and constructing a state machine that can handle different animations for multiple game objects. Furthermore, the robot arm's control architecture designed in the game engine will be integrated into a VR device that will become the user interface (UI) for the actual UR10e arm. Thus, an operator can control the actual robot arm remotely using an HMD and VR controllers.

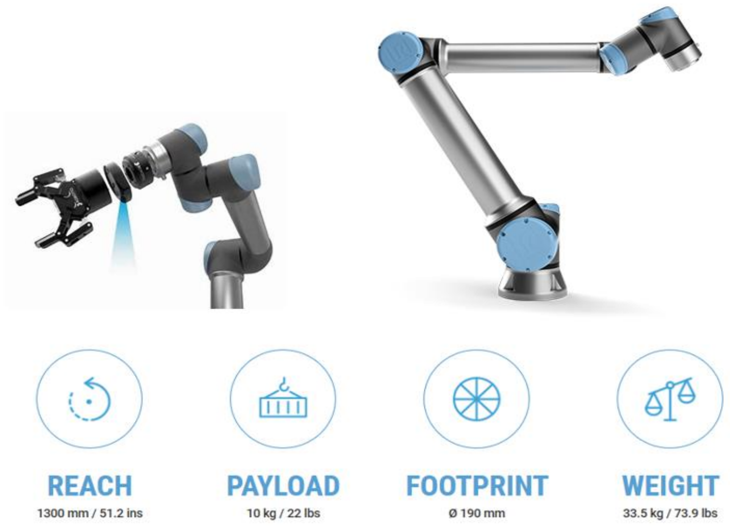


Figure 2. UR10e with 2-Finger Gripper

The research project aims to create a VR representation simulating simple tasks for the UR10e robot arm to perform entirely in software using the tools provided by the game engine. The simulation will test the control architecture of the UR10e arm such that it will function properly once it is integrated onto the VR device. There will be three subsystems that will encompass the entire project. CAD modeling is one subsystem where Fusion360 will be the CAD software used and the CAD parts of the UR10e including its EE can also be borrowed from several sources in GrabCAD which provides an open source library of CAD models. The second subsystem is the game engine and Unity is the game engine that will be used for the project since it is most often used in the research projects conducted for similar purposes. The final subsystem is the VR integration where the VR device that will be used is the HTC Vive VR system.

There are several expected limitations that might be encountered in the project. For instance, the tradeoff between accuracy and speed of operating the robot arm. Improvement in accuracy can compensate for the lack of speed of the design since faster operation of the arm has a tendency of overexerting motions due to the lag of detection from its sensors. One of the algorithms that are planned to be implemented in the project is the inverse kinematics solver [1] (see Appendix in page 36). This algorithm solves an optimization problem to minimize the distance between the robot arm's EE and its intended location. However, the algorithm can rarely achieve an optimal result regardless of the number of iterations to solve the optimization problem [1]. The VR technology may require the simulation to be enhanced in terms of graphics such that it will have an immersive virtual experience. The proposed project only aims to test the practicality of integrating robotics and VR technology onto a virtual simulation generated by a game engine.

CHAPTER II

METHODS

The methods for the research project are categorized into the three subsystems illustrated in [Figure 3]. The CAD modeling subsystem involves assembling and importing a CAD model of the robot arm. The game engine subsystem deals with designing a VR simulation of the robot arm operation and creating the control architecture of the robotic system using an optimization algorithm. The VR integration subsystem is where the control architecture designed in the game engine is integrated onto the VR HMD and controllers to control the actual robot arm.

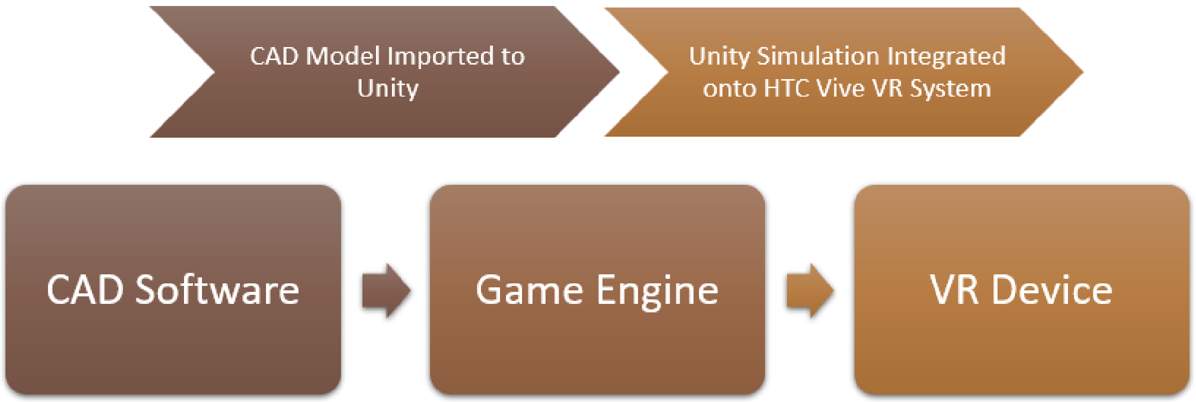

CAD Model Imported to Unity
Unity Simulation Integrated onto HTC Vive VR System
CAD Software
Game Engine
VR Device

Figure 3. System Level Description

CAD Modeling

In this subsystem, the CAD parts of the UR10e is assembled in Fusion360, a CAD software. Joint constraints are assigned to the designated joints of the robot arm corresponding to its six DOFs. The pivot points of the joints are also added using the CAD software. Afterwards, the CAD model is imported into the game engine in FBX file format [Figure 4]. The validation of the transfer is conducted by visually checking any deformations of the model in the game engine. Furthermore, the complete transfer of mesh data, which is a collection of vertices that renders the shape of the CAD model, is checked in the game engine. There must be no loss of data during the transfer from the CAD software to the game engine. A specific EE of the robot arm is chosen based on the chosen functionality of the robotic arm in the graduate research that will build upon this project. The EE CAD model will be assembled in Fusion360 and it is transferred to the game engine where it will be integrated onto the robotic arm.

