DEVELOPMENT OF VIRTUAL SIMULATION FOR EVALUATION OF ROBOTIC ASSISTIVE ENVIRONMENT

by

SHUBHAM RAOSAHEB GUNJAL

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2021

To my parents

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Panos Shiakolas for constantly motivating and encouraging me, and also for his invaluable advice during the course of my research studies.

This thesis is copyrighted by Shubham Gunjal. However, this research would have never been started and completed without the guidance, advice and direction of Prof. Shiakolas. As such, I provide Prof. Shiakolas unequivocal rights to use this research and results for any purpose.

I also thank my academic advisor Dr. Seiichi Nomura, and Dr. Ioannis Schizas for their interest in my research and for taking time to serve on the thesis committee.

Thank you to members of MARS Lab, Tushar Saini, Abdul Rahaman, Shashank Kumat, Shane Whitaker for their help.

Thank you to my parents, Mandakini and Raosaheb, my sisters Pooja and Pranjali, and my partner Whitney for their support along this journey.

May 11, 2021

ABSTRACT

DEVELOPMENT OF VIRTUAL SIMULATION FOR EVALUATION OF ROBOTIC ASSISTIVE ENVIRONMENT

SHUBHAM RAOSAHEB GUNJAL, M.S.

The University of Texas at Arlington, 2021

Supervising Professor: Dr. Panos Shiakolas

Robotics research in virtual simulation has the advantage of reduced testing time, verification of algorithms on different systems before implementation, and cost-saving. Today's simulation softwares are capable of providing the physics of the simulation such as torque requirements of the joints which aid in the selection of appropriate hardware.

In this research, Webots, an open-source physics-based simulation, and visualization software is evaluated and then used to study and evaluate the performance of two robotic systems for a robotic assistive environment for persons with upper limb disabilities. This includes a wheelchair, a table with objects, and two robotic arms attached to the wheelchair. The two arms are a custom-developed 4-degrees of freedom arm and a UR5e arm from Universal Robotics. This assistive environment also includes three cameras to identify objects to be manipulated with object geometry and grasping specific information defined and extracted from a metadata file. The motion controller code is written in MATLAB using the Robotics Systems Toolbox.

An algorithm that accepts keyboard input to execute the desired motion has been developed.

The environment allowed us to successfully study multiple scenarios for interaction and object grasping and provided the tools for visualization as well as detailed information and characteristics of the motion including the occurrence of collisions. After verifying the performance in simulation, the inverse-kinematics solution was stored for later analysis. The goal of using these virtual tools is to reduce the testing and analysis time for robotics projects and fine-tune them before implementation. The success of this research provides the confidence to implement the environment in the research lab.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This chapter contains an introduction and some background of this research in Human-Robot Interaction (HRI), also known as collaborative robotics. HRI is introduced with examples of some of its applications. Preceding research and the need for virtual simulation is explained. This chapter concludes with the outline of this thesis report.

1.1 Introduction to Human-Robot Interaction

HRI is an engineering field involving the design and implementation of robotic systems that interact with a human in different ways and are controlled by humans by various processes such as Electroencephalography (EEG) signals, voice commands, facial expressions, remotes, etc. The interaction can be classified as remote interaction and proximate interaction. The remote interaction involves control of robots from spatial and/or temporal distance such as Mars rovers. The proximate interaction involves the control of robots which are located in the same environment as the human. [2]

HRI is an emerging field of robotics with research currently been carried out in various aspects of it such as artificial intelligence [3] [4], teleoperation of robots [5] [6], and soft robotics [7] [8]. HRI has found applications in numerous fields such as healthcare, mining and rescue, and space exploration. An example of HRI in space is the use of the Canadarm aboard the International Space Station which is remotely operated from inside the station to move the payload. Industrial robots are widely

used for mass productions such as cars. However, for a small production volume, the problem of the high cost of integration and reprogramming arises. These costs can be reduced by the implementation of a HRI system which would eliminate the need for reprogramming and the need for highly trained personnel for small-scale production. In healthcare, HRI can be used to assist a disabled person such as upper body disabled person.[9] Agriculture is an emerging area for HRI with an application such as target recognition and spraying tasks.[10]

1.2 Overview of preceding research

Prior research in the MicroManufacturing and Automation and Advanced Robotic Systems (MARS) Lab included the design and implementation of robotics hardware and controller for a Human-Robot Interaction (HRI) environment. A HRI system to command a Biomimetic Artificial Hand (BAH) by cloud-assisted voice command and a safety vision system was researched. This research concluded that the proposed interaction modality can be successfully employed to communicate with a robot.[11] Another HRI system was tested on a Robotic Prosthetic Hand (RPH) as a testbed for sensor glove-based tele-control using an artificial neural network.[12] Another research focused on the development of a human thumb tracking device for telemanipulation. This research was implemented on a developed sensor device and flex-glove and was successful in the determination of human thumb motion and control of the artificial robotic thumb.[13] Another HRI research project was a wheelchair assisted by a robotic arm. This research aimed to assist a human with an upper body disability. The hardware developed for this research is shown in figure 1.1.

Figure 1.1: Robotic arm assisted wheelchair

1.3 Need for virtual simulation

Performing HRI research on real hardware and robots provides good research data and output, it has some drawbacks. Such a research approach requires building a setup of the hardware before the actual tests could be performed. This generally is a time-consuming process and involves the cost of hardware. Any changes in the setup, if required, are difficult to make such as finding the correct power of the motors for joints. Another requirement of such a research approach is to do the work hands-on in the lab. Thus, it makes working remotely difficult or impossible.

A HRI research approach using virtual simulation overcomes all of the drawbacks mentioned above. A virtual simulation makes it easy to build an environment and edit it with any changes with significantly less time and less cost. The virtual simulation can provide the data and the output that is close to real hardware test, assuming the simulation parameters are well defined and mimic the real world. This research also provides information about the required hardware such as motor power, controller parameters, force, etc. Thus, conducting research in virtual space first can provide accurate requirements of the hardware, which then can be used to build a real setup for the research. And with virtual simulation, it is possible to perform the work remotely which proved beneficial during the Covid-19 pandemic.

1.4 Thesis outline

The structure of this thesis is as follows. Chapter 2 introduces the available simulations environments, and the environment used, Webots. It details the simulation environment modeled along with the definition of the objects and the matrix transformation relationships between various coordinate frames. Also discussed is the process of mapping objects from the Webots simulation environment to the MATLAB inverse kinematics environment. Chapter 3 consists of all the object manipulation processes such as pick-up, drop-off, bringing the object to the user, pick of the hidden objects. It also consists of the inverse kinematics constraints applied on the robotic arm and the physics of simulation. Chapter 4 gives the results, discussions and experimental implementation of the simulation using an Arduino setup. Chapter 5 provides conclusions of the research and recommendations for future research. The appendix consists of the main controller code and an image of Webots interface.

CHAPTER 2

SIMULATION ENVIRONMENT

This chapter gives a background of various virtual simulation packages available with some examples of their use in research and explains in detail the structure of Webots simulation, which is used in this research. The assistive wheelchair modeled, its robotic arm, and the object definition is discussed. The transformation relationships between the robotic arm, camera, human, and the objects are defined. In addition, the keyboard interaction modality to control the simulation and the mapping process is explained.

2.1 Background and literature review of virtual simulations

There are various simulation packages available to build a virtual simulation environment for robotics. Popular simulation software include Gazebo [14], Webots [15], and CoppeliaSim [16]. The overall structure of these simulation environments is similar with a few differences such as the type of physics engine, in-built robots, importing external robots, etc. Gazebo has multiple physics engines to perform dynamics of simulation which include Open Dynamics Engine (ODE), Bullet, Simbody, and Dynamic Animation and Robotics Toolkit (DART). CoppeliaSim also has multiple physics engines which include ODE, Bullet, Newton, and Vortex Dynamics. Webots only offers one physics engine in ODE. All these three simulation environments can be coded in multiple programming languages such as MATLAB, Python, Java, and C++.

These simulation packages have been used widely in academia and industry for research on robotic systems. They provide an economically efficient and time-saving way for testing out research. Zhang and Liu used Gazebo and ROS to compare the robustness of different controllers on 7 degrees of freedom robotic arm with the task of pick and place an object. [17] They successfully compared the performance of 5 types of controllers which included active inference controller, model reference adaptive controller, joint position controller, joint velocity controller, joint effort controller. In similar research, CoppeliaSim was used to simulate the physics of a 6 DoF robotic arm in an automation cell for reconditioning fan blades with grinding. [18] They were able to get the forces exerted on the arm by the griding process from the physics engine of CoppeliaSim. In an obstacle avoidance research project, the authors Ginesi, Meli, Roberti, Sansonetto, and Fiorini used CoppeliaSim to simulate obstacle avoidance in different scenarios such as industrial pick and place tasks, a mobile robot in a dynamic environment, and a surgical robot to show the scalability of their controller. [19] An Ocean Engineering research team used Webots to develop an economically effective virtual training environment for controlling underwater manipulator. [20] This research simulation was done in kinematics mode and studied various operational modes of the manipulator. Many similar research studies have been conducted for the operation and evaluation of robotic systems underwater. [21] [22]

This literature survey on the virtual simulations in robotics research shows that the desired robotics systems and processes in diverse environments can be tested and results with reliable accuracy can be obtained. Furthermore, The use of virtual simulation provides confidence for the implementation of these projects in the real world. In the MARS lab, CoppeliaSim and Webots were chosen for a project on the assistive robotic environment and to compare them for future use. This research in this thesis used Webots.

2.2 Introduction to Webots

Webots is a robotic simulation software that provides rapid prototyping of the environment. Users can create a simulation that may contain passive objects such as tables and active objects such as robots and sensors. An inbuilt library consists of various objects and robots that can be imported into the simulation directly. Custom-made objects and robots can be imported as a Virtual Reality Modeling Language (VRML) file. Available sensors and nodes that can be used to build the robotic environment include a camera, motors, distance sensors, hinge joints, GPS, etc. Multiple robots can be programmed individually in the same simulation.[15]

The controller code for Webots simulation may be written in a coding languages that include MATLAB, C, C++, Python, and Java. In this research, MATLAB was used for its inbuilt robotic systems toolbox. [15]

Webots also has an inbuilt physics engine which is based on Open Dynamics Engine(ODE)[23]. This allows the user to retrieve the results of the physics of the simulation such as forces and torques. [15] ODE is an open-source, high-performance library for simulating rigid body dynamics. ODE is used by many software packages for simulating a virtual environment.

2.3 Assistive wheelchair model

The assistive robotic environment modeled in this research is of the wheelchair shown in figure 1.1 with a custom-made 4 degrees of freedom (DoF) arm with 4 revolute joint motors. This custom-made arm was created in SOLIDWORKS and exported as a URDF [1] and later converted and imported into the simulation as a VRML file. This 4 DoF was created to be a cost-effective arm to attach to the wheelchair.

[1]URDF: Unified Robot Description Format

This arm has limitations with approaching objects of non-revolute geometries such as a square box from its side.

To overcome the limitations of the custom arm, a second robotic arm with 6 degrees of freedom, UR5e [24] was added to the wheelchair in the simulation as shown in figure 2.1. This arm has 6 revolute joint motors, base, shoulder, elbow, and 3 for wrists. Thus, adding this robotic arm to the assistive wheelchair provided the capability of approaching the objects of non-revolute geometries from the side of their edges. This arm has a reach of 850mm and can handle a payload of 5 kg which is ideal for the intended use of an assistive environment. The UR5e arm was imported from the inbuilt library of Webots.

All the physical properties of the arms such as mass, inertia are modeled as defined in the VRML file and the library of Webots. These properties are later used for retrieving the torque requirements for the joints.

The wheelchair also has 3 cameras attached for detection and location of the objects and the human face. The Overhead camera is the primary camera used to detecting and locating the objects on the table. The UR5e camera is used when the objects are hidden from the overhead camera and moves along with the arm. The side camera locates the human face for bringing the objects to the user.

2.4 Objects and their definition

Every object defined in Webots simulation has a frame assigned to it which is by default located at the geometric center of the object. The objects created within the simulation have by default their y-axis pointing upwards or along the length of the object for all the objects, indicated by green arrow. This default setting may be changed if required. Custom-made objects may have their frame at a different location and with the different axis pointing upwards or along the length of the

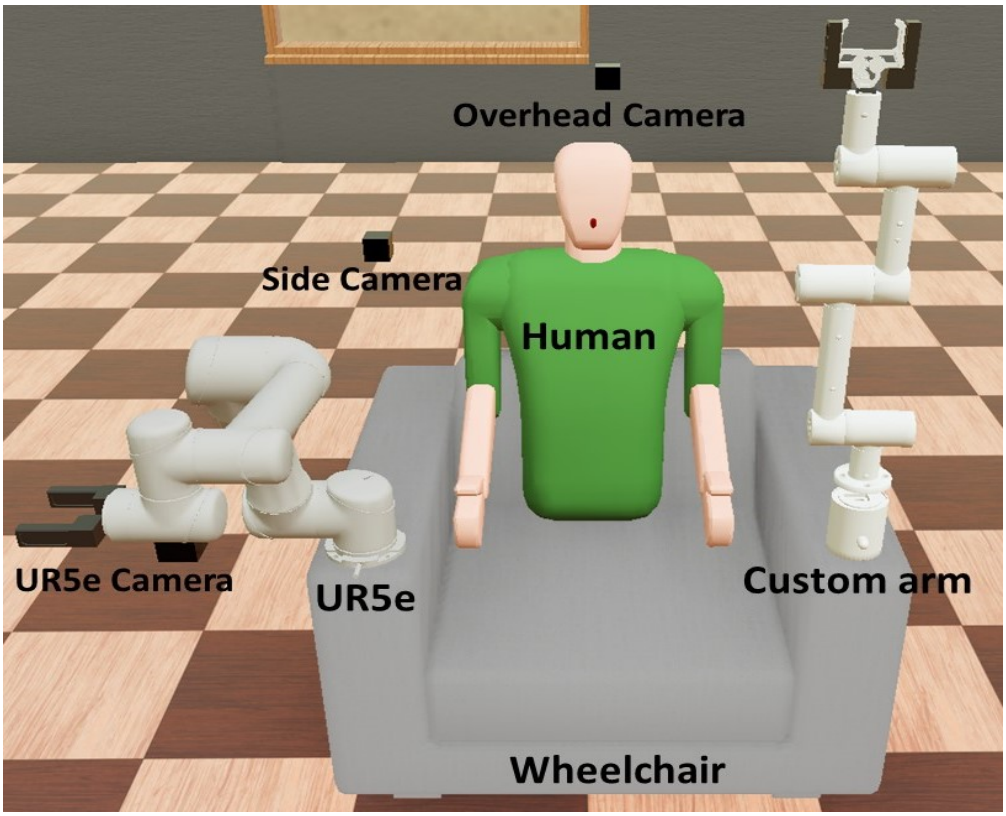


Figure 2.1: Assistive wheelchair

object. Figure 2.2 shows an object created within the Webots simulation with its frame. The position and orientation of objects are defined with respect to their frame and are used for approaching and picking up the object.