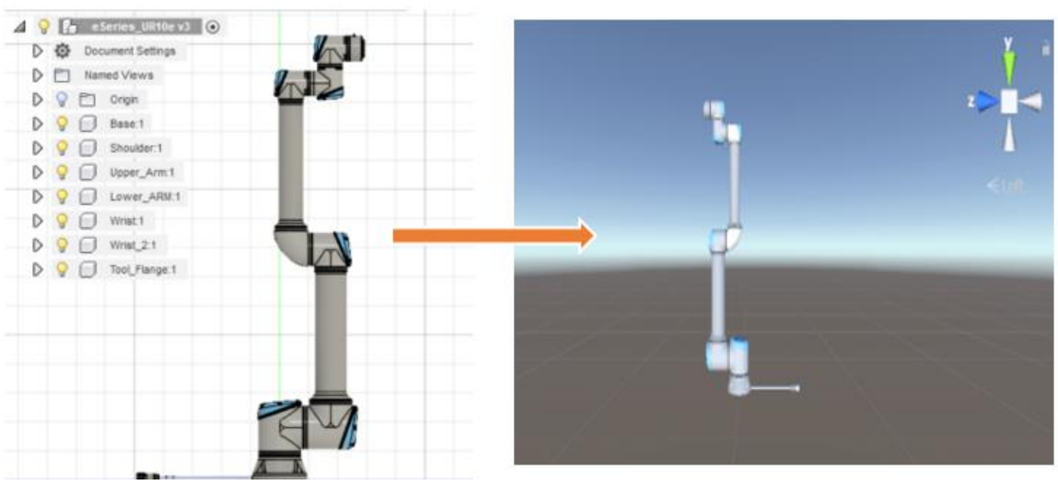


Figure 4. Import CAD Design to Unity as a Game Object

Game Engine

Most of the project's methods are executed in the Unity game engine. Unity provides editing tools and applications that are mostly used towards game development. Unity assets are also available to instantiate prebuilt components instead of reconstructing them. Furthermore, the Unity API provides various VR/XR SDK packages that target specific VR devices such as Oculus and HTC Vive [6]. Therefore, the VR design of the project is constructed and tested within the game engine to integrate it to the VR device used. After the CAD model of the UR10e is imported to Unity, the game engine model is checked for any deformation. The mesh of the

CAD model is a collection of vertices that encompass the shape and volume of each part. When the CAD model is imported to Unity, the mesh data must be completely preserved to indicate no loss of data. Unity’s mesh renderer component can then project the rigid shape of the UR10e model onto the simulation such that its dimensions are similar to that of the CAD model.

Once the complete transfer of the model is verified, a component called rigidbody is added onto each part of the UR10e robot arm. This component can control the transforms of game objects and subjects the them to the laws of physics such as gravity and collisions [6]. The rigidbody component also allows the addition and adjustment of a game object’s mass which affects the momentum of collisions with other game objects. It is important when the robot arm interacts with other game objects in the virtual environment. The robot arm has a certain payload that determines the maximum weight of the objects it can lift. This must also translate in the virtual environment where the robot model cannot lift objects with weights above its payload.

Colliders are also added in order to trigger collisions when they overlap each other [6]. A collider is a built-in Unity component that wraps the surface area of a game object to detect physical interactions with other objects. A collision between objects occurs when their colliders come in contact with each other. When a collision is detected, the objects’ respective rigidbody components will allow the Unity physics engine to determine how these objects will react from the collision. The collider can also be used as an event trigger where contact with other game objects will set active the event of the collision. When a collider is used as an event trigger, the rigidbody of the object will not cause physical contact with the other objects that will collide with it. In other words, the objects will pass through each other. Event trigger colliders are used in the project to differentiate collisions. Collisions between parts of the arm must be differentiated from collisions of the arm with other game objects. To prevent the arm from tangling with its own parts, the arm must stop moving when it comes in contact with itself.

For the UR10e, a mesh collider is needed to encompass the entire model based on the mesh data collected from the CAD model. When the CAD model of the robot arm is imported into Unity, each individual part of the UR10e has its own mesh that maps the shape and size of the object. Therefore, there are individual mesh colliders that can be derived from the meshes of the different parts of the robot arm [Figure 5]. These mesh colliders must recognize contact amongst themselves by using the event trigger mode and writing a script to handle the contact behavior.

The script will stop the operation of the arm when it detects an event of a collision triggered by certain parts of the arm. It will only stop when the arm collides with its upper and lower parts causing entanglement. The upper parts are the wrists and lower arm while the lower parts are the base, shoulder and upper arm. There is no trigger on the collision between the lower arm and upper arm because they are already connected to each other. The script will also allow the arm to have physical contact with other game objects by temporarily disabling the trigger mode of the colliders when they collide with the other objects. The behavior of the collision between two objects can also be influenced by the texture and stiffness of the objects. These attributes can be adjusted by adding a physic material into the collider of an object.

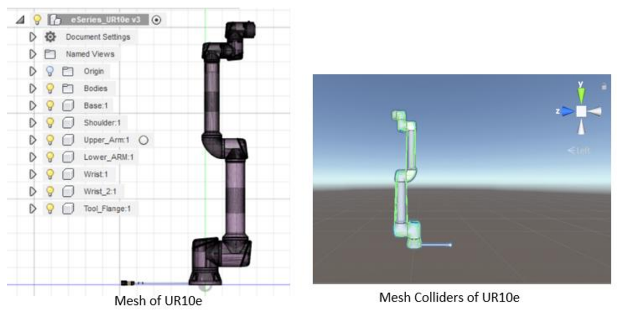


Figure 5. Mesh and Mesh Colliders of Robot Arm

A hierarchy of all the parts of the robot arm is constructed in the game engine. The hierarchy starts with the base as the root since it anchors the robot arm in place. Thus, it is the parent of all the parts. Then, the hierarchy descends with each joint as a node such that the movement of a joint involves relative movement of its children. The movement of the robot arm is through the rotations of the joints. To manipulate the EE of the arm to reach a certain point in space, inverse kinematic techniques must be applied to direct the rotation and displacement of each joint such that the EE can reach its desired position in the virtual environment. To apply the logic and mathematics involved in utilizing inverse kinematics, an algorithm must be constructed to describe the behavior of system that solves the inverse kinematics objective function. The algorithm is written in C# script that is attached to the UR10e model in the game engine to integrate it into the arm.

Inverse kinematics describes how the joints of the arm will orient themselves based on where the EE is positioned [9]. There are two inverse kinematics algorithm that are tested for the project. The method that was initially used is called FABRIK which is Forward and Backward Reaching Inverse Kinematics [1]. It iteratively performs an approximation of the new coordinates of the joints to reach the target.

Indicated at the top image of [Figure 6] is the backward process of the algorithm. The target indicates the desired location of the EE. For each joint, the new coordinates are determined by finding the direction of which it must move towards while maintaining the length between joints. It starts with the joint preceding the EE which is P_3. For instance, P_3 must move towards the direction of the vector that points towards the target. The distance between the target and P_3' is the original length between P_3 and EE. The next joint will move in the direction of P_3' and so on. The process ends with the base displaced from its original position.

The bottom image of [Figure 6] is the forward process. It uses the same method but starts with the joint after the base. P_1 is directed towards the vector from P_0' to P_1' formed by the backward process. The algorithm always ends with the forward process where the base is stable. The process iterates multiple times to improve the distance between the target and EE. However, at a certain point, the improvement of the solution is insignificant regardless of the number of iterations it will run through.

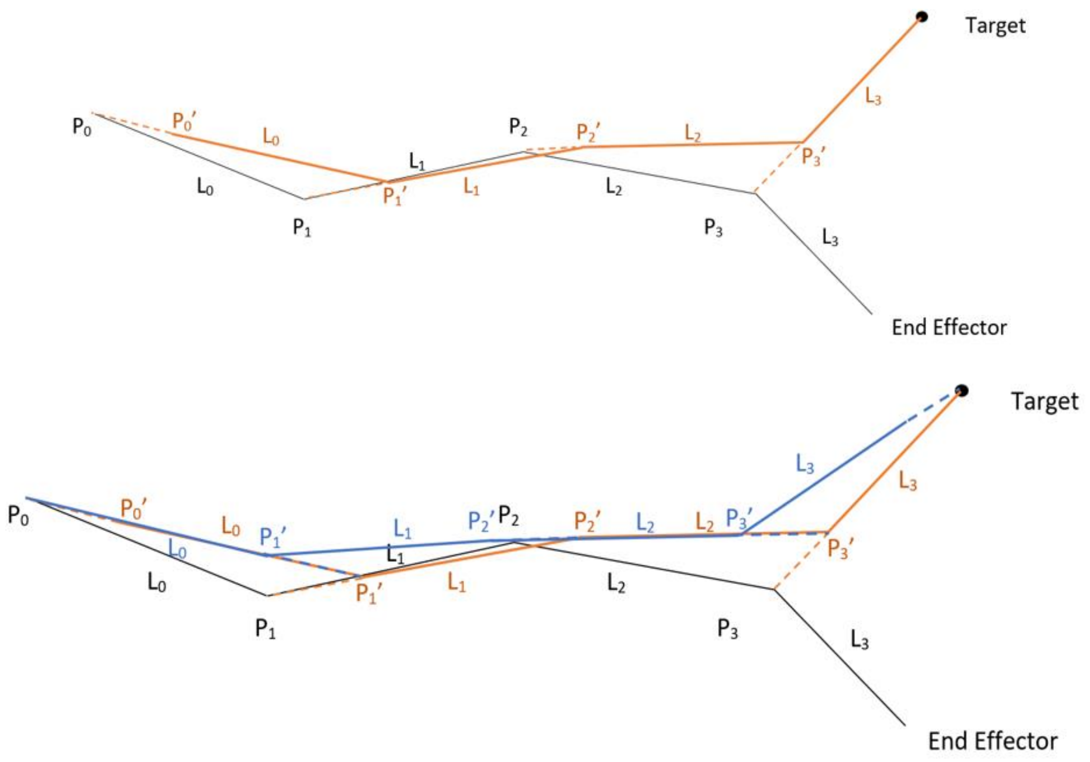


Figure 6. Forward and Backward Reaching Inverse Kinematics

There is a flaw in using the FABRIK algorithm while maintaining the constraints on the joints of the robot arm because it encounters gimbal lock. This restricts the movements of the arm and sometimes the best solution given by the algorithm is inaccurate which means the distance between the EE and target is still significantly large. Thus, when the solution renders onto the frame, the arm is at a drastically different orientation than it was in the previous frame.

An alternative algorithm is the gradient descent method used in solving optimization problems. The distance between the EE and the target is the function to be optimized by the gradient descent method. By reconfiguring the rotation of each joint, the EE should move closer to the target for each iteration of the algorithm. In other words, let each joint's rotation be, x, in the function, $\boldsymbol{f(x)}$, that defines the distance between EE and target. For each joint, i, the descent direction, $\Delta \boldsymbol{x_i}$, is determined by sampling neighboring values for the angle of rotation of the joint at a fixed sampling distance and calculating its estimated partial gradient [Equation 1] [9].

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x_i + sampling\_distance,\ x_{i+1,\ldots,}\ x_n) - f(x_i,\ x_{i+1,\ldots,}\ x_n)}{sampling\_distance}$$

Equation 1. Partial Gradient Approximation

In the example seen in [Figure 7], the descent direction is always opposite the gradient to minimize $\boldsymbol{f(x)}$. Thus, the descent direction is equal to the negative of the partial gradient [Equation 2] [9].

$$\Delta \boldsymbol{x_i} = -\frac{\partial \boldsymbol{f(x)}}{\partial \boldsymbol{x_i}}$$

Equation 2. Descent Direction

The step size is a multiple of the gradient vector and it determines the magnitude of which the angle value will traverse towards the descent direction. The step size used in the gradient descent method is a constant learning rate, L, which is fixed and predetermined. After calculating its partial gradient, each joint's angle, $\boldsymbol{x_i}$, will be updated for every iteration [Equation 3] [9].

$$\boldsymbol{x_i} = \boldsymbol{x_i} + (\boldsymbol{L} * \Delta \boldsymbol{x_i}) = \boldsymbol{x_i} + (\boldsymbol{L} * -\frac{\partial \boldsymbol{f(x)}}{\partial \boldsymbol{x_i}}).$$

Equation 3. Updating Angle of Rotation for ith Joint

The algorithm starts from the joint subsequent to the base so that it will take less iterations for the EE to be close enough to the target. This is because the larger rotations from the lower joints will close the distance between the EE and target more efficiently. The algorithm is written on a script that is attached to the EE in Unity to apply inverse kinematics to the UR10e robot arm. In addition, whenever the arm collides with its own parts during operation, the arm can untangle itself by sampling neighboring points in the opposite direction causing the joints to rotate opposite to where they were rotating just before the collision occurred. In this way, the arm can still reach the target using gradient descent without further colliding with itself or stopping the operation entirely.