2.5 Hierarchy of nodes

The simulation environment of Webots is built on a hierarchical basis. Figure 2.3 shows the relationships of different nodes that form the assistive wheelchair. An arrow between the two nodes represents an inheritance relationship. A derived node at the arrowhead inherits all the field properties and functions of a base node at the arrow tail.[15]

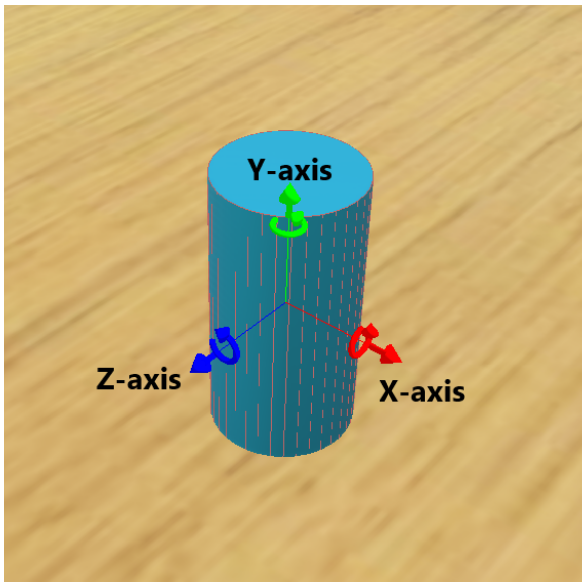


Figure 2.2: Object definition in simulation

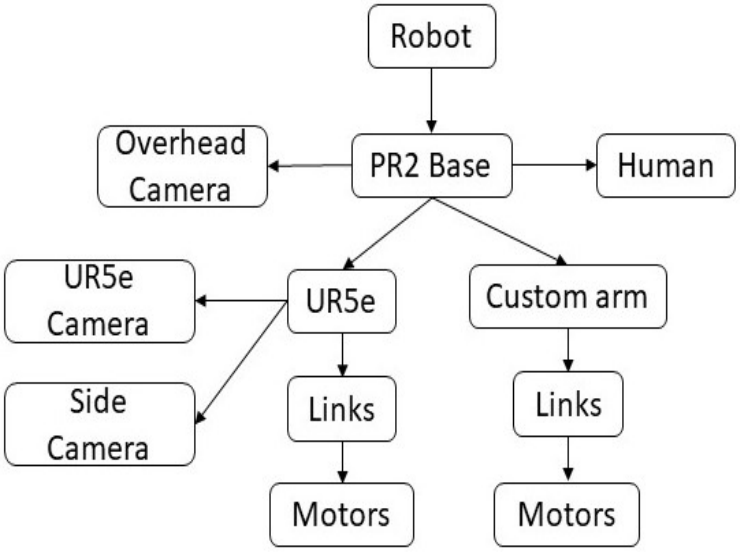


Figure 2.3: Hierarchical relationships

The build of the wheelchair starts with a *ROBOT* node. A controller code is written for just the *ROBOT* node that can control all its children nodes. The base frame of the PR2 robot is added as the base of the chair with wheels to move. The UR5e arm, the custom arm, overhead camera, human are made children of the PR2

Base. Both the arms have links and motor joints that form their body. UR5e arm has two more children as the attached camera and the side camera. In the PR2 base, both the arms are $SOLID$ nodes that can physically interact with the objects in the environment. The cameras and the human are just $SHAPE$ nodes that do not physically interact with the environment.

2.6 Transformation relationships

The position and orientation of an object are defined with respect to its frame by a transformation matrix given by equation 2.1, where R denotes the elements of the rotation matrix and P denotes the elements of the position vector. [25]

$$T = \begin{bmatrix} R_{xx} & R_{yx} & R_{zx} & P_x \\ R_{xy} & R_{yy} & R_{zy} & P_y \\ R_{xz} & R_{yz} & R_{zz} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

In equation 2.1 for the rotation matrix part, the first subscript denotes the axis to be projected and the second subscript denotes the axis projected upon. For the position vector, the subscripts denote the coordinate along the specified axis.

Figure 2.4 shows the use of the overhead camera for locating the objects on the table. The transformation of the overhead camera frame with respect to the robotic arm frame, ${}^{Arm}_{OCam}T$ indicated with a blue vector, is known and fixed. The camera finds and outputs the location of objects on the table, thus providing the transformation of the object frame with respect to the camera frame, ${}^{OCam}_{Obj}T$ [2] indicated with another blue vector. With these two transformations known, the relation given by equation 2.2, can be used for solving the position and orientation of the object, i.e. the transformation of the object frame with respect to the frame of the arm, ${}^{Arm}_{Obj}T$

[2] OCam: Overhead camera, Obj: Object

indicated with a red vector. This equation is used to locate objects visible to the overhead camera and later used to find inverse kinematics solutions.

$$ {}^{Arm}_{Obj}T = {}^{Arm}_{OCam}T \, {}^{OCam}_{Obj}T \tag{2.2} $$

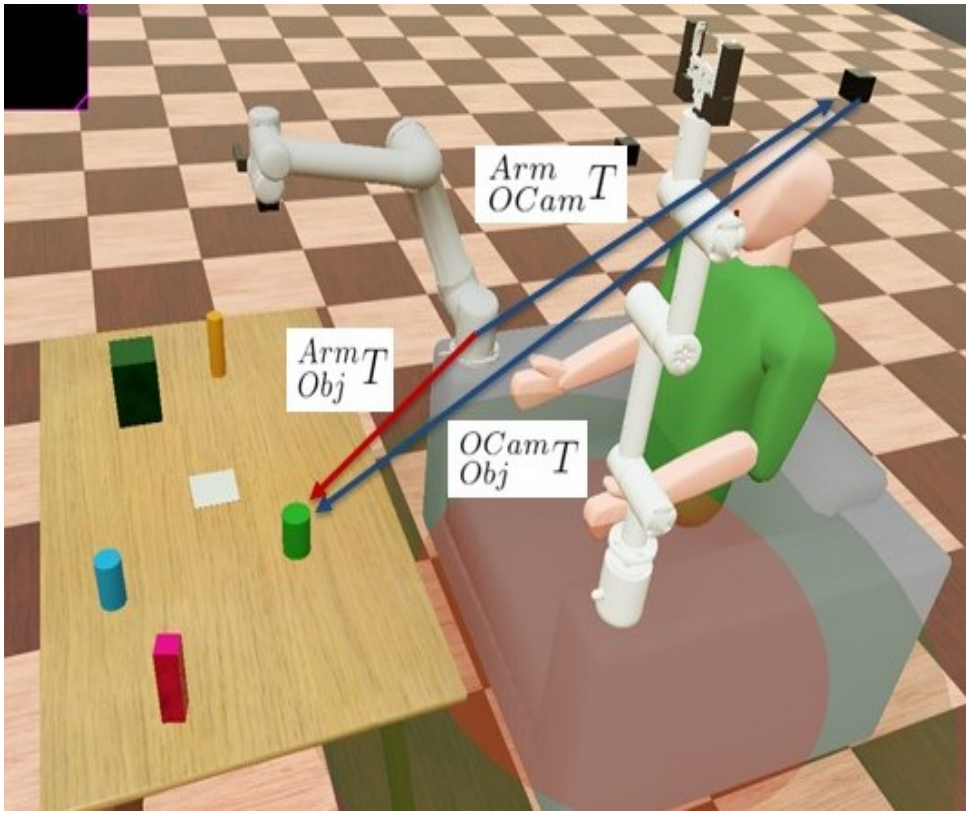


Figure 2.4: Transformation relationships of overhead camera

Figure 2.5 shows the use of the side camera and UR5e camera. The UR5e camera is primarily used for locating the objects which are hidden from the view of the overhead camera with respect to the robotic arm frame. In this figure, the transformation of the UR5e camera frame with respect to the robotic arm frame, ${}^{Arm}_{UCam}T$ indicated with a blue vector, is dynamic and changes as the arm move through

the environment. This camera outputs the position and orientation of the object, i.e. the transformation of the object frame with respect to the camera frame, ${}^{UCam}_{Obj}T$ indicated by a blue vector. With ${}^{Arm}_{UCam}T$ and ${}^{UCam}_{Obj}T$ known, equation 2.3 can be used for solving the transformation of the object frame with respect to the robotic arm frame, ${}^{Arm}_{Obj}T$ indicated with a red vector. The use of the UR5e camera for locating the hidden objects is explained in greater detail in chapter 3.

$$ {}^{Arm}_{Obj}T = {}^{Arm}_{UCam}T \, {}^{UCam}_{Obj}T \tag{2.3} $$

The side camera[3] is used for locating the human face with respect to the robotic arm frame. The transformation of the side camera frame with respect to the robotic arm frame, ${}^{Arm}_{SCam}T$ indicated with a blue vector is known and fixed. The side camera outputs the transformation of the human face frame with respect to its frame, ${}^{SCam}_{Face}T$ indicated with a blue vector. With ${}^{Arm}_{SCam}T$ and ${}^{SCam}_{Face}T$ known, equation 2.4 can be used for finding the transformation of the face frame with respect to the robotic arm frame, ${}^{Arm}_{Face}T$ indicated with the red vector.

$$ {}^{Arm}_{Face}T = {}^{Arm}_{SCam}T \, {}^{SCam}_{Face}T \tag{2.4} $$

2.7 Interaction modality

An interaction modality is a method of communicating and controlling the simulation as desired. Various modalities can control a simulation such as electroencephalogram (EEG) signals, voice commands, and manual control with hands-on instruments. As the first phase of virtual simulation research, this research used a computer keyboard to command the simulation. However, the simulation is coded in such a way that other interaction modalities can be easily implemented in the next phase of research.

[3]SCam: Side camera

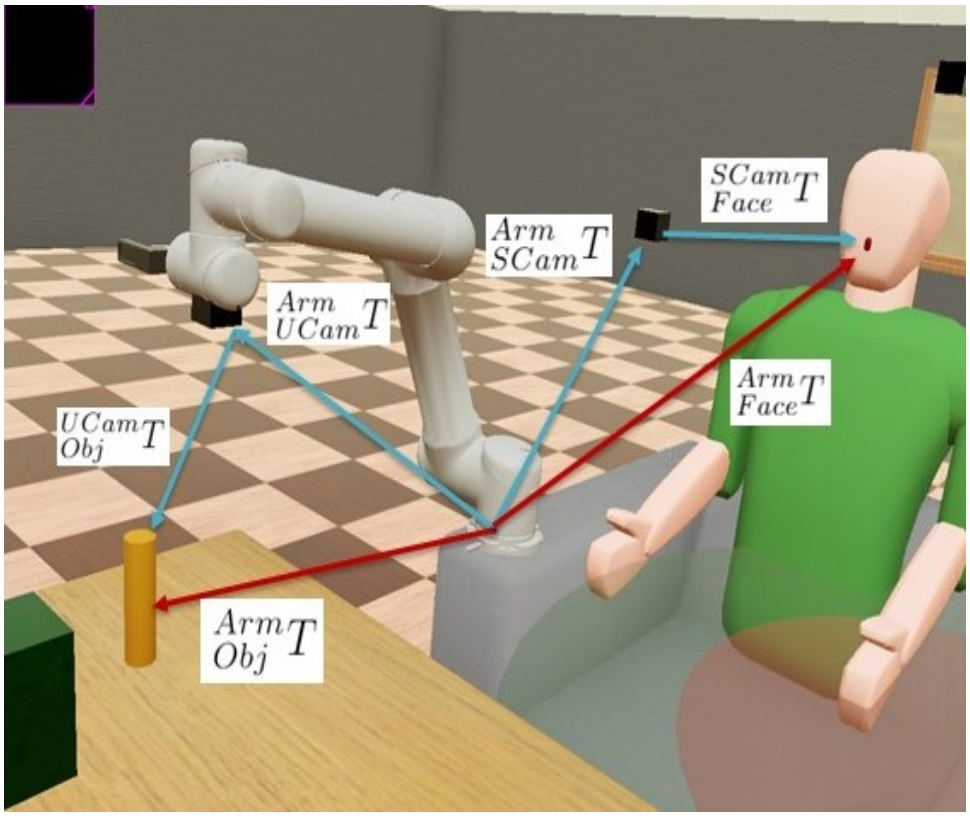


Figure 2.5: Transformation relationships of side and UR5e camera

A computer keyboard is used to command the robotic arm by mapping the ASCII code of the keys to specific commands. The acronym ASCII stands for American Standard Code for Information Interchange.

Figure 2.6 shows the overview of the process. A while loop runs every 64 milliseconds. Thus, the minimum time between any two commands or physics engine values returned is 64 milliseconds. This time step can be varied. A smaller time step increases the accuracy but decreases the speed of the simulation. A larger time step reduces the accuracy but increases the speed of the simulation. As long as the simulation is running, the while loop keep running and accepting commands from the keyboard to perform the mapped function. The simulation functions of the assistive

wheelchair and the robotic arm are mapped to different keyboard keys as given in table 2.1

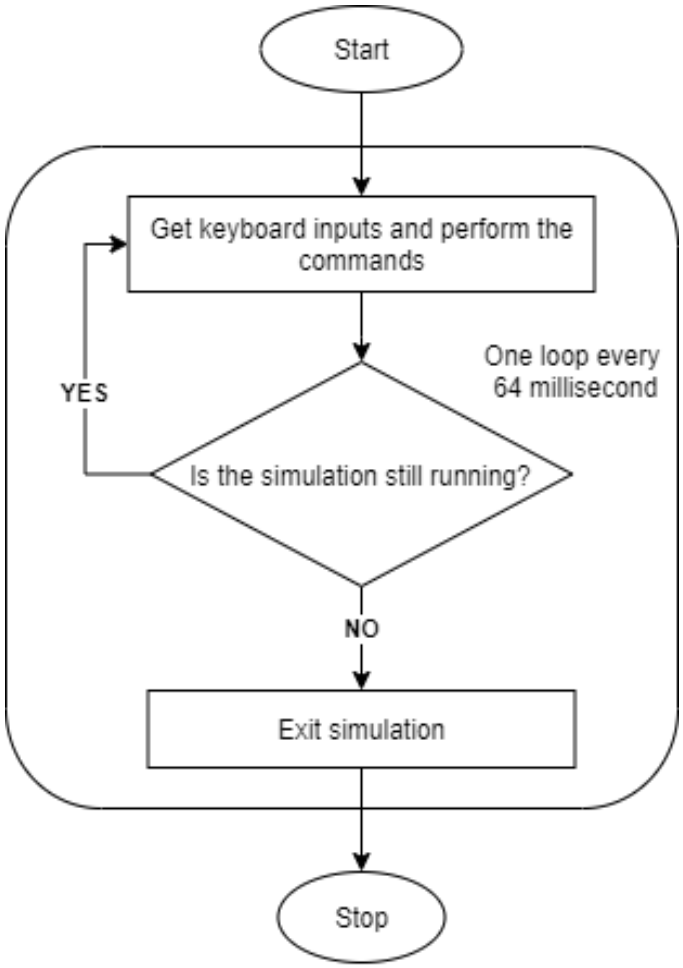


Figure 2.6: Keyboard modality flowchart

2.8 Object Mapping

This chapter explains the mapping process which replicates the objects from the Webots space to the MATLAB space. In this process, a Metadata file containing

Table 2.1: Keyboard command keys

Keyboard Key	**Function**
Arrows	Move the wheelchair
Page up	Pick-up an object
Page down	Place down an object
Home	Bring an object to user
Number pad	Select an object

the object information is used. This process is essential for obtaining information about the collisions from the MATLAB inverse kinematics space.

2.8.1 Metadata file

In this research, the simulation of the assistive environment is based upon two software, Webots, and MATLAB. Webots contains the information about the environment and MATLAB contains the controller code for running the simulation. It is essential to map the objects from the Webots environment into MATLAB for finding the collision between the arm and the objects and between the links of the arm itself, and trajectory points of the inverse kinematics solution in a collision. To facilitate this process of mapping objects, a Metadata file is used, which contains information about the geometry of the objects in the environment.

A Metadata file contains predefined information about all the objects in the environment. Table 2.2 shows the fields of the Metadata file with some example objects. The *object* field contains the name of the objects in the environment. The $Height$ and $Width$ fields are defined in meters and contain the dimensions of objects. These two fields are used for mapping the objects to MATLAB inverse kinematics space with their appropriate dimensions. [4] The *class* field informs about the geometry type of the object and is utilized in defining the approach path for the gripper. The

[4]The $Width$ field for a $Circle$ is its diameter

cylindrical objects can be approached by the gripper from any angle but for square cross-sectional objects, an approach along one of its sides is preferred for a better quality of grasping. The *Grasping Height* field defined in meters is used to determine the grasping point on the object with respect to its frame origin along its vertical axis. The grasping height is explained in greater detail in section 3.1.1. This Metadata file can be written in any format readable by MATLAB.

Table 2.2: Metadata file example

Object	Height(m)	Width(m)	Class	Grasping Height(m)
Blue can	0.1	0.05	Circle	0.025
Green can	0.09	0.06	Circle	0.0225
Magenta box	0.15	0.04	Square	0.0375

2.8.2 Mapping process

The object mapping process is used to map the objects from the Webots simulation space to the MATLAB inverse kinematics space. This allows the MATLAB inverse kinematics solution to determine if any collisions are occurring between the arm and objects or between links of the arm. The solution also outputs the trajectory points during which the collisions are occurring.