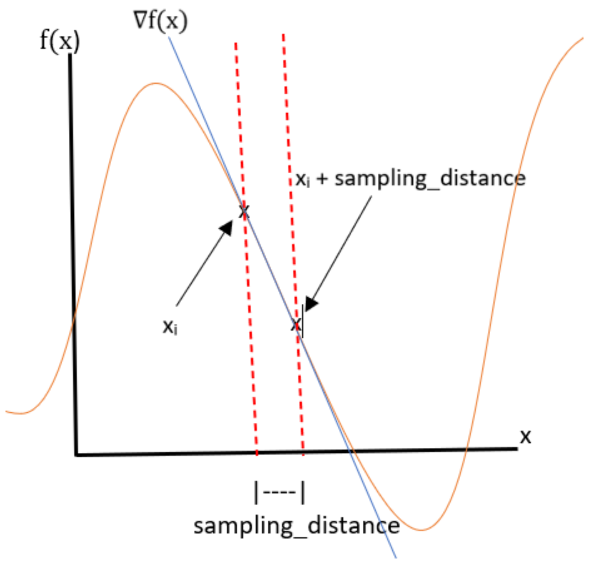


Figure 7. Example of Gradient Descent Method

After the inverse kinematics solver is incorporated onto the robot arm model, the arm can be controlled by moving the target which the EE follows. The target can be any game object such as a sphere, pointer or an empty game object that has no shape. To visually observe and troubleshoot the simulation in the game engine, the target is assigned to be a sphere. A control of the user's field of view (FOV) of the simulation must also be added in order to observe the movements of the robot arm in different angles. This is done by adding a script to manipulate the Main Camera which is the primary camera that renders the simulation onto the scene window during play mode in the game engine. The UI used in the Unity game engine is the Logitech Dual Action controller [Figure 8]. Three scripts are written to implement the UI with the analog controller. One script controls the movement of the target such that it can direct where the UR10e will reach. The left joystick and the up/down gamepad are used to move the target within the 3D space in the simulation. The other script controls the rotation of the camera's FOV in the simulation to enable the user to observe in different angles. The right joystick is used to control the FOV of the camera. Finally, the last script is used to trigger the grasping motion of the gripper tool attached to the tool flange of the robot arm. The gripper used in this project is the 2-fingered Robotiq gripper seen in [Figure 2]. The trigger is Button A. When it is pressed, the gripper will close its grip and opens when the button is released.

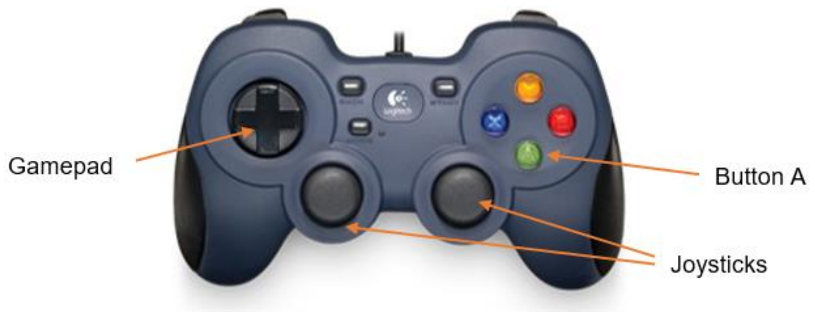


Figure 8. Analog Controller Used as User Interface in Unity

Finally, the simulation must be validated within the game engine before it is integrated onto a VR system. The validation is done by having the robot model perform a simple pick and place task. The robot arm must be able to interact with other game objects within the virtual environment. Objects with different shapes, sizes and weights must be involved in the validation task to test the capabilities of the virtual model of the UR10e. To verify the simulation, the robot arm model must be able to reflect the performance of the actual robot design in real-world scenarios.

VR Integration

Due to unforeseen events surrounding the COVID-19 virus in spring 2020, this subsystem is not completed at the time of publication for this URS thesis.

In this subsystem, the user interface used in the virtual system constructed in the game engine subsystem must be translated onto the VR controllers and the tridimensional simulation must be displayed on the HMD through Unity API which targets several VR devices [6]. The simulation constructed in Unity will be translated onto the VR device using Unity's OpenVR package that enables VR integration in the game engine. The simulation projected in the HMD must also incorporate the video images from the camera that is directed to the actual robot's perspective FOV. The latency of the feedback from the camera must satisfy the minimum requirement to avoid disrupting the cohesion of the simulation with the actual environment such that the operator can still have an immersive experience in the VR environment. After the simulation is integrated into the VR device, the control of the actual robot must be validated. The VR system can be validated by repeating the pick and place task executed in the game engine. The graduate research supervising this research will use HTC Vive VR device, so it is the VR system utilized in the project.

CHAPTER III

SYSTEM CHARACTERISTICS

Functional / Performance Requirements

Design Data Consistency

The must be zero loss of data from importing the CAD model of the UR10e arm into the Unity software such that the scale, shape, and DOFs of the UR10e still remain unchanged. The main purpose of the project is to accurately depict the UR10e robot in a virtual simulation such that it can be trained and tested entirely in the simulation.

Accurate Representation of Motion Behavior

The speed at which the virtual robot model performs must reflect the torque values at each joint of the actual model such that the capabilities of the virtual robot are consistent with the actual design. The performance of the virtual robot must be consistent with the actual model.

Physical Characteristics

Max Payload

The maximum payload that the UR10e arm model can sustain is 10 kg [5]. The UR10e can only carry load less than 10 kg. To avoid malfunction of the actual design, detecting weight of objects is necessary when testing the VR simulation.

Max Reach

The maximum reach of the UR10e model is 1300 mm with all 6 DOFs [5]. Each joint of the UR10e robot arm enables two parts linked by the joint to rotate 360^{o} around the joint's fixed axis [5]. This is a constraint of the actual UR10e design and defines its reach radius.

Weight

The UR10e Robot weighs approximately 33.5 kg with a maximum capacity payload of 10 kg. Thus, the maximum weight of the whole device is estimated to be 43.5 kg. The programmed speed at which the robot arm performs should be adjusted to maintain stability of the actual robot during operation.

Mounting

The weight of the UR10e robot arm is around 33.5 kg [5] and its maximum payload is 10 kg. The robot must be mounted on a stable base that can sufficiently support at least 43.5 kg weight.

Failure Propagation

The simulation in Unity should incorporate an algorithm that stops execution of the simulation in case of connection failure between the VR system and the UR10e robot. This is a safety precaution for testing the VR simulation.

Physical Interface

Video Interfaces

The HTC Vive HMD connects to a computer where the simulation runs via a 3-in-1 cable that is composed of the HDMI, USB and power cables. The display of the positions of the joints of the robot from cameras attached to the UR10e robot will be fed back to the Unity program via ethernet. USB cables will be used to connect cameras onto the computer running the program.

User Control Interface

HTC Vive VR wireless controllers will be used as UI for the UR10e robot arm. The controllers indirectly control the actual robot through the VR system. To validate simulation in the Unity game engine, another UI is needed to manipulate the UR10e robot arm. In this case, an analog controller is used. The joysticks are used to control the movement of the target, which directs the movements of the UR10e, in 3D space. The analog controller can then be translated onto the VR controllers and the HMD in the VR subsystem.

HMD HTC VIVE

The HMD possesses 1080 x 1200 pixels per eye resolution, 90 Hz refresh rate and 110^{o} FOV [7]. The immersive tridimensional environment of VR technology has high-end visual standards that satisfy the minimum requirements to have an engaging proprioception [4] for operators to immerse into the virtual environment.

Virtual Interface

Importing CAD Designs to Game Engine

The CAD parts must be assembled into one CAD file. The CAD file must be converted into a Unity compatible file format such as FBX or OBJ. This can be done by using Fusion 360, another CAD software other than SolidWorks. Fusion 360 can export the CAD file as an OBJ or FBX file. That file can be imported to Unity as a game engine asset (see Appendix in page 36).

VR Integration of Simulation

XR settings in Unity game engine provides compatibility with VR devices by enabling VR supported applications or VR SDKs for the simulation program. The Unity XR API can interact with multiple VR systems including the HTC Vive without any external plug-ins. Enabling XR settings allows automatic rendering of the main camera with virtual cameras as children set up in the simulation onto the HMD.

UR10e Visual Feedback

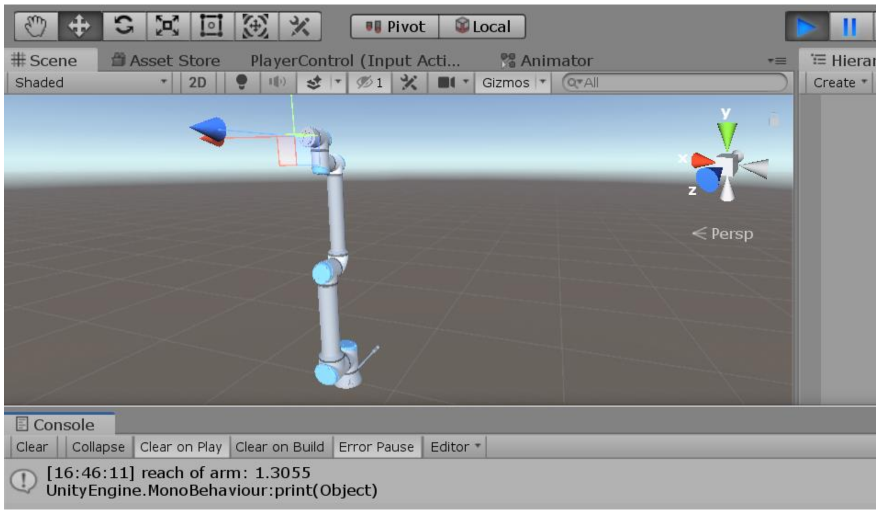
Visuals of the positions of the UR10e joints are provided by cameras attached to the robot arm. The user will then move the virtual robot model based on the reference points provided by the cameras.

CHAPTER IV

RESULTS

Reach Validation

The reach of the arm is determined in the game engine by measuring the height of the robotic arm from its base to its EE at its rest position where the arm stands upright and there are no rotations of all the joints as seen in [Figure 9].



```
[16:46:11] reach of arm: 1.3055
UnityEngine.MonoBehaviour:print(Object)
```

Figure 9. Rest Position of UR10e with Reach Measurement

In Unity, one unit of measurement for distance is equivalent to 1 meter. According to the specifications of the actual UR10e design, the reach of the arm should be 1.3 meters. Thus, the scale of the virtual design is accurate since the reach of the arm is approximately 1.3 units.

The validation of the arm’s reach is conducted by having an array of waypoints within its reach and positioning the target to each of these points until the arm reaches all the points within the array. The target is set to the next waypoint once the arm has successfully reached the target's current position. The next waypoint is chosen randomly to observe the arm's transition between two random points such that it can also test the flexibility of the arm contorting to different orientations of its joints. The waypoints generated 0.1 meters apart by incrementing 0.1 meters from the origin to the arm's reach, which is 1.3 meters, in each of the three axes, x, y and z. The total number of waypoints generated is 9126. Points that are within the base and shoulder of the arm are excluded since the base is immobile and the arm cannot penetrate its own parts to reach these points. The test is successful if the arm reaches all the points. To visualize points reached by the arm, spheres are instantiated and positioned to the points that the arm reached.

Results and Adjustments

During this validation, there are numerous times that the arm collided with itself to reach the target. During the collision, the arm stops operation momentarily and starts moving in the opposite direction to reach the target. Regardless of learning rate and iterations of the gradient descent algorithm to solve the inverse kinematics of arm, the arm initially moves relatively slower when reaching the target in the opposite direction due to a collision with itself. These collisions of the arm are significantly due to the rotation of the arm's elbow which can swing the upper part of the arm towards the shoulder and base of the arm.

The gradient descent attributes that can influence the performance of the arm are the learning rate, iterations, and distance threshold. In this validation, the arm's speed is mainly influenced by the learning rate and the number of iterations in running the algorithm. The learning rate is the magnitude at which a joint rotates from one iteration of the gradient descent algorithm. The number of iterations is how many times the algorithm runs before rendering the solution on each frame. Although a higher learning rate and a greater number of iterations allow the arm to react quicker from the movement of the target, these will cause the arm to jitter when it encounters collision with itself. A higher learning rate will cause an overshoot of the arm in approaching the target. In addition, a higher number of iterations will cause the joints of arm to rotate more in each frame such that there is a greater chance of a late detection in triggering an event of a collision with its own parts. Therefore, the late detection and the overshoot will cause the arm to jitter when colliding with itself. Along with finding the optimal learning rate and number of iterations, another solution is to immediately exit the algorithm and render the latest solution when the arm is close enough to the target. This is done by adding the distance threshold parameter in the algorithm. This is the set distance between the EE and the target that the algorithm must achieve in order to stop solving for another solution. This prevented the arm to overshoot oscillating from one solution to another. The optimal learning rate and number of iterations that were determined by trial and error are 50 and 10 respectively. After this validation, the resulting graphical representation of all the points reached by the arm is seen in [Figure 10].

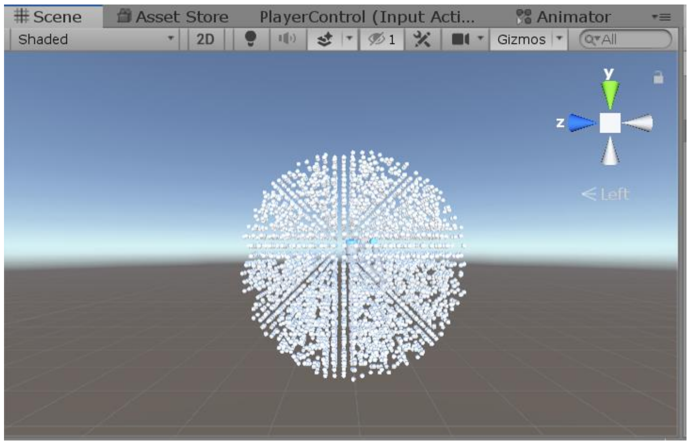


Figure 10. Validation of 9126 Points Within Reach of Arm

Pick and Place Validation

The next validation of the Unity simulation is testing the arm's pick and place performance. To perform the pick and place task, the arm must be able to grasp an object and place it to a specified location in the tridimensional space within the game engine while maintaining control of the object. The pick and place tasks are performed with user control through an analog controller and with automated control where the robot performs the task with no aid from the user. There are three subtasks for the automated control of the arm that were performed successfully in the game engine.