2.8.2.1 Mapping process flowchart

Figure 2.7 illustrates the steps of mapping. In the first step, object names and the object transformation matrices with respect to the overhead camera are retrieved from the Webots simulation space. This step provides the location and orientation of the objects with respect to the frame of the camera, i.e. ${}^{OCam}_{Obj}T$. In the next step, these transformation matrices are converted into transformation matrices that are

defined with respect to the robotic arm frame, i.e. ${}^{Arm}_{Obj}T$ with the help of equation 2.2. After this step, the location and orientation of the frame of objects are known with respect to the frame of the robotic arm. The next step provides the dimensions to the objects that are defined with respect to their frames. The dimensions of the objects are obtained from the Metadata file. In the next step, object meshes are created of the dimensions obtained in the previous step, at the location specified by the transformation matrices, ${}^{Arm}_{Obj}T$. This completes the process of mapping the objects from Webots space to MATLAB space.

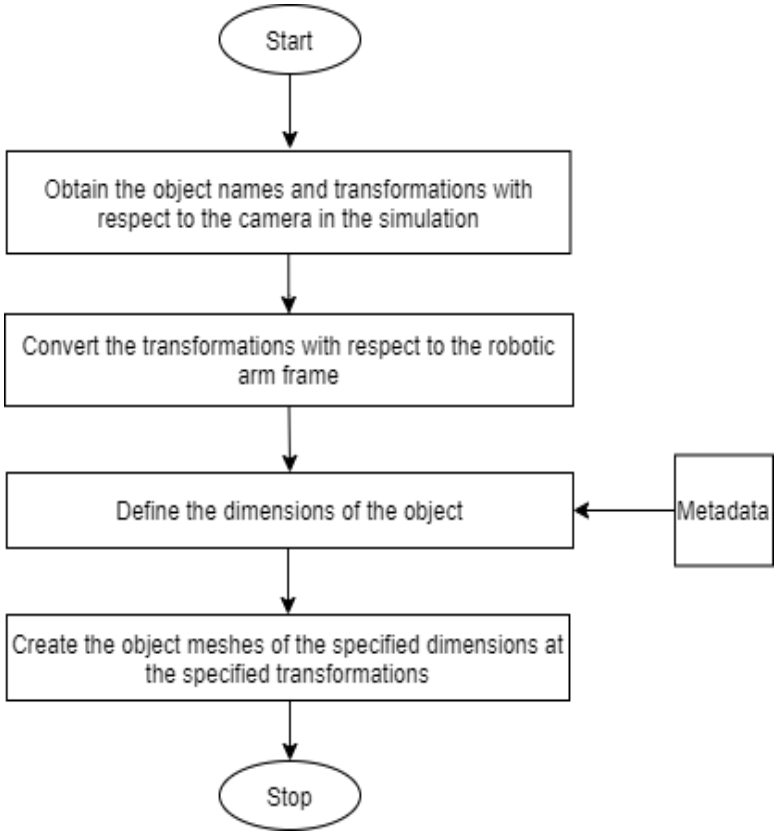
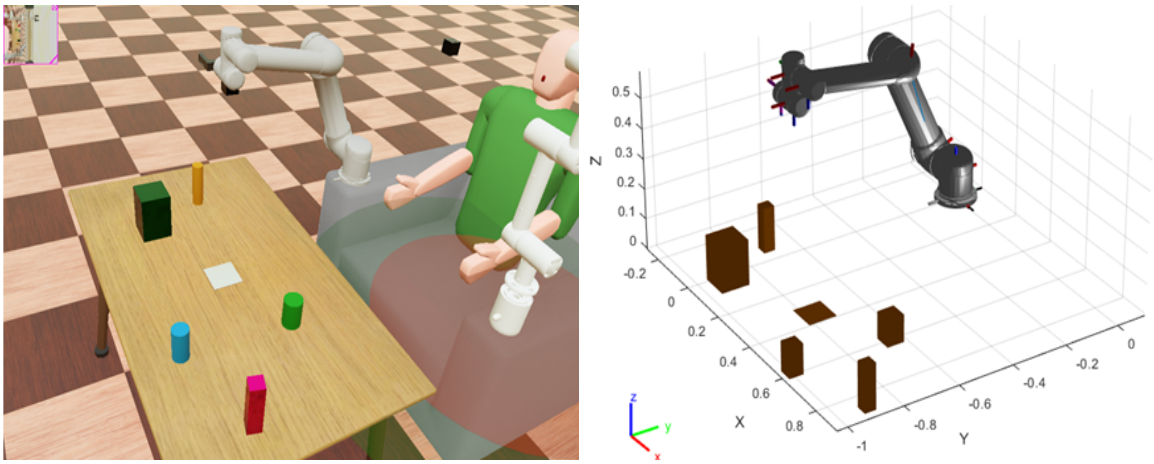


Figure 2.7: Mapping process flowchart

Figure 2.8 shows an example of the objects from the Webots simulation space mapped to the MATLAB inverse kinematics space. All the objects are mapped as rectangular cross-sections due to the limitations of the MATLAB 2019a version.



(a) Objects in Webots (b) Objects in MATLAB

Figure 2.8: Object mapping from Webots to MATLAB

This chapter provided a literature survey on the use of virtual simulations in robotic systems research with an introduction to Webots simulation. It explained the assistive wheelchair model used and the various components used in it. Furthermore, the simulation environment setup, object definitions, and the transformation relationships between various components of the environment are defined. The use of Metadata file was explained in the process of mapping objects from Webots space to MATLAB space.

CHAPTER 3

OBJECT MANIPULATION PROCESSES AND SIMULATION PHYSICS

This chapter describes the different processes for manipulating the objects such as pick-up, drop-off, bringing the objects to the user. The two types of gripper approaches are explained and the grasping height is defined with respect to the object frame. The inverse kinematics constraints applied to the robotic arm while performing these operations are described. Moreover, the performance of the two arms, UR5e, and the custom arm is compared. And the physics of the Webots simulation for torque requirements and collisions in Webots and MATLAB space are explained.

3.1 Gripper approach and grasping

3.1.1 Object grasping height

It is desirable to have the ability to grasp different objects at different heights to accommodate for varying cross-sections along the length of the object and the maximum gripper width. The gripper attached to both arms in this simulation has a maximum width of 0.09 m. Thus, it will be unable to grasp objects that are wider than 0.09 m. Figure 3.1 shows an object with varying cross-section along its length with wider sections of diameter 0.10 m and the middle cross-section of 0.05 m diameter. Thus, the gripper would only be able to grasp this object at its middle section as shown in figure 3.2.

The grasping height of the object is defined in a Metadata file as shown in table 2.2. This height is specified from the origin of the frame of the object along the vertical axis as shown in figure 3.3a. The gripper's outward axis, indicated by red

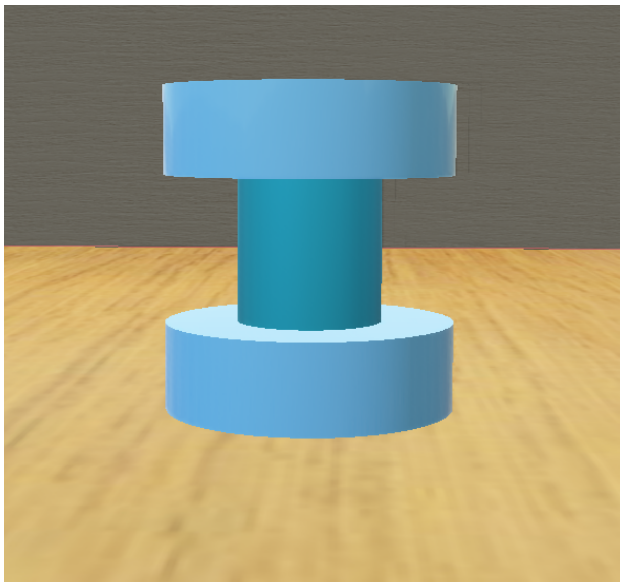


Figure 3.1: An object with varying cross-section along its length

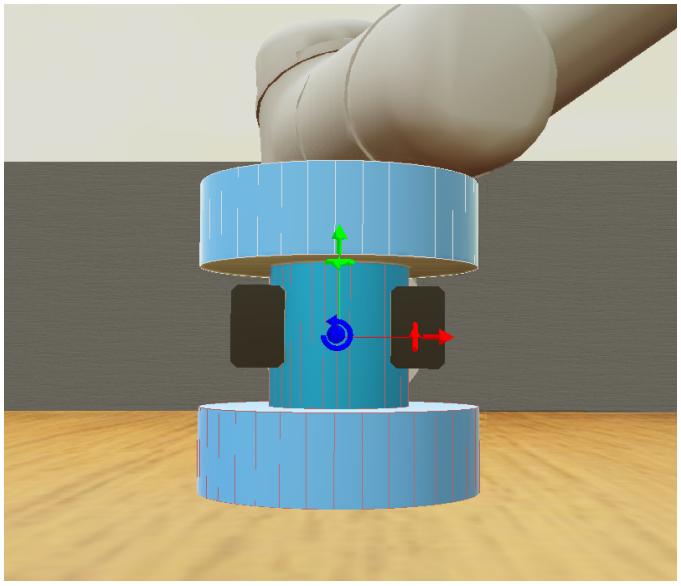


Figure 3.2: Grasping an object with varying cross-section along its length

color in figure 3.3b performs grasping at this height. In this example of figure 3.2, the grasping height is kept zero as the ideal grasping point is at the center of the object.

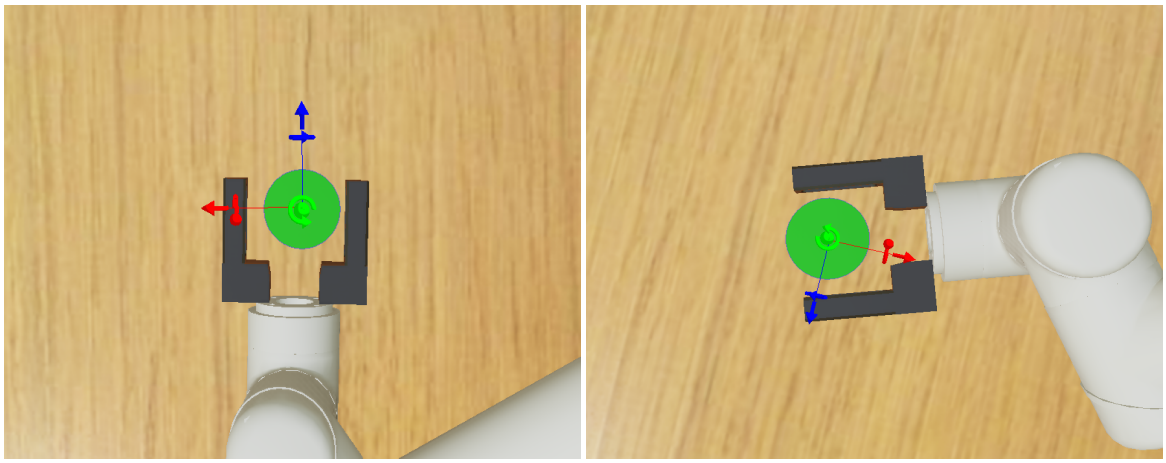
(a) Object frame (b) Gripper frame

Figure 3.3: Object grasping height

3.1.2 Gripper approach towards an object

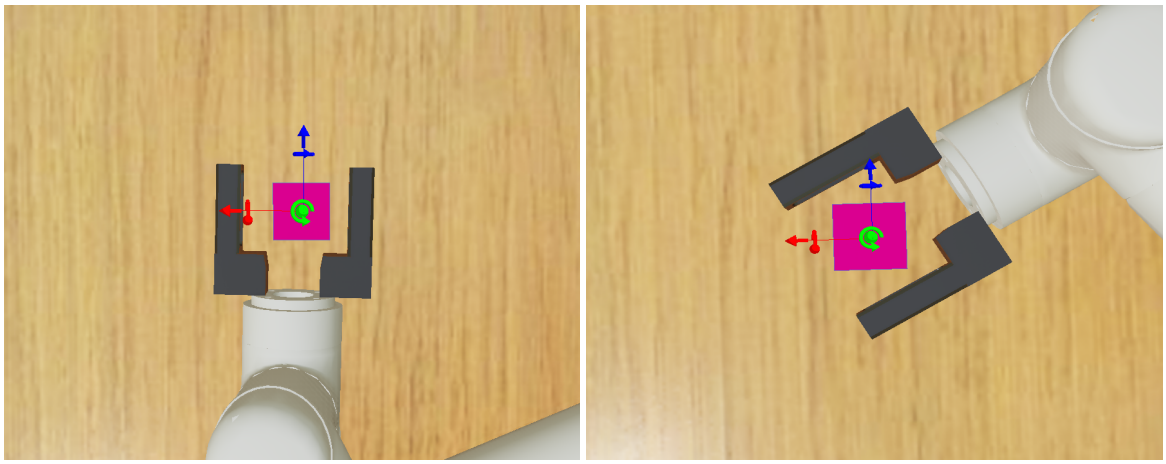
The ability to approach an object from different angles is important for a robotic arm as the objects of the different cross-sections can be handled better. For an object of circular cross-section, the approach from any angle is the same as far as the quality of grasping is concerned as shown in figure 3.4. However, for non-circular cross-sections like a box, the quality of grasping depends on the angle that the gripper approaches the vertical surface of the object. In such a case, it is desired to approach the object perpendicular to its side rather than at an inclined angle or along the edge as shown on figure 3.5.

From the above discussion, the approach of the gripper is defined in two different ways; The axis approach and the displacement approach. It has been shown that the axis approach is used to approach an object along a specified axis of the object frame. This is preferred to approach an object of non-circular cross-section like a box. And the displacement approach is used to approach an object along a straight line joining an object origin to the robotic arm origin. This approach is preferred to approach an object of circular cross-section like a cylinder.



(a) Axis approach (b) Displacement approach

Figure 3.4: Approaching circular cross-sectional object



(a) Axis approach (b) Displacement approach

Figure 3.5: Approaching non-circular cross-sectional object

3.1.3 The gripper axis approach

The gripper axis approach is specifically used for approaching the objects of non-circular cross-sections such as square boxes along a specified axis. Figure 3.6 illustrates this approach. This figure shows the UR5e arm with objects in the MATLAB space.

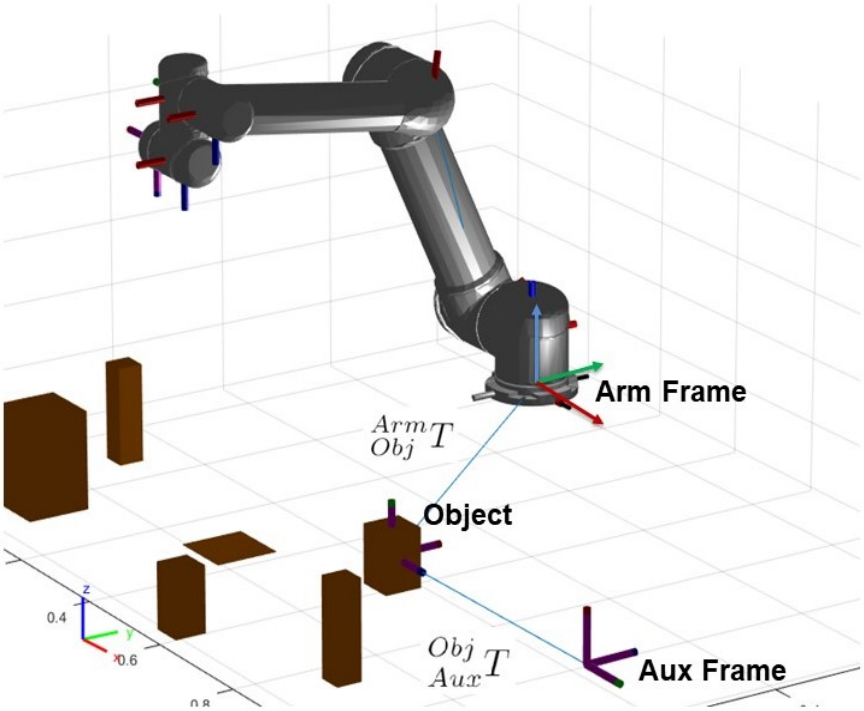


Figure 3.6: Gripper axis approach

The object frame is defined with respect to the robotic arm frame in the form of transformation given by equation 3.1.

$$^{Arm}_{Obj}T = \begin{bmatrix} T_{xx} & T_{yx} & T_{zx} & P_x \\ T_{xy} & T_{yy} & T_{zy} & P_y \\ T_{xz} & T_{yz} & T_{zz} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

For the axis approach, an auxiliary frame is defined with respect to the object frame in the form of transformation given by equation 3.2. In other words, the auxiliary frame could be created by simply displacing the object frame along any one of its axes. Equation 3.2 is applicable for the axis approach along the Z-axis only as the displacement D is defined at Z coordinate position in the transformation matrix. For

approach along the X-axis and Y-axis, the displacement D needs to be defined in the X and Y coordinate position of the matrix respectively. The displacement D is any value greater than zero.

$$ {}^{Obj}_{Aux}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & D \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

The auxiliary frame is created for the reason of implementing a certain constraint on the gripper frame of the robotic arm for its approach. For approaching along an axis, the gripper frame's outward axis, which is the Z-axis indicated with blue color, in this case, is under a constraint to point to the origins of the object frame and the auxiliary frame as shown in figure 3.7. Thus with this constraint, the gripper frame's outward axis is aligned to the line joining the object frame origin and the auxiliary frame origin. The gripper frame stays in this constraint from the approach point up to the grasping point.