Automated Control

Subtask 1

In the first subtask, there are six cubes placed on the edge of a platform and the arm is directed to move each of the cubes forward half a meter away from the edge. The half meter line is represented by a red line on the platform seen in [Figure 11].

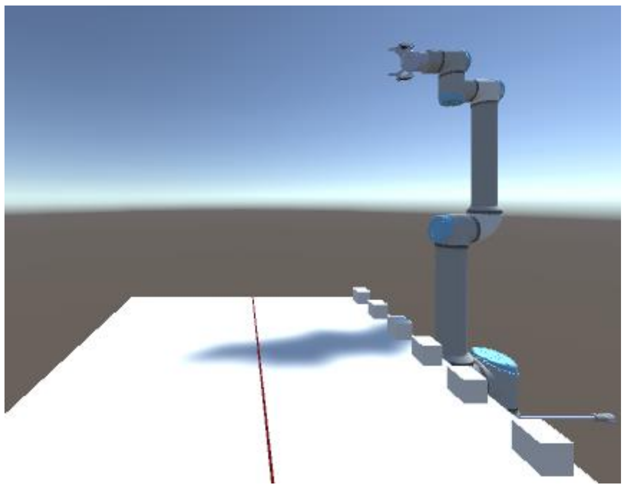


Figure 11. Half Meter Line on Platform with Cubes in Their Initial Positions

The automation of the arm is done by storing the positions of all the objects in an array and their respective destinations in another array. Then, the target is directed to the position of each object and it is then moved to the object’s designated location. This process iterates until the target has traversed through each member of both arrays. The grasping ability of the twofingered gripper tool is due to two types of colliders on both fingers of the gripper [Figure 12]. The mesh colliders of the two fingers of the gripper allows them to collide with other game objects. However, it seems that the game engine does not allow objects to be crushed or deformed by forcing two colliders to pin it down. Thus, the gripper is not able to grip an object in the game engine using only the mesh colliders of its two fingers. The result is that the two fingers of the gripper will pass through the game object that it is gripping as if they are in event trigger mode. Adding a rigidbody to each finger of the gripper to provide solidity to the fingers does not solve the problem. The result of adding rigidbodies is that the two fingers will break apart from gripping a game object regardless of the mass set for both fingers' rigidbodies. The solution that provided the optimal result is to add a box collider on the surface of each finger. It is difficult to control how much force the gripper applies to a game object when gripping so the box colliders will serve as a trigger to stop the two fingers from clamping down on a game object when it comes in contact with the surfaces of the fingers.

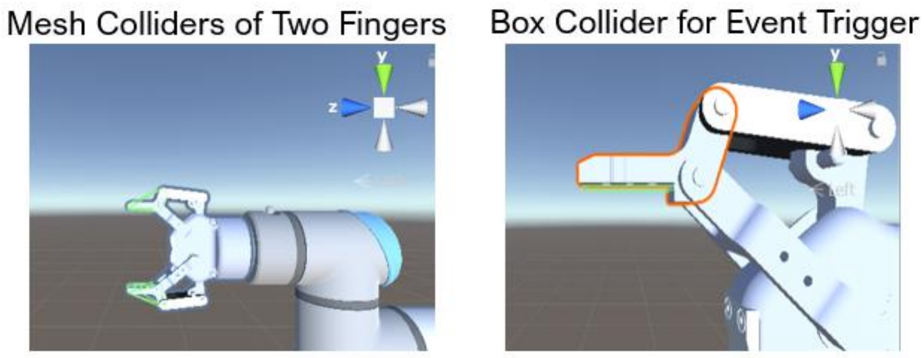
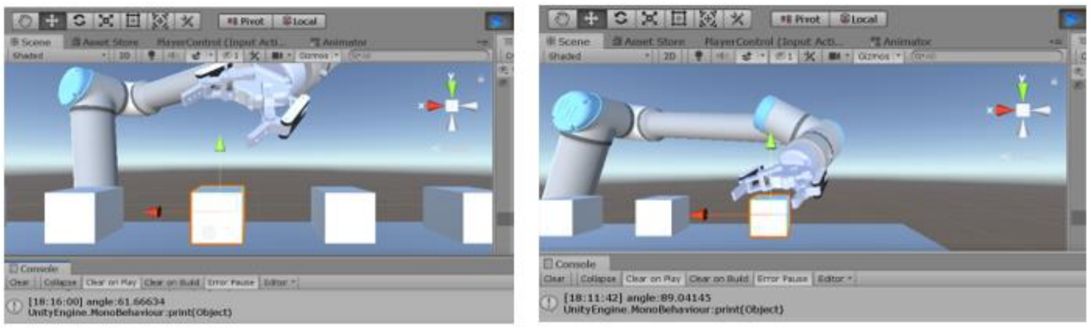


Figure 12. Two-Finger Gripper with Two Types of Colliders

Once the event of a collision from a game object is triggered by both box colliders, the game object that is clamped down will become a child of the gripper such that it inherits the displacement of the gripper. The gravity effect on the game object is also disabled for it to follow the gripper. Thus, the game object will be pinned down between the two fingers. However, the game object's physical contact with the mesh colliders of the fingers is still effective. Therefore, the game object can bounce, slip and fall from the grasp of the two fingers due to its interaction with the mesh colliders.

Results and Adjustments

The gripper needs to be in position to grasp the object as it approaches the item. Therefore, the gripper hovers directly above the object to position itself before grasping. The rotation of the gripper tool corresponds to the rotation of the target. Thus, the target rotates until the gripper is in position to grasp the object. The gripper is in position when the two fingers are horizontally aligned with the object it hovers. [Figure 13] illustrates the proper positioning of the gripper. The angle between the object's vertical vector and the horizontal vector between the two fingers must be less than 95^o and more than 85^o. This is the threshold in which the gripper must rotate to achieve proper positioning to grasp the object.



```
[18:16:00] angle:61.66634
UnityEngine.MonoBehaviour:print(Object)
```

```
[18:11:42] angle:89.04145
UnityEngine.MonoBehaviour:print(Object)
```

Not in Proper Position In Proper Position

Figure 13. Proper Positioning of Gripper When Grasping Objects

Once the gripper is in position, it can have a better grasp on the object and maintain control of it during the process of pick and place. However, there are still cases where the objects slip and fall from the gripper's grasp. The solution to this is by adding a physic material to the game objects such that they can have set friction and firmness. Setting more friction allows the object to have a rougher texture. The firmness of the object is the bounce setting of the physic material. Lesser bounce allows the object to be more rigid. By adjusting the dynamic and static friction of the object to 0.8 out of 1 and setting the bounce to 0, the gripper can maintain control of the objects throughout the pick and place task. [Figure 14] displays the image when the arm has successfully performed the subtask.

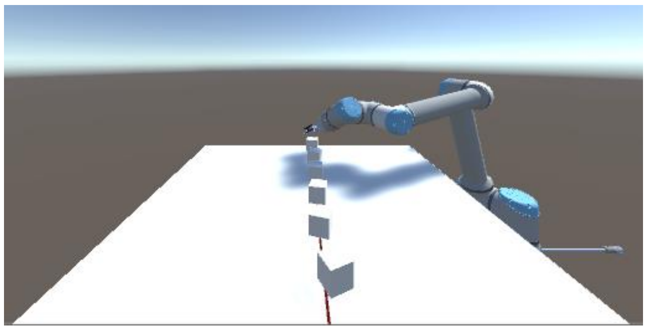


Figure 14. Validation of Subtask 1

In addition, [Table 1] shows the success rate of the robot arm in performing the subtask by tabulating its success or failure in each of the 10 trials.

Table 1. Subtask 1 Success Rate of 80% from 10 Consecutive Trials

Trials	Cubes Moved to Half Meter	Successful Trial
1	6	Yes
2	6	Yes
3	6	Yes
4	5	No
5	6	Yes
6	6	Yes
7	6	Yes
8	5	No
9	6	Yes
10	6	Yes

Subtask 2

The next subtask is to stack the six cubes on top of each other on a platform. In this subtask, the robot model is programmed to take the positions of all the cubes initially and randomly chooses which object to pick. The first object chosen is placed onto a set location on the platform. The subsequent object that is chosen is placed on the same location. However, for each subsequent object chosen, the height of its destination increments from the previous object placed. The subtask is completed once all six cubes are stacked up together and remain stable.

Results and Adjustments

There are cases when the gripper tips over the stacked cubes by moving directly towards the next object to pick up. Therefore, the gripper tool is slowly lifted for every object it picks up and places down. This will avoid collisions with other game objects on the platform during the pick and place process. Another adjustment is the height at which each object is placed on top of the stack. When the gripper places the object on top of the stack, it must gradually lower the object until it can be properly placed on the stack and remain stable. The gripper must avoid dropping the object on stack to prevent it from knocking over the stack. In this validation, the highest number of cubes that are stacked successfully is 5. With the sixth cube, the stack topples over due to the added weight. The cubes are stacked in varying positions since the gripper grasps the objects in different positions. Therefore, the validation is adjusted. The arm successfully completed the pick and place task if it can stack 5 cubes or more. [Figure 15] displays the image when the arm has successfully stacked 5 cubes on top of each other on a platform. In addition, [Table 2] shows the success rate of the robot arm in stacking at least 5 cubes by tabulating its success or failure in each of the 10 trials.

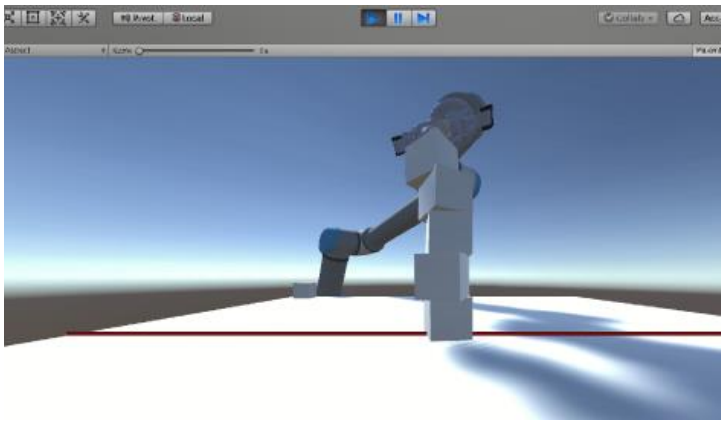


Figure 15. Validation of Subtask 2

Table 2. Subtask 2 Success Rate of 90% from 10 Consecutive Trials

Trials	Cubes Stacked	Successful Trial
1	5	Yes
2	5	Yes
3	5	Yes
4	5	Yes
5	4	No
6	5	Yes
7	5	Yes
8	5	Yes
9	5	Yes
10	5	Yes

Subtask 3

The third subtask involves limited aid from user to simulate the human robot collaboration (HRC). The subtask requires the arm to grasp and pick up the object specified by the user while maintaining control of it. Then, the arm must be able to place the object to the location specified by the user on the platform without dropping the it. The user specifies the object and its destination by using the mouse pointer.

Results and Adjustments

The positioning of the gripper to properly grasp an object varies. Therefore, some objects may not be properly grasped by the gripper because these objects may slip from the grasp of the gripper or the gripper misses the objects without proper positioning. To mitigate this issue, the speed of the arm is minimized to gain more accuracy when grasping an object. In addition, the friction of objects and the gripper is adjusted through the physic materials assigned to them. [Figure 16] displays images of the robot arm performing a pick and place task with a cube. The leftmost image shows the robot directed by the arrow pointer to pick up a cube. The rightmost image shows the robot arm directed by the arrow pointer to place the cube to the specified location. In addition, [Table 3] shows the success rate of the robot arm in performing this subtask for 10 trials by tabulating its success or failure in each trial with a specified object.

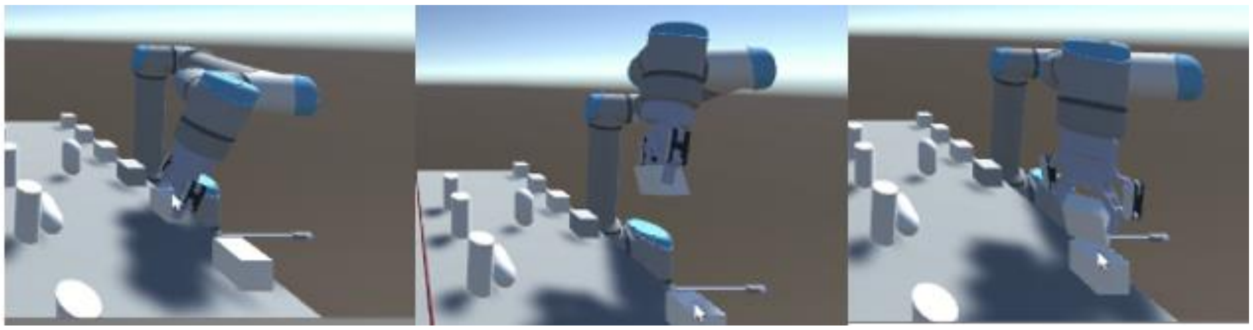


Figure 16. Validation of Subtask 3

Table 3. Subtask 3 Success Rate of 70% from 10 Consecutive Trials

Trials	Object Picked and Placed	Successful Trial
1	cube	Yes
2	cylinder	Yes
3	capsule	Yes
4	cylinder	No
5	cube	Yes
6	sphere	Yes
7	cylinder	Yes
8	capsule	No
9	sphere	No
10	capsule	Yes

Manual Operation

The validation for the user control of the arm in the game engine is essential during training and testing of cobots like the UR10e. This validation is similar to the third subtask for the validation of the automated control of the arm. To validate the user control, the arm must be able to grasp a certain object and place it to a specified location on a platform through the use of an analog controller.