3.1.4 The gripper displacement approach

The displacement approach of the gripper is useful for approaching the objects with circular cross-sections such as cylinders along the line joining the robotic arm frame origin and the object origin. Figure 3.8 illustrates this approach. This figure shows the UR5e arm with objects in the MATLAB space.

For this type of approach, the auxiliary frame is defined with respect to the robotic arm frame by the transformation given by equation 3.3. The coordinates D_1 and D_2 lie on a projected vector on the XY plane joining the robotic arm origin and the object origin. These coordinates need to be greater than the corresponding coor-

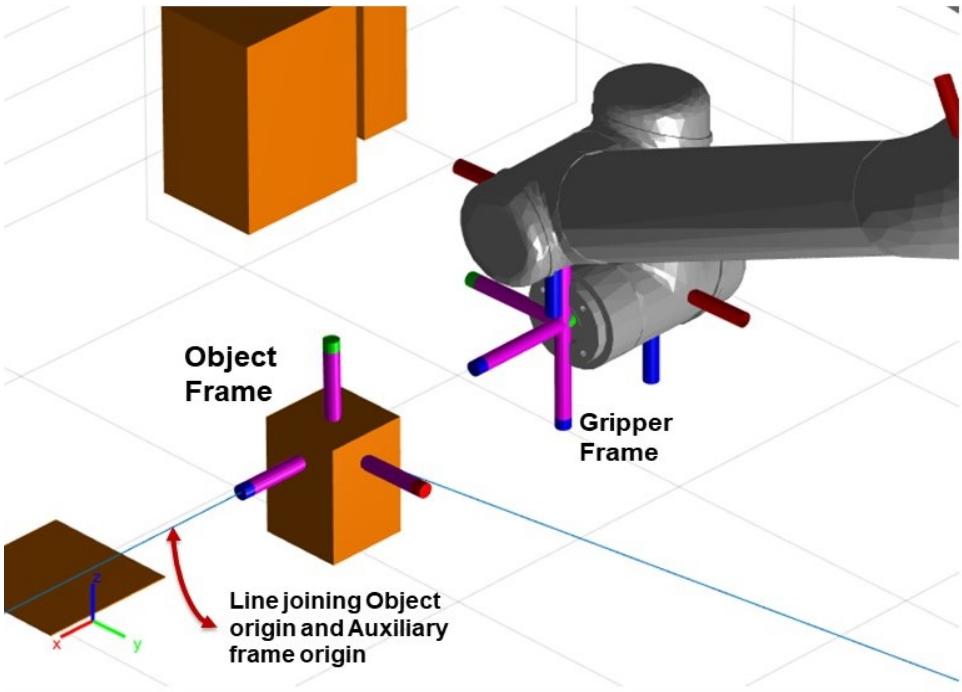


Figure 3.7: Gripper aligned along the object's z-axis

dinates of the object frame with respect to the robotic arm frame. The Z coordinate is maintained the same as the object frame, P.

$$ {}^{Arm}_{Aux}T = \begin{bmatrix} 1 & 0 & 0 & D_1 \\ 0 & 1 & 0 & D_2 \\ 0 & 0 & 1 & P \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3} $$

For approaching along a displacement line, the gripper frame's outward axis is under a constraint to point to the origins of the object frame and the auxiliary frame. Thus, the gripper's outward axis is aligned to the line joining the origins of the object frame and the auxiliary frame.

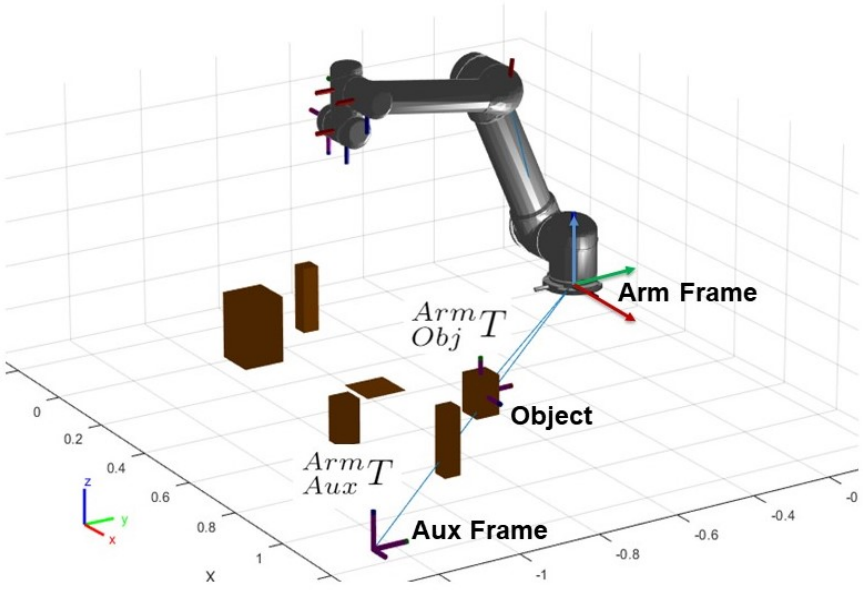


Figure 3.8: Gripper displacement approach

3.2 Pick-up of visible objects

The objects in the simulation environment which are visible to the overhead camera are referred to as visible objects. And the objects which are not visible to the overhead camera are referred to as hidden objects. For the detection and pick-up of the hidden objects, the UR5e attached camera is used which is described in detail in section 3.3.

Figure 3.9 shows the flowchart for the pick-up of visible objects as seen through the overhead camera. In the first step, objects are detected by the overhead camera and displayed as an array with their serial numbers and names. Next, user input is given as a serial number to pick up an object. Then the geometry of the selected object is identified with the input of the Metadata file. The grasping point of the object is also defined along its vertical axis as given by the grasping height given in the Metadata file. In the following step, the transformation matrix of the object with

respect to the robotic arm frame ${}^{Arm}_{Obj}T$, given by equation 2.2, is fed as an input to the inverse kinematics solver to solve for the required trajectory to grasp the object. Then the solved trajectory is checked for collisions. The result of the collision detection is displayed and an option to carry or abort the arm motion is provided.

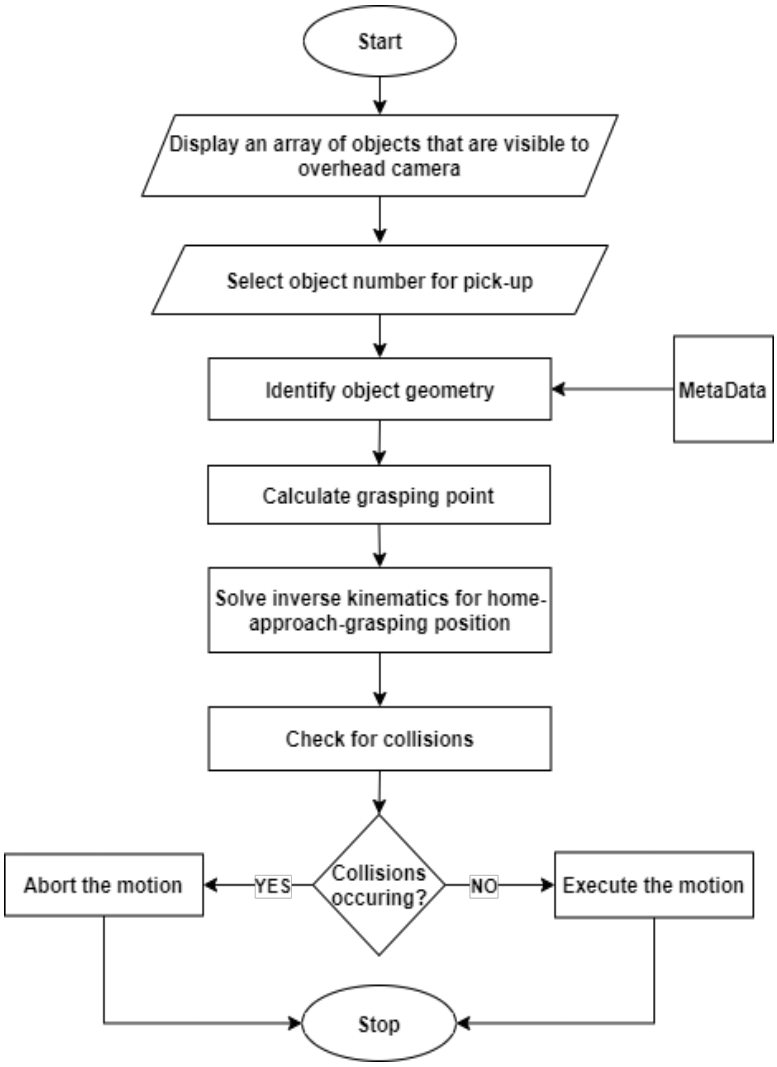


Figure 3.9: Visible Object pick-up flowchart

This is a brief overview of the pick-up process for the visible objects. The following subsection details the inverse kinematics constraints applied for this arm motion.

3.2.1 The inverse kinematics constraints applied

The inverse kinematics solver used in this simulation is the generalized inverse kinematics solver provided by the MATLAB robotic systems toolbox. This inverse kinematics solver satisfies multiple constraints to achieve the required motion. These constraints define the trajectory of the robotic arm such as approach distance, gripper alignment, and position tolerance, and work-space boundary of the arm to avoid collisions. Following are the constraints applied on UR5e and the custom arm for various object manipulation processes.

A *Cartesian Bounds* Constraint [1] requires the specified frames of the robotic arm to remain within the specified Cartesian coordinates defined with respect to the base frame of the arm. Equation 3.4 shows the format of this constraint. Each row of this constraint specifies the minimum and the maximum distance along each axis that the specified frames are allowed to move in.

$$Cartesian\ Bounds = \begin{bmatrix} X_{min} & X_{max} \\ Y_{min} & Y_{max} \\ Z_{min} & Z_{max} \end{bmatrix} \tag{3.4}$$

Figure 3.10 shows the application of this constraint on the UR5e arm. The gripper frame and the forearm frame of UR5e are constrained the Z direction and have no constraint in the X and Y direction as given by equation 3.5. This constraint is

applied to both the robotic arm to prevent collision with the table. The surface of the table is assumed to be at 0.05 m along the Z-axis of the base frame of the arm.

$$Cartesian\ Bounds\ on\ gripper\ and\ forearm\ frame = \begin{bmatrix} -\infty & \infty \\ -\infty & \infty \\ 0.05 & \infty \end{bmatrix} \tag{3.5}$$

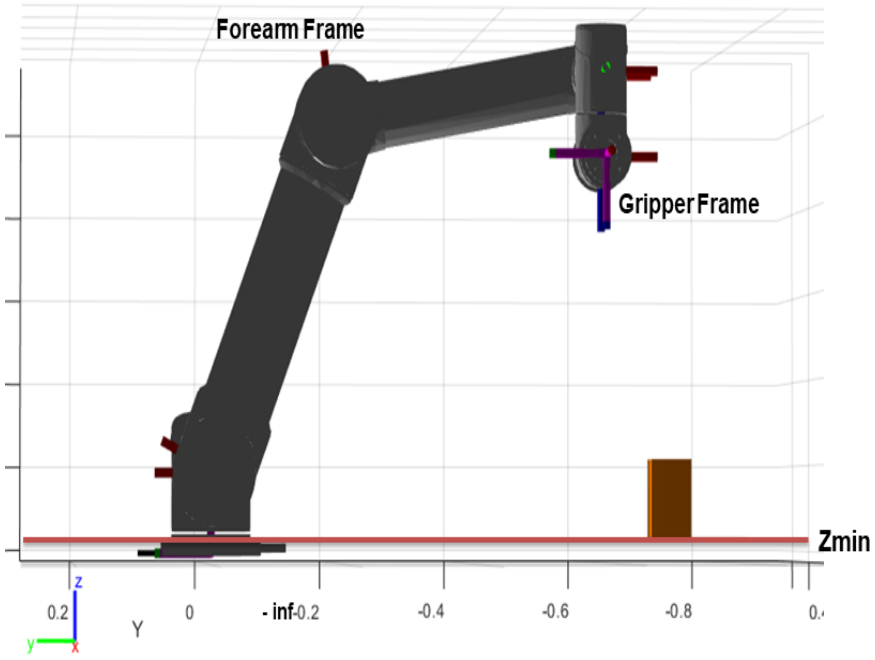


Figure 3.10: Cartesian Bounds Constraint

A *Joint Bounds* Constraint [1] applied limits the maximum change in the joint configuration along a trajectory. This provides a smooth trajectory for the arm motion by equally spacing the trajectory way-points.

An *Aiming* constraint [1] requires the Z-axis of any body to aim at a target point on another body. This constraint is applied on the gripper frame to align the gripper's Z-axis, the axis pointing outward indicated by blue color, to point to the

center of object frame and auxiliary frame. Figure 3.11 shows this application. The detailed use of this constraint is explained in section 3.1 for approaching an object in two ways, the axis approach and the displacement approach.

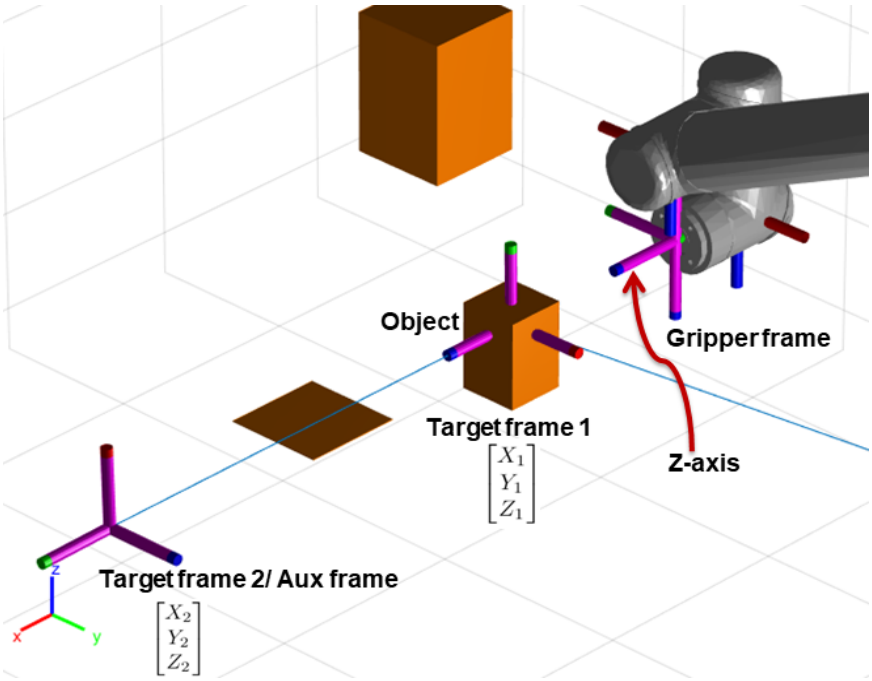


Figure 3.11: Aiming Constraint

An *Orientation Target* Constraint [1] keeps the gripper frame orientation constant from the approach point to the grasping point of the object. This constraint also allows to specify the tolerance for the orientation angle. The format is given by equations 3.6 and 3.7.

$$fixOrientation \;=\; robotics.OrientationTarget(gripper) \tag{3.6}$$

$$fixOrientation.OrientationTolerance \;=\; deg2rad(2) \tag{3.7}$$

The approach point is a point near an object at which the gripper must first reach before beginning its approach towards the grasping point. A *Position Target*

constraint [1] is used to define the approach point by setting the position of the object frame in the gripper frame as given by equation 3.8. This constraint specifies the distance at which the gripper must begin its approach towards the object and the distance from the object to stop at. Figure 3.12 illustrates the concept.

$$Target\ Position\ in\ gripper\ frame = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ Approach\ Distance \end{bmatrix} \tag{3.8}$$

The approach distance is calculated by equation 3.9. The object width is taken from the metadata file. The gripper's length is taken into consideration to avoid collision with the object while reaching the approach point.

$$Approach\ Distance\ =\ Object\ Width\ + (1.5\ Gripper\ Length) \tag{3.9}$$

All the above constraints are applied to the robotic arm for solving the required trajectory for object pick-up.

3.3 Pick-up of hidden objects

Sometimes an object may be hidden from the overhead camera. This can happen when a bigger object is covering up a smaller object from the view of the overhead camera. In such a case, the UR5e attached camera is used.

Figure 3.15 shows the pick-up process flowchart of the objects that are hidden from the overhead camera. In the first step, all the objects that are visible to the overhead camera are displayed in an array consisting of object serial numbers and names. Then the user inputs the serial number of the object that is closest to the hidden object. In most cases, this is the object that is covering up the hidden object from the field of view of the overhead camera. Once the closest object is selected, the transformation matrix of that object is fed into the inverse kinematics solver to solve

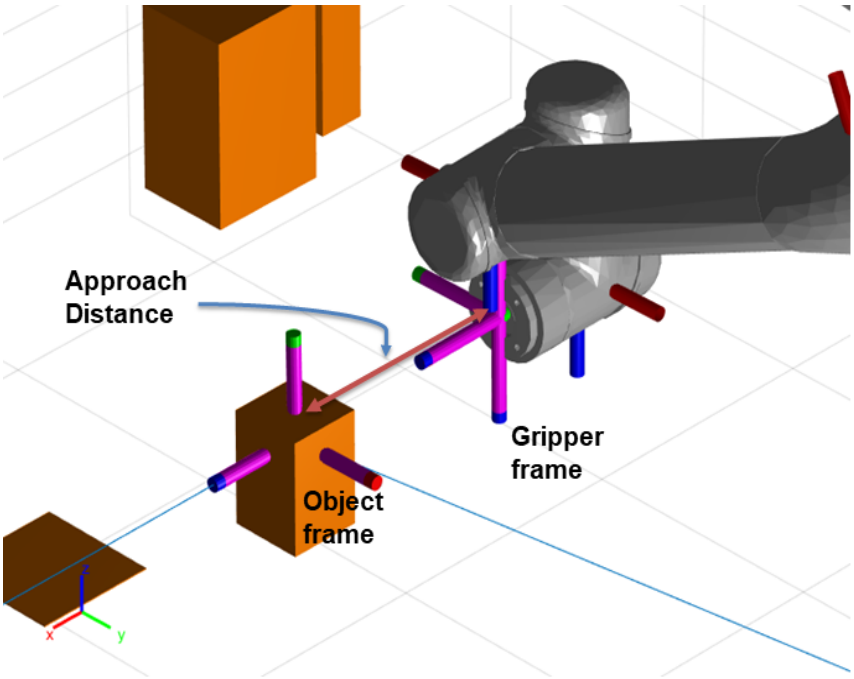
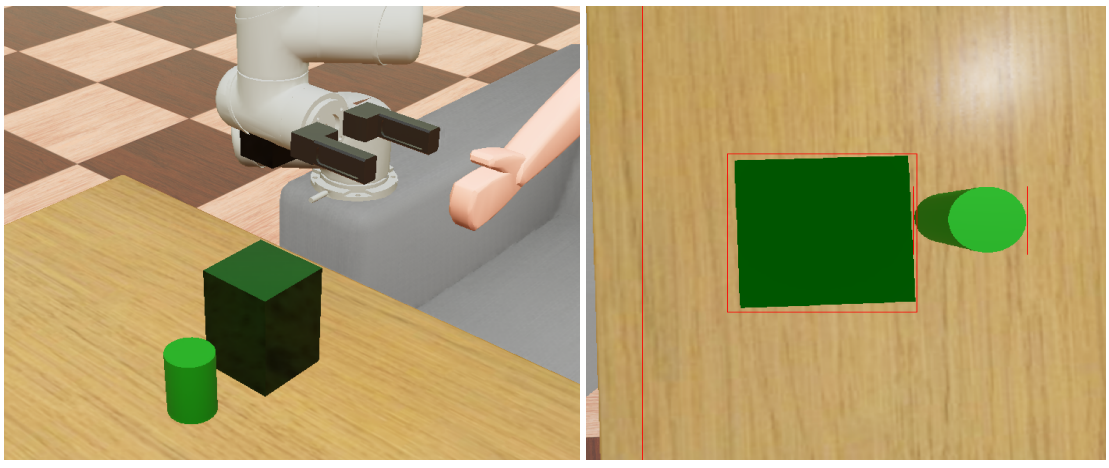


Figure 3.12: Approach Distance

for the trajectory and to reach at a certain height above the object. At the end of this trajectory, UR5e camera view is oriented downward towards the object.



(a) Reaching over the object (b) View of UR5e camera

Figure 3.13: Use of UR5e camera

Figure 3.13 shows an example of this. The green cylindrical object is hidden from the view of overhead camera by a bigger box in front of it. UR5e arm reaches over the box with UR5e camera pointing downward and detects the hidden object. Then an array of objects which are visible to UR5e camera is displayed with their serial number and names. The user selects the serial number of the hidden object. The object geometry is identified with the use of Metadata file. The transformation matrix of the hidden object with respect to the base frame of the robotic arm, i.e. ${}^{Arm}_{Obj}T$, is calculated as given in equation 2.3, and fed to the inverse kinematics solver to solve for the required trajectory. A collision check is performed for the solved trajectory and an option to carry or abort the motion is provided to the user. Figure 3.14 shows a hidden object reached by UR5e arm.

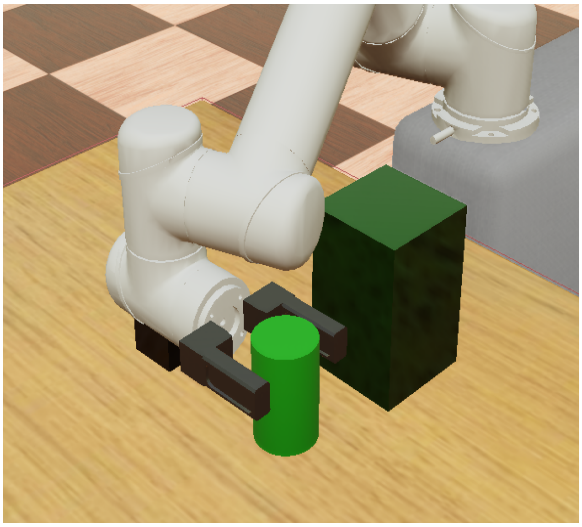


Figure 3.14: Hidden object reached

During the entire process of hidden object pick-up, the constraints specified in section 3.2.1 are applied to the robotic arm.

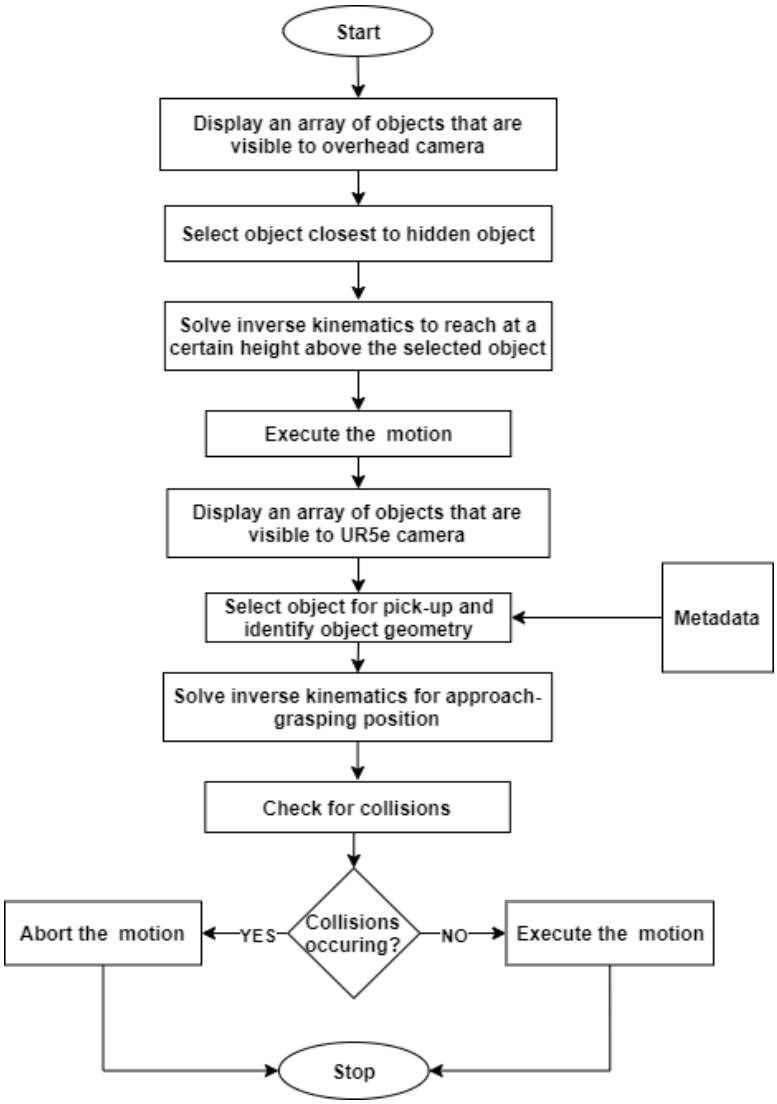


Figure 3.15: Hidden object pick-up flowchart

3.4 Bringing objects to user

After the object is picked up, the arm reaches an intermediate holding position. From this position, the object can be brought to the face of the person sitting on the wheelchair. Figure 2.5 shows the relationship of arm base frame to face frame of the person, i.e. ${}^{Arm}_{Face}T$. For detecting the location and orientation of face frame, side

camera is used. Furthermore, by displacing face frame along its Z-axis, an auxiliary frame as shown in figure 3.16 is generated.

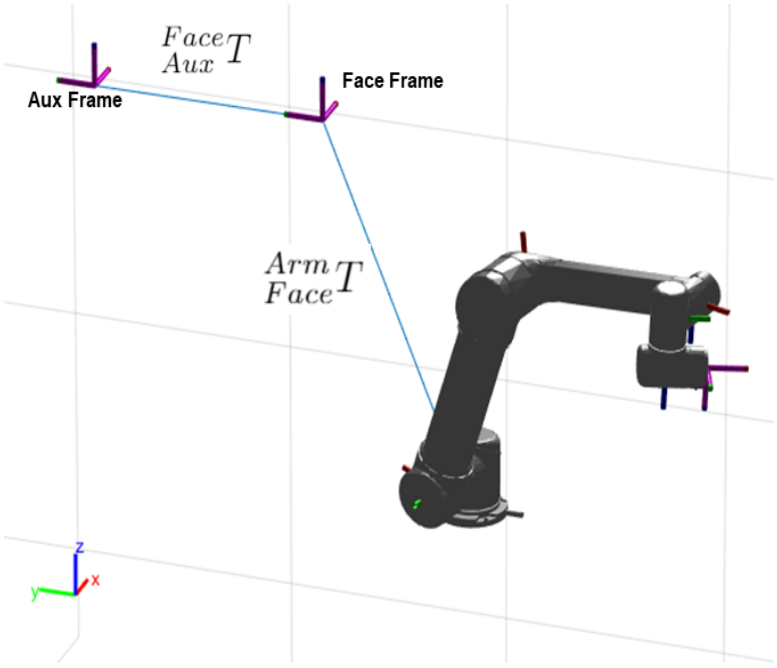


Figure 3.16: Face frame displacement

The process for bringing the object to the user is shown as a flowchart in figure 3.17. A picked up object is held by the robotic arm at an intermediate holding position until the next command is issued. As a first step, an array of objects visible to the side camera is displayed with serial number and names. Then, the object number associated with the human face is selected. Then the transformation matrix of face frame with respect to base frame of the arm is calculated and fed into the inverse kinematics solver. An auxiliary frame is created by displacing face frame by some arbitrary distance along its z-axis. This auxiliary frame is later used for the axis approach for approaching face frame along its Z-axis as described in section 3.1.3. After obtaining the inverse kinematics solution trajectory for the arm, collision

detection check is performed in MATLAB and options to execute or abort the motion are provided. If the motion is executed, the object is brought to user and held their for a predefined amount of time. After that, the arm brings the object back to intermediate holding position. Now the object is ready to be placed back down on the table. This process is illustrated in section 3.5.

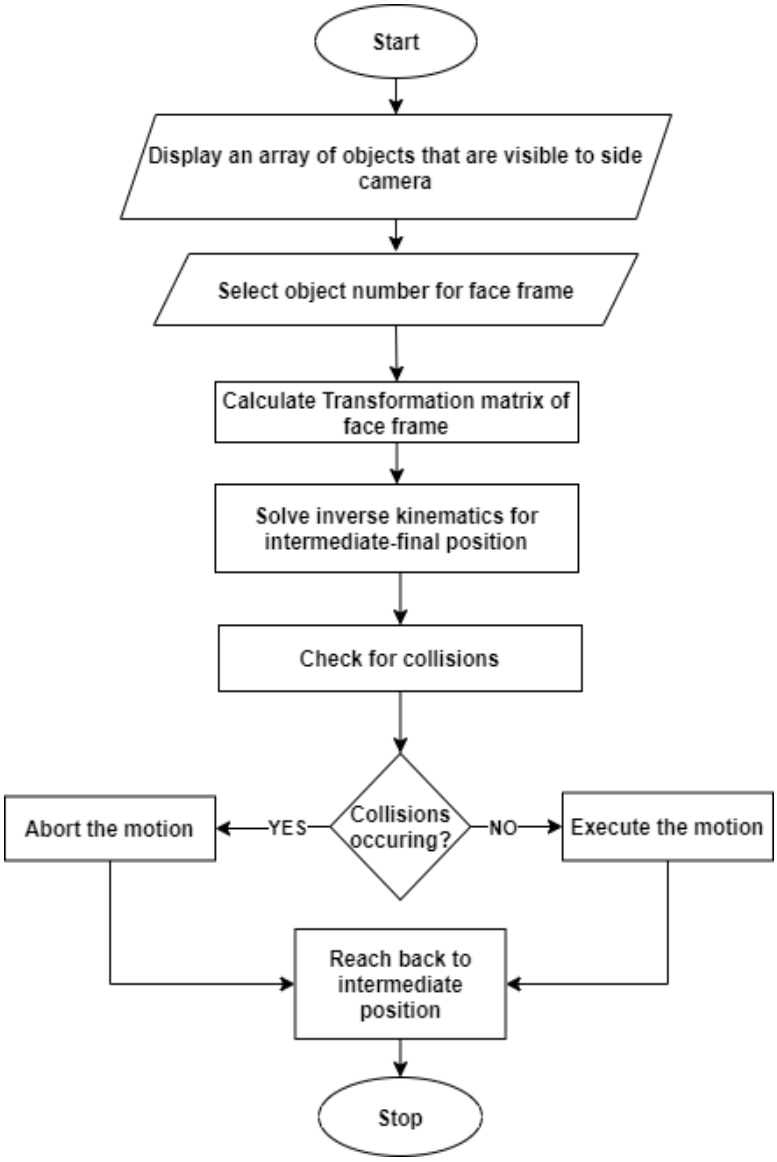


Figure 3.17: Bringing object to user process

3.5 Drop-off of objects

After the object is picked up or brought to the user, it is held at an intermediate position until a next command is issued. This intermediate position is shown in figure 3.18, where both the arms are holding objects. The picked up object can be placed on other objects called pads in the environment. Pads are the objects of very low thickness in the order of 0.001 m. Figure 3.18 shows a white colored pad in the bottom left hand corner.

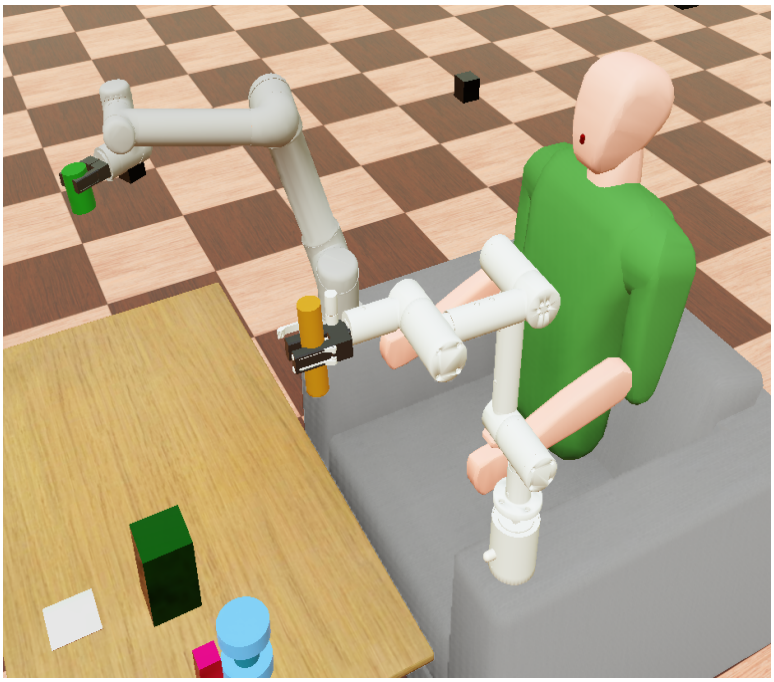


Figure 3.18: Intermediate position of the two arms

The detailed process of object drop off is explained as a flowchart in figure 3.19. The object drop off process starts from the intermediate position of the arm shown in figure 3.18. As a first step, an array of objects visible to the overhead camera including the pads are displayed with their serial number and names. Next, the user selects the number associated with the desired drop off pad. The final transformation

matrix of the object (pad) with respect to the base frame of the arm is obtained, i.e. ${}^{Arm}_{Obj}T$. In the P_z co-ordinate of this transformation matrix, a user defined

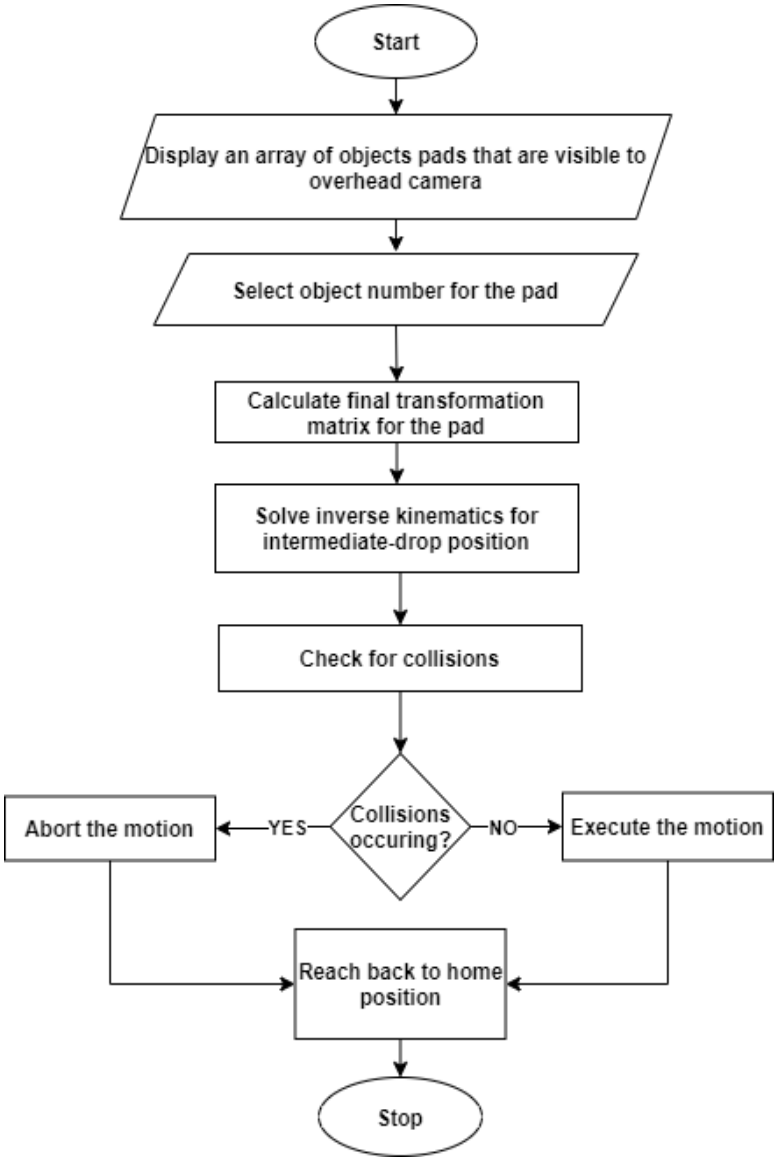
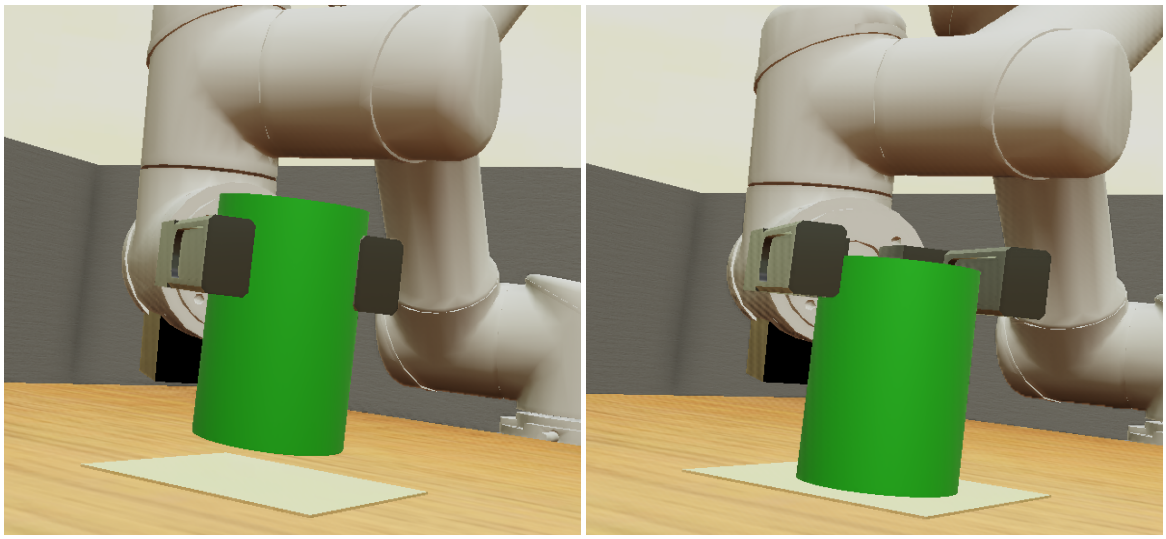


Figure 3.19: Object drop off process

positive distance of 5 cm is added as a drop off height. The object is brought to this height first before release of the gripper as illustrated in figure 3.20. This modified

transformation matrix is fed into the inverse kinematics solver to get the required trajectory of the arm. Collision check is performed and options to execute and abort are provided. After dropping off the object, the arm returns to its home position. For drop off, only the displacement approach of the gripper is used.



(a) Reaching over the pad (b) Release of the object

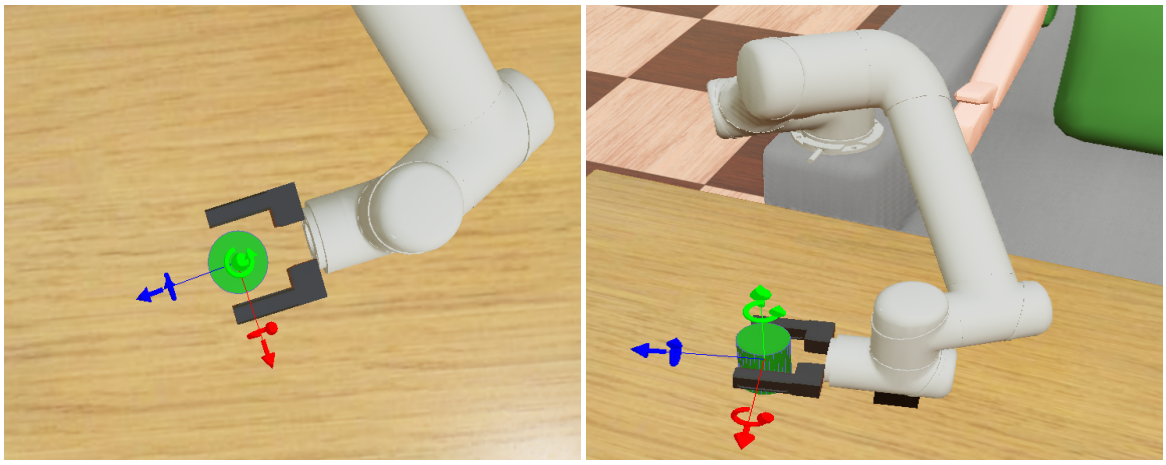
Figure 3.20: Object drop off

3.6 Performance comparison of the two arms

This section provides a comparison of the performance of the UR5e arm and the custom arm. UR5e is 6 degrees of freedom arm, whereas the custom-made arm is of 4 degrees of freedom. Thus, the UR5e arm is capable of performing more complex tasks than the custom arm as described in the following paragraphs.

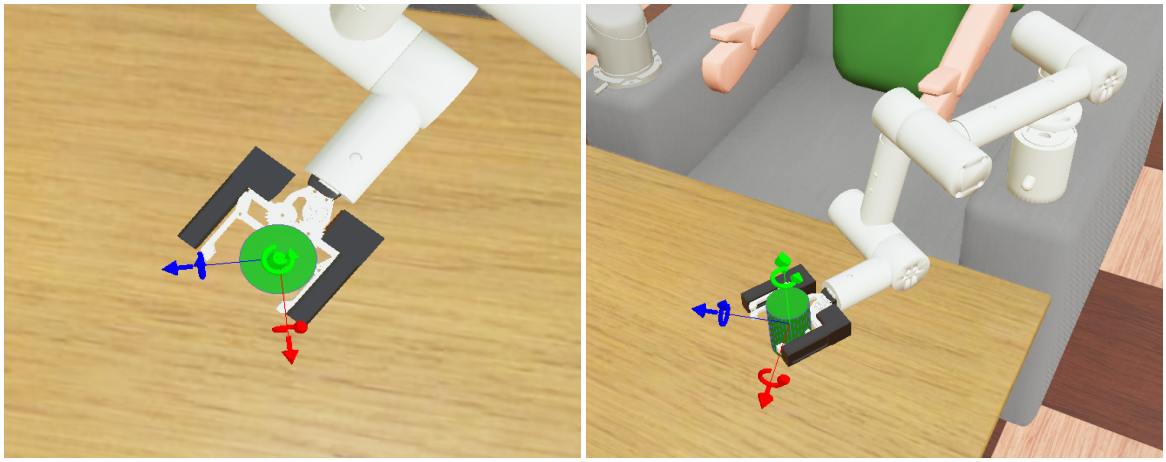
With two additional degrees of freedom over the custom-made arm, UR5e is capable of approaching an object along its axis within its work space. The custom arm is not capable of axis approach and can only perform displacement approach.

Figures 3.21 and 3.22 show the results of how the two arms perform when tasked to approach along the Z-axis of the object which is indicated with blue color. As observed, the UR5e arm performs the task properly, while the custom arm fails to align its gripper along the Z-axis of the object.



(a) UR5e gripper top view (b) UR5e total view

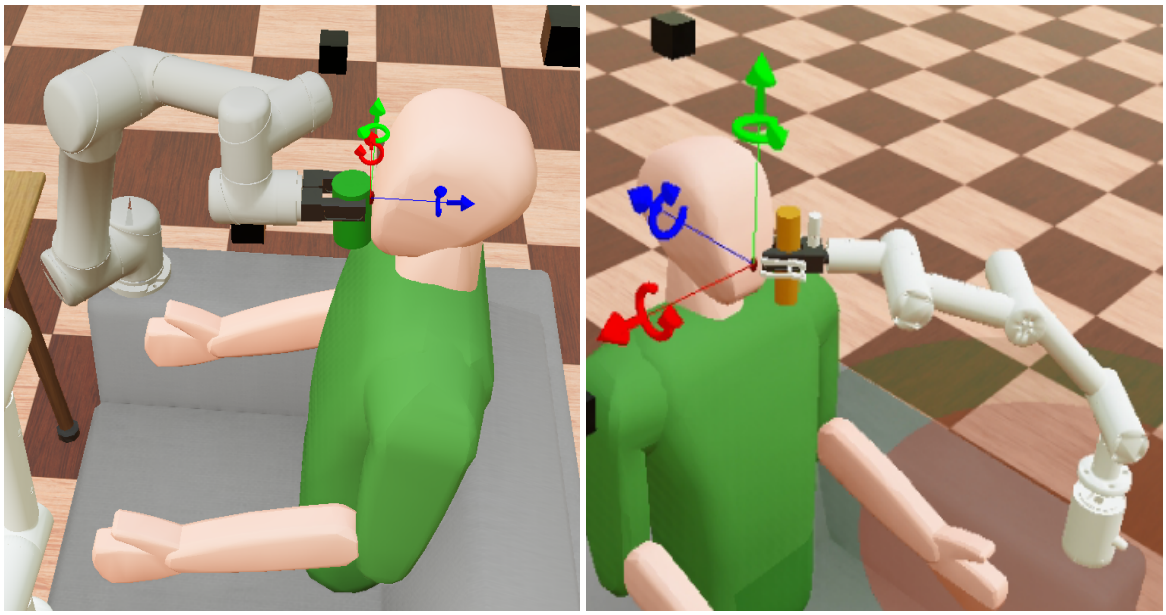
Figure 3.21: UR5e approaching an object along z-axis



(a) Custom arm gripper top view (b) Custom arm total view

Figure 3.22: Custom arm approaching an object along z-axis

The same limitation of the custom arm is also seen when it brings the object to the user and cannot approach the human face from the front or along the Z-axis of the face which is indicated with blue color, whereas UR5e successfully approaches the face from the front as shown in figure 3.23.



(a) UR5e bringing an object to user (b) Custom arm bringing an object to user

Figure 3.23: UR5e and the custom arm bringing an object to user

3.7 Simulation physics

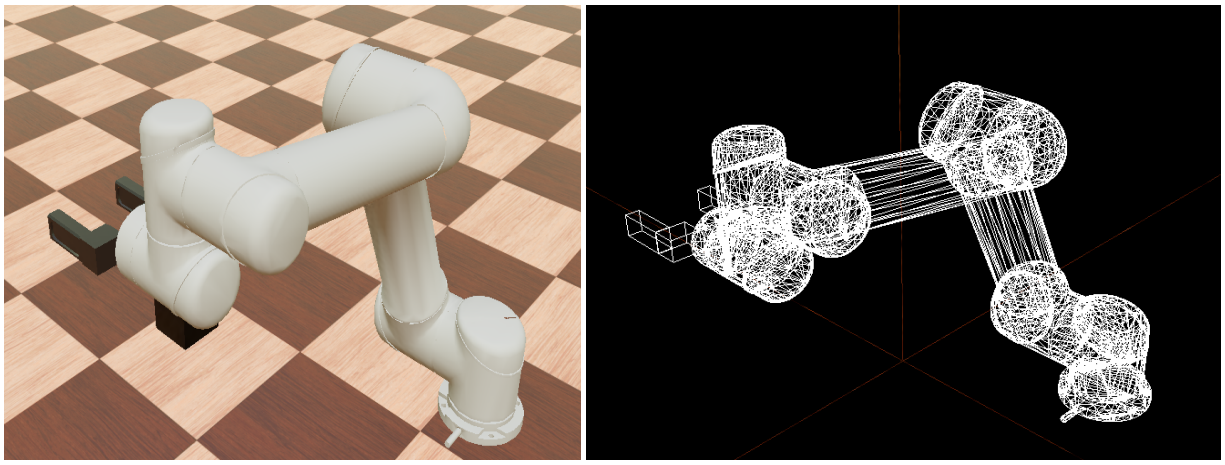
This chapter describes the method that is used in the Webots simulation space for detecting the collisions and retrieving the torque requirement of the robotic joint motors. Also discussed is collision detection in the MATLAB space.

3.7.1 Physics of Webots space

Webots simulation space is built with a hierarchy of various nodes creating objects in the environment. Different nodes serve different purposes. In what follows, the nodes essential for simulation of physics are discussed.

3.7.1.1 Collision in Webots and essential nodes

A *solid* node is the base node for detecting objects in a collision. Hence, the objects which are created using this node in the simulation have active collision detection such as the robotic arms and objects on the table. Within the *solid* node, a *bounding object* field exists which specifies the geometrical primitives used for collision detection. If the *bounding object* field is null, no collisions can be detected. Each *bounding object* field can be defined with one or more primitives. The primitive shapes are defined in such a way as to approximate the physical bounds of the object. The various primitives that can be used are box, capsule, cylinder, sphere, IndexFaceSet, etc. Figure 3.24 shows the UR5e arm with its bounding object. [15]



(a) UR5e Arm (b) UR5e bounding object

Figure 3.24: UR5e and its bounding object

A *physics* node is used to specify the parameters for the Open Dynamics Engine such as mass, the center of gravity, inertia matrix, density, damping, etc. The *physics* node is a child of the *solid* node. If the *physics* node is defined, the solid will have physics behavior. If the *physics* is not defined, the solid will have a kinematic behavior. The *bounding object* field with the *physics* node is used to compute the inertia matrix of the objects by assuming a uniform distribution of the mass in the primitives of the *bounding object*. [15]

The *bounding object* primitives help the objects in Webots to interact physically with each other. Thus, when two *solids* having a defined *bounding object* touch each other, a collision occurs and the objects exert equal and opposite forces on one another.

3.7.2 Simulation torque of motors

Both the robotic arms simulated, UR5e and the custom arm, were exported as an URDF[1] file and imported into the simulation space as a VRML[2] file. These files contained the physical information about the arms such as their mass, center of gravity, inertia, etc. which is preserved into the simulation and used to calculate the motor torque requirements for each joint. With the use of the appropriate inbuilt command functions in Webots, the torques were obtained.

Figure 3.25 shows the torque requirement for the motors of the custom arm in a simulation scene where the arm is reaching an object for pickup. The time for this motion was defined to be 9 seconds in the inverse kinematics solver. Similarly, the torques for the UR5e arm can be obtained as shown in figure 3.26, which is for a motion of reaching an object for pick up. In both these figures, the arms reach the

[1]URDF: Unified Robot Description Format

[2]VRML: Virtual Reality Modeling Language

approach point at 1 second and then the arm moves from the approach point to the grasping point over the next 8 seconds. As a result, the torques change less after the first second. These times are user defined and can be varied.

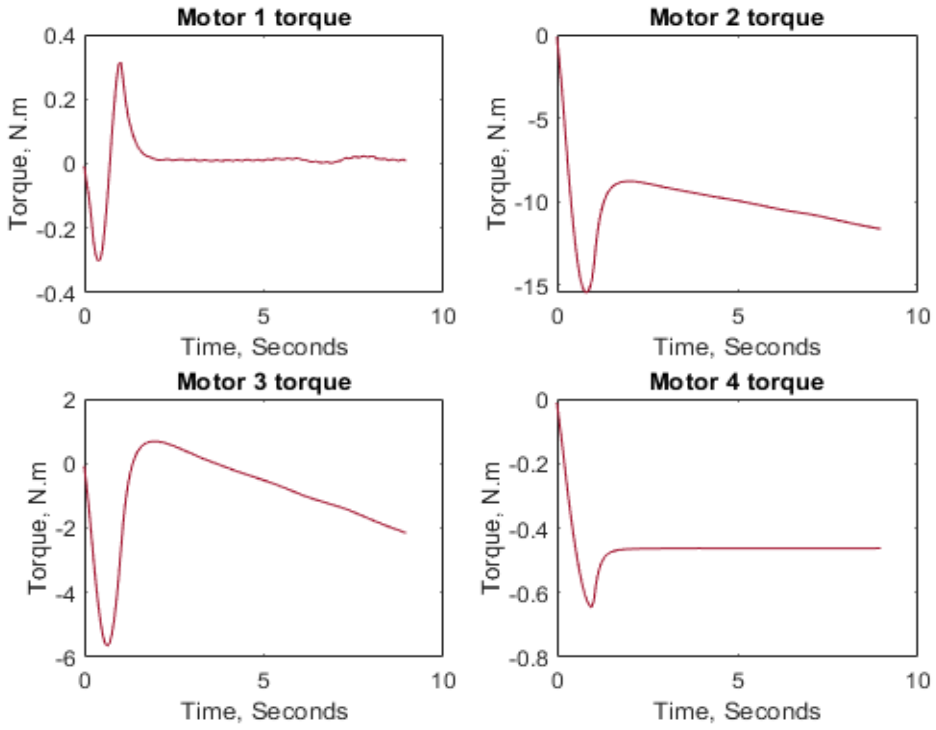


Figure 3.25: Torques returned by Webots for the custom arm

Obtaining these simulation torque results guides the selection of motors with appropriate torque output, which can be used for building a real robotic arm. These physics results provided by the Webots were verified by a simple experiment as follows. The custom arm was held steady in the configuration shown in figure 3.27, with joint 4 bent at 90 degrees and all other joints at 0 degrees. The gripper had a can grasped whose mass was varied. The torque exerted on the joint 4 by the can mass and the mass of the link itself was calculated manually and compared to the torques returned

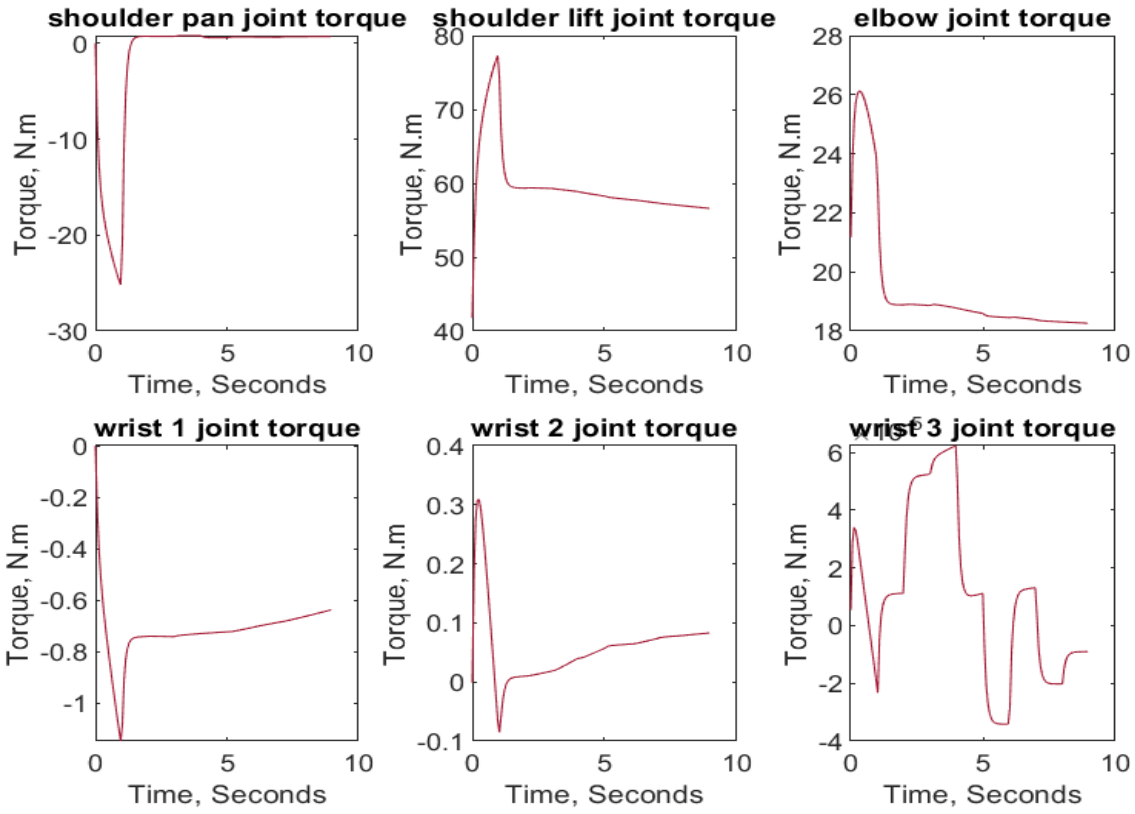


Figure 3.26: Torques returned by Webots for UR5e arm

by Webots as shown in table 3.1. The torques were calculated by using the center of gravity of the link 4 and the can, and their distance to the joint axis as shown by equation 3.10. In this equation, F is the force exerted by the link 4 and the can due to gravity and n is equal to 2 for the two elements, link 4 and can, and d is the distance from the center of gravity to the joint axis. The calculated results strongly agreed with the Webots returned results as seen from low percentage errors. This validation provided confidence in the physics simulation of Webots.

$$Predicted\ Torque = \sum_{i=1}^{n} F_i d_i \tag{3.10}$$

Figure 3.27: Torque verification using the custom arm

Table 3.1: Torque verification

Can mass(kg)	Predicted torque joint 4 (Nm)	Webots torque joint 4 (Nm)	% Error
1	-2.3500	-2.3701	0.8553
2	-4.5290	-4.5156	-0.2958
3	-6.7067	-6.6191	-1.3061

3.7.3 Collision in MATLAB Space

Collisions occurring in the Webots simulation space help understand how the physics of the robotic arm and objects would be affected. However, it does not specify the specific points in the trajectory of the arm where the collision is happening, nor the links of the robotic arm which are in collision. To find out these trajectory points in the collision and the links of the robotic arm in the collision, MATLAB space is used.

The robotic arms in Webots are represented as meshes defined by the URDF file in MATLAB. And the objects are defined as primitive of the box. The dimensions of

these primitive boxes are obtained from the metadata file. The details of this mapping of robotic arms and objects from Webots space to MATLAB space are discussed in section 2.8.2. MATLAB provides an inbuilt function, *checkCollision* to check for collisions at every point of the inverse kinematics solution. In MATLAB, the robotic arm is moved through all the trajectory points and the minimum distance between the arm mesh and the objects meshes is found. Figure 3.28 shows how the minimum distance is defined for two convex shapes. If this minimum distance becomes negative, a collision is detected.

This function is limited to detecting collisions between objects of convex geometries only and is not reliable for a minimum distance below 10µm between two meshes.

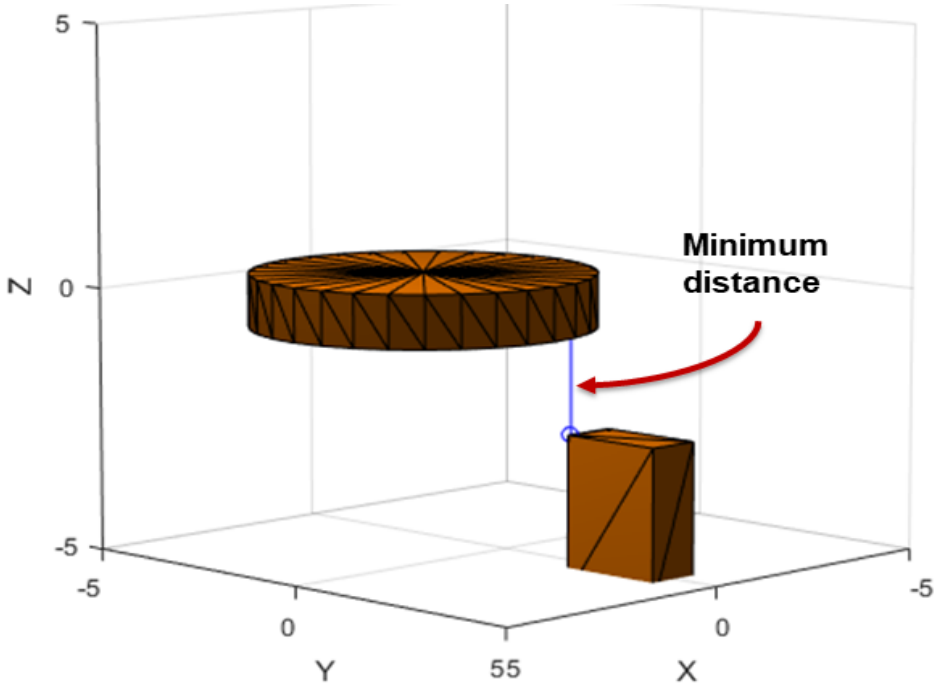
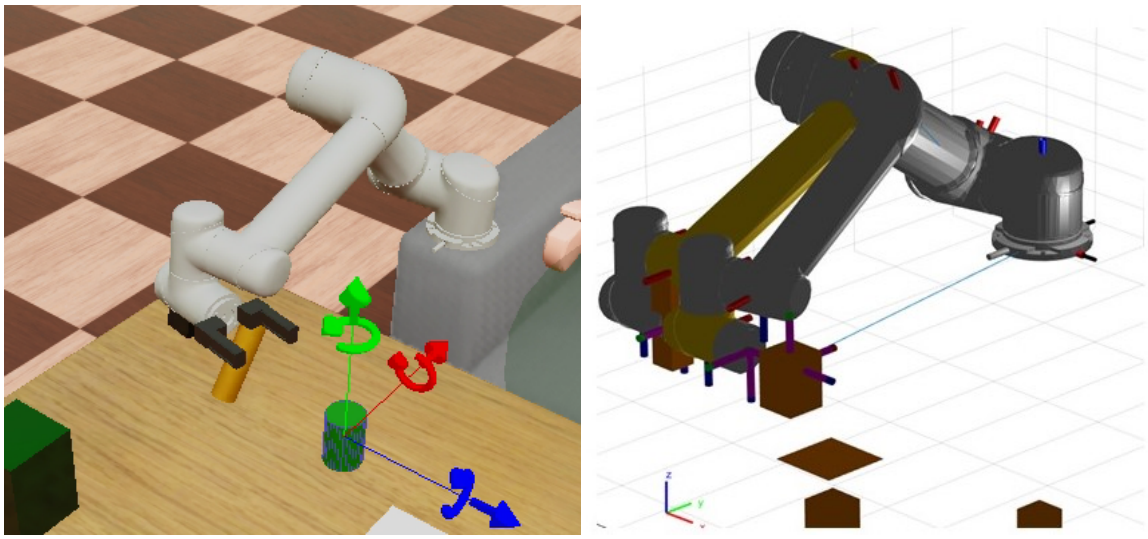


Figure 3.28: Objects in MATLAB space [1]

Figure 3.29 shows an example of collision in both Webots space and MATLAB space. In this example, UR5e is reaching for the green object but collides with the orange object in its trajectory. MATLAB space highlights the links of the arm in a collision and two trajectory points in a collision. In future steps of this research, this information may be used to avoid collision by solving for a better trajectory to reach an object without collision.



(a) Webots space (b) MATLAB space

Figure 3.29: Collisions in Webots and MATLAB

In this chapter, the various object manipulation processes were discussed. This included approaching and picking up an object, bringing the object to the user, placing the object down on the table. The two types of gripper approaches, i.e. axis approach and displacement approach were also described for grasping objects of circular and non-circular cross-sections. The importance of grasping height with respect to the variable vertical cross-section of an object was explained. Finally, the performance of the two robotic arms was compared. This chapter also discussed how Webots

can provide reliable physics results from the simulation. The information necessary for physics simulation was explained in detail which included essential nodes, and the physics information about a robotic arm in a URDF file. An example scenario of object pickup was described and torques returned for the custom arm and UR5e were presented. A validation experiment was performed that provided confidence in the obtained results. Lastly, the collision detection process in MATLAB was explained.

CHAPTER 4

RESULTS, DISCUSSIONS, AND EXPERIMENTAL IMPLEMENTATION

This chapter provides a summary of results obtained from this research. Furthermore, the mapping of the virtual simulation of a robotic arm onto a hardware setup is shown experimentally. The simulation is run on the custom arm and the result is mapped to four servo motors that run on an Arduino interface with MATLAB. Similar mapping can be done with the UR5e arm with six servo motors. This hardware implementation aims to show that the simulation results can be given to a real set of hardware.

4.1 Results and discussion

The simulation environment of Webots provided a reliable platform to simulate and visualize the assistive wheelchair with robotic arms and its environment with various objects. The assistive wheelchair was built with the base of the PR2 robot and chair, which were available in the inbuilt library of Webots. The robotic arms, UR5e and custom arm, were imported externally. This versatile nature of Webots allowed the building of a custom simulation environment that was needed for this research. Webots interfaced with MATLAB, which enabled the use of MATLAB's robotic systems toolbox and all its features. Details of this are discussed in chapter 2. Thus, Webots can be used to further research more into HRI.

Two different types of gripper approaches were implemented based on the geometry of objects, i.e. the displacement approach and the axis approach. It was observed that the displacement approach was sufficient for grasping objects with cir-

cular cross-sections only, while the axis approach was good for grasping objects with circular as well as non-circular cross-sections. The detailed information about this can be found in section 3.1.

The objects in the environment were broadly classified as visible and hidden based on if they were in the view of the overhead camera. Depending on this classification, two different processes were implemented for the pick-up of an object. For picking up a visible object, only the information from the overhead camera was used. And for picking up an object that was hidden from the overhead camera, an additional camera attached to UR5e was used. The UR5e camera was thus primarily used for finding the location and orientation of hidden objects. The details of these two pick-up processes are discussed in sections 3.2 and 3.3. For the process of bringing objects to the user, another camera was used, called the side camera, which provided information about the location and orientation of the user's face. The details of this process are discussed in section 3.4. Thus, in total, 3 cameras were used for the assistive wheelchair, which provided the capability to grasp a visible as well as a hidden object and bring it to the user.

The performance of the two robotic arms used, i.e. the UR5e and the custom-designed, was compared. The custom arm had 4 degrees of freedom, which limited its capability. In particular, the custom arm was ineffective for approaching an object along one of its axes and was only successful for executing the displacement approach. This limitation of the custom arm was overcome by the UR5e arm which had 6 degrees of freedom. With two additional degrees of freedom over the custom arm, the UR5e was successful in executing both kinds of approaches towards an object, i.e. the axis approach and the displacement approach. The details of this comparison can be found in section 3.6.

The physics of the simulation was obtained from Webots physics engine and verified with manual calculations. These physics torques obtained from the Webots guide selection of hardware motor. In addition to detecting collisions in Webots, collisions were also detected in the MATLAB space with the help of the inbuilt function of the robotic systems toolbox. This provided the trajectory points and links in the collision for the motion of robotic arms. The details of this can be found in section 3.7. In future research steps, this information may be used to perform collision avoidance and thus verifying the motion of the arm before passing it to the hardware.

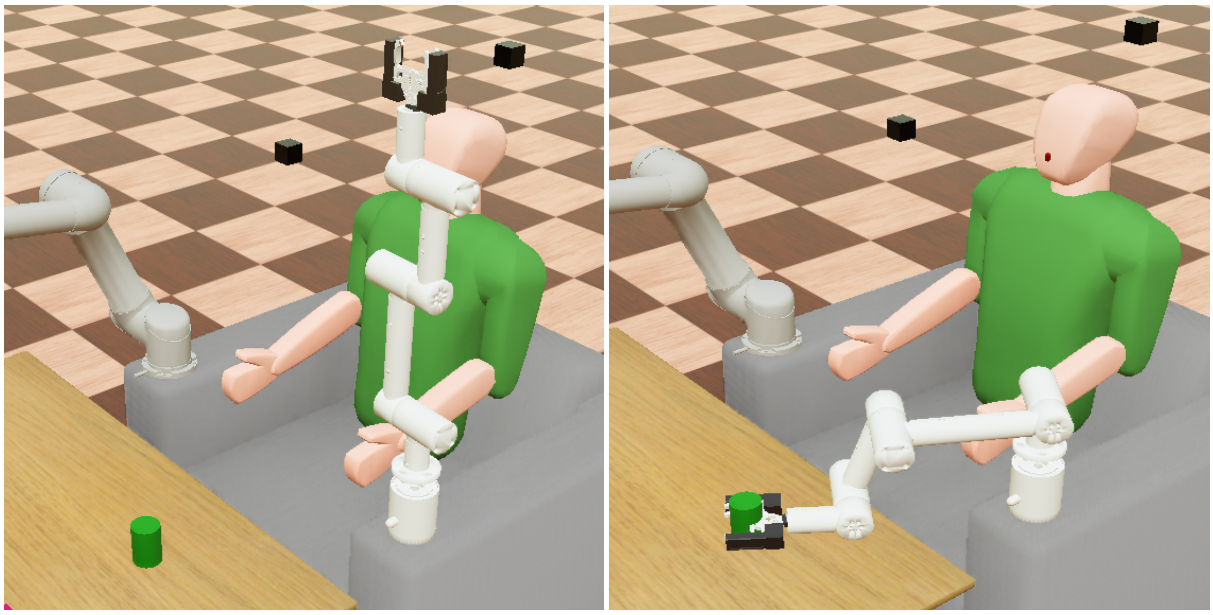
In the following section, a simple Arduino experiment is designed and performed on the hardware of 4 servo motors to show that the results of the simulation can easily be transferred to the hardware environment.

4.2 Experimental implementation of the simulation

The robotic arm simulated is the custom arm which has four revolute joints. The task of this arm is to reach and grab an object on the table within its workspace. The simulation is run on Webots and the inverse kinematics solution is saved into a text file for hardware implementation later. Figure 4.1 shows the initial and final position of the custom arm while performing the pick-up task. Table 4.1 shows the initial and final joint angles as returned by the inverse kinematics solver from MATLAB. These are the angles that are exported to the servo motors via an Arduino interface.

Table 4.1: Solution of inverse kinematics solver

Joint Number	Initial angle (rad)	Final angle (rad)	Servo input command
Joint 1	0	2.49	0.7930
Joint 2	0	1.62	0.5159
Joint 3	0	1.10	0.3503
Joint 4	0	-1.12	0.3567



(a) Initial position (b) Final position

Figure 4.1: Simulation of custom arm for object pick up

An Arduino interface with MATLAB is used to run the simulation results on a set of 4 servo motors, each representing a revolute joint of the custom arm. The rotation of a servo is represented on a scale from 0 to 1, which is provided as a command for a servo to rotate as shown in table 4.1, called servo input command. For the servos used in this experiment, a 0 rotation command represented a 0-radian angle and a 1 rotation command represented a π radian angle. Thus the joint angles are mapped in such a way that 0 radians represent a servo position of 0, and π radians represent a servo position of 1. Equation 4.1 shows how the servo input angle is calculated from the final joint angle that is provided by the simulation. As seen from the equation, modulus of the final angle is taken, which makes the servo input command a positive value. This modulus is taken because servo input can only be positive values between 0 and 1. Thus, negative joint angles are mapped as positive servo rotation. In a joint motor which can rotate in both the positive and negative

directions, this conversion to positive input would not be required. Thus, both the positive and the negative rotation commands can be given to the joint.

$$Servo\ input\ command\ =\ \frac{|Final\ angle|}{\pi} \tag{4.1}$$

Figure 4.2 shows the setup of 4 servo motors that are connected to an Arduino board. A servo motor only provides position control and no velocity control. Thus, this experiment can only verify that the final rotation angle of a servo is representing the final angle given by the inverse kinematics solution.

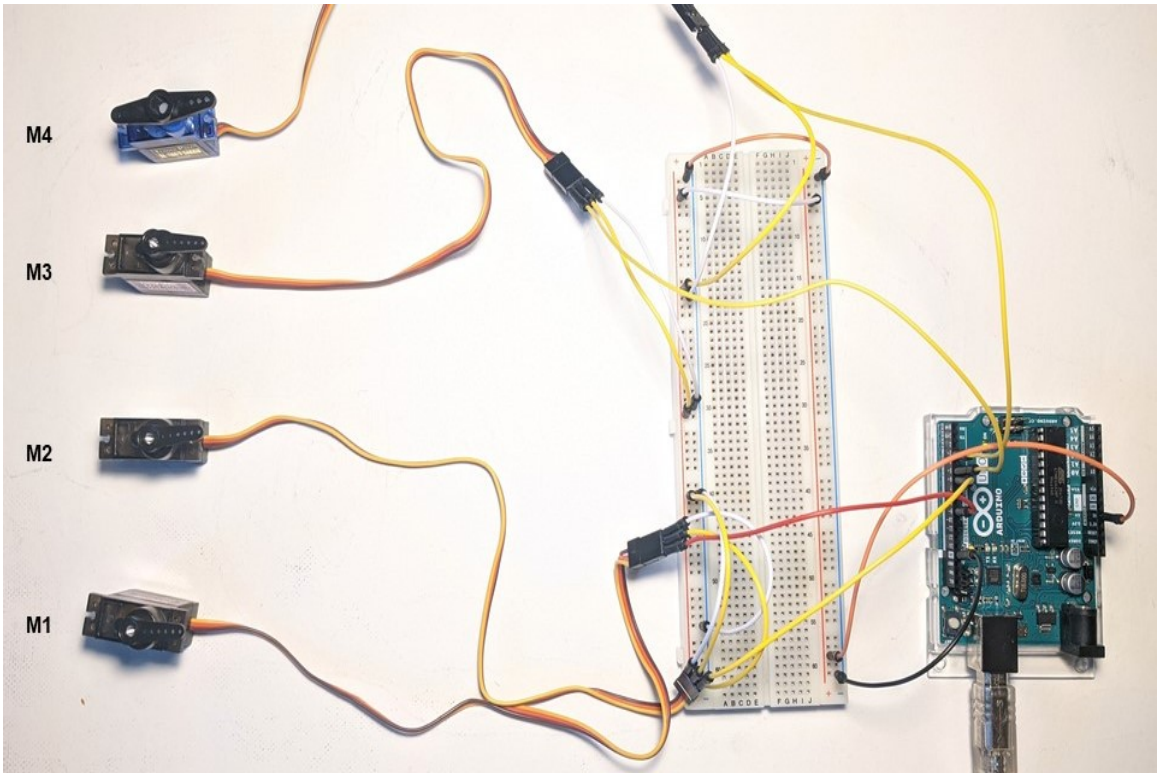
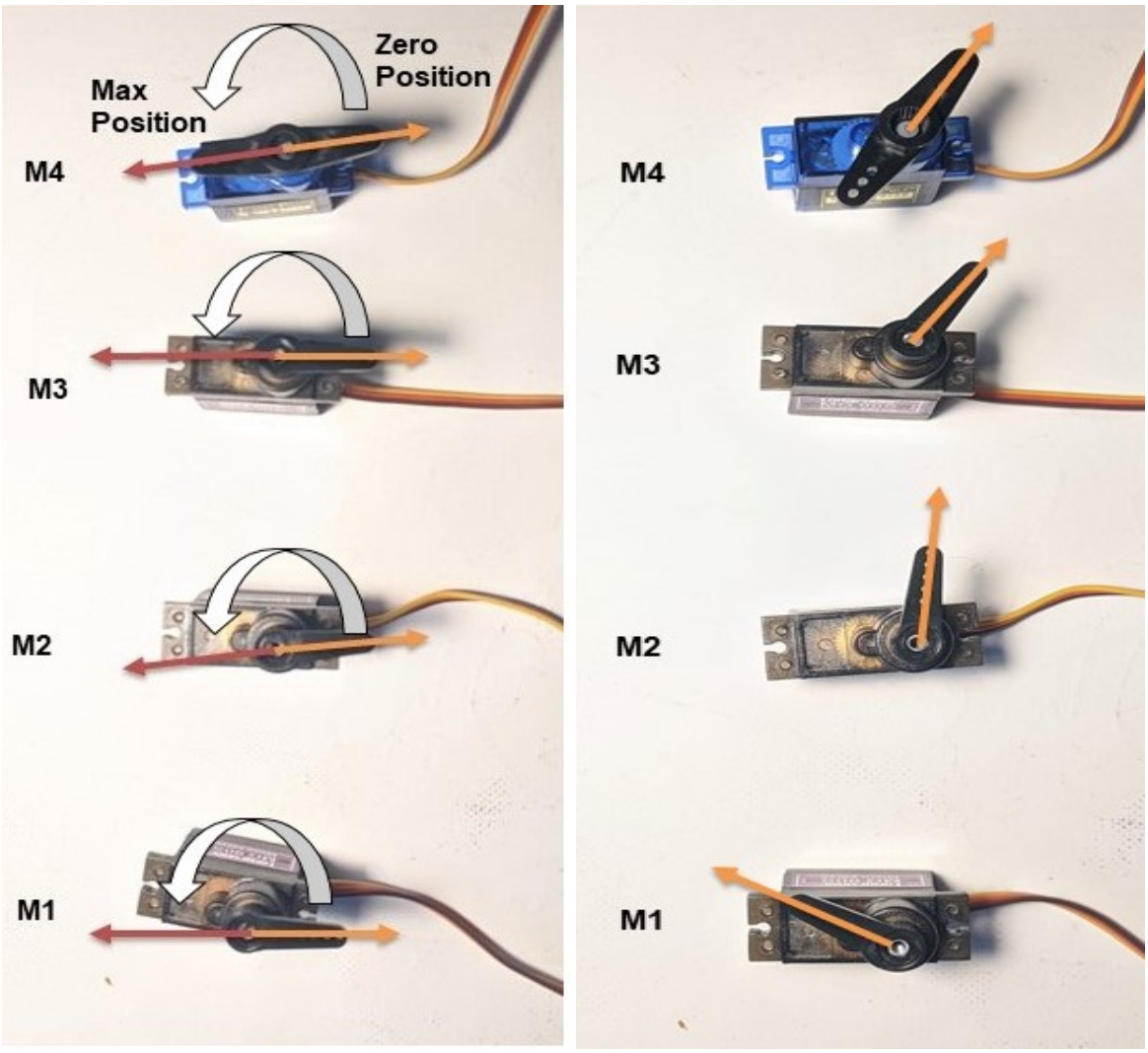


Figure 4.2: Servo motors setup with Arduino

Figure 4.3a shows the 0(minimum) and 1(maximum) positions that a servo can achieve and the direction of rotation indicated by the curved arrows. The 0 positions

is indicated with orange arrows and the 1 position is indicated as a red arrow. Figure 4.3b shows the final position of the servo motors. As seen in the figure, the servo achieves the expected amount of rotation as servo input shown in table 4.1. Motor 1 rotates 0.7930 fractions out of the maximum rotation of 1. Similarly, motors 2, 3, and 4 rotate 0.5159, 0.3503, 0.3567 fractions of the maximum rotation. Thus, it has been shown that the simulation results can be transferred to a hardware setup.



(a) Zero and Max position of servos (b) Final position of servos

Figure 4.3: Servo positions

In this chapter, a summary of experimental results was given. And it was shown how the simulation results are transferable to a setup of real hardware. This simple experiment provides confidence for the hardware implementation of this simulation environment.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The objective of this research was to develop a virtual robotic simulation for an assistive wheelchair and evaluate various object manipulation processes before implementation in an actual hardware environment. To perform this task, Webots was chosen as the simulation platform and an environment with an assistive wheelchair with two attached robotic arms was modeled. The two robotic arms were UR5e and a custom-designed arm with four degrees of freedom. The modeled assistive wheelchair also included three cameras for detecting objects in the environment.

Various object manipulation processes were evaluated. These included pick up, drop off, and bringing of an object to the user. With the help of the built simulation environment, all these processes were successfully tested. The process of picking up visible and hidden objects was implemented. For the process of picking up the visible objects, only the overhead camera was used. While for the process of picking up objects which were hidden from the primary overhead camera, the UR5e attached camera was used and was observed to work for successful pick up of such hidden objects. Furthermore, two types of gripper approaches towards an object were tested in the algorithms, i.e. an axis approach and a displacement approach. It was observed that the axis approach was preferred for picking up objects with non-circular cross-sections and the displacement approach was preferred for picking up objects with circular cross-sections.

A Metadata file consisting of predefined information about the objects in the environment was created. This file was used for determining the grasping height for objects with varying cross-sections along their vertical axis, and mapping objects from the Webots simulation space to MATLAB space. The collisions were detected in both the Webots space and MATLAB space.

The physics of the simulation, i.e. the torque requirements for all joints of the robotic arms were collected for various processes. This physics collection aimed to properly select the motors of appropriate torque for the real hardware implementation.

5.2 Recommendations for future work

The interaction modality implemented in this research was a keyboard for controlling the HRI simulation of an assistive wheelchair. However, there are various modalities that are studied for the field of human-robot interaction [26] [27] [28] [11]. Some of these include electroencephalogram (EEG) signals from the brain, voice commands, and haptic commands. The algorithm implemented in this research is capable of implementing and testing these different modalities in place of the keyboard with slight modification to the controller code.

The virtual simulation built allows for quick testing of algorithms on the robotic arms of the wheelchair. This research implemented a part of an algorithm that could detect collisions happening between a robotic arm and objects and between the links of a robotic arm itself and outputs the trajectory points that were in collision. This environment and algorithm developed can further be modified to include the capability to avoid