Results and Adjustments

The joystick sensitivity of the control causes inaccuracies in operating the arm it is adjusted through input settings within the game engine. Moreover, the gripping of objects is more adjustable resulting in a higher success rate of grasping the items than with automated control. [Figure 17] provides two images of the robot arm performing a pick and place task with a cube. The left image shows the robot manually controlled by the user to grasp a cube. The right image shows the robot directed by the user to pick up the object and moving it to the location chosen by the user .

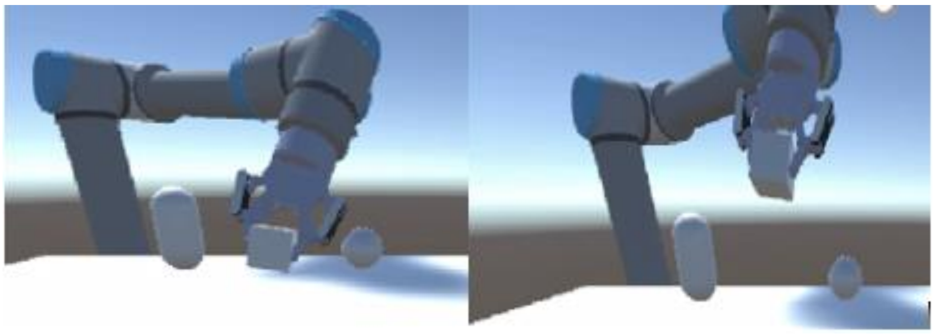


Figure 17. Validation of Manual Control

Furthermore, [Table 4] shows the success rate of the robot arm in performing this subtask for 10 trials by tabulating its success or failure in each trial with a specified object.

Table 4. Manual Control Success Rate of 60% from 10 Consecutive Trials

Trials	Objects Picked and Placed	Successful Trial
1	cube	Yes
2	cube	Yes
3	sphere	No
4	cylinder	Yes
5	cube	Yes
6	capsule	No
7	sphere	Yes
8	cylinder	Yes
9	capsule	No
10	sphere	No

VR Integration and Operation

Due to unforeseen events surrounding the COVID-19 virus in spring 2020, this validation is not completed at the time of publication for this URS thesis.

CHAPTER V

CONCLUSION

The aim of the project is to provide a system for VR teleoperation of the UR10e robot arm such that an operator can control and observe the robotic design remotely through a VR device. The project can later be used in reinforcement learning applications for further research on automation. Although the VR integration is not completed, the virtual simulation in the game engine can still provide a software platform to aid reinforcement learning implementations. However, there are still further steps to be done to integrate the simulation onto a VR system. The visual graphics of the simulation must be up to par with the quality required for the VR system. This can be done using Unity's post-processing tools to improve the visual quality of the simulation. Unity's OpenVR package can provide the necessary tools to translate the simulation controls and visuals onto the HTC VIVE device which is the intended VR system that would be used in the project.

There are several procedures and components that could be improved or added to enhance the productivity of the research. For troubleshooting in the game engine, a graphical user interface (GUI) can be added onto the Main Camera during simulation such that sliders and buttons are organized onto the screen. Sliders can be used to control the rotation of each joint of the robot arm to test the DOFs with the constraints. A button to disable and enable inverse kinematics algorithm can also be incorporated to test conditions with and without the inverse kinematics. There may also be a small window projecting the rotation angles and transforms of each joint during simulation to aid in debugging the system.

In addition, complex procedures could have been partitioned to ease debugging since it is difficult to debug larger, more complicated components. For example, the FABRIK algorithm used in the research runs under the constraints imposed on the joints to reflect the degrees of freedom of the actual robotic design. A problem occurred where the arm cannot reach the target due to gimbal lock where rotation axes are positioned parallel to each other. The joint constraints added in the CAD software did not transfer to Unity. Thus, the constraints were added within the game engine by attaching scripts to each of the joints. These scripts running concurrently with other components in the simulation makes it challenging to debug the problem. It may have been easier to separate the joint constraints to another subsystem where it would be validated separately.

REFERENCES

[1] Aristidou, A., & Lasenby, J. (2011). FABRIK: A Fast, Iterative Solver for the Inverse Kinematics Problem. *Graphical Models*, *73*(5), 243-260.

[2] De Giorgio, A., Romero, M., Onori, M., & Wang, L. (2017). Human-Machine Collaboration in Virtual Reality for Adaptive Production Engineering. *Procedia Manufacturing*, *11*, 1279-1287.

[3] Lees, Ü., R. Hudjakov, and M. Tamre (2014). Development of Virtual Reality Interface for Remote Robot Control. *Tallinn University of Technology*.

[4] Pérez, L., Diez, E., Usamentiaga, R., & García, D. F. (2019). Industrial Robot Control and Operator Training Using Virtual Reality Interfaces. *Computers in Industry*, *109*, 114-120.

[5] The UR10e Collaborative Industrial Robot. (2020). *Universal Robots*.

[6] Unity User Manual (2019.2). (2019). *Unity Documentation*.

[7] Vive VR System. (2020). *Vive*.

[8] Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K., & Abbeel, P. (2018, May). Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation. In 2018 IEEE International Conference on Robotics and Automation (ICRA) (pp. 1-8). IEEE.

[9] Zucconi, A. (2017, April 10). Inverse Kinematics for Robotic Arms.

APPENDIX

Collaborative Robots	Industrial robotic systems, often referred to as cobots, that interact closely with human operators to be guided and supervised by them. Complex tasks done meticulously by humans can only done accurately and efficiently by semi-automated machines supervised by humans.
Game Engine Asset	Figures, scripts, and other tools/materials that are used, modified and adjusted within the game engine. These can be instantiated along with its properties within the game engine.
Game Object	A figure/object that is an asset in the game engine. A single game object can be composed of other game objects that are referred to as its children.
Inverse Kinematics	A mathematical process that is usually applied in robotics to calculate the joint orientations of a kinematic chain in order to position the end-effector to a desired point relative to the base of the chain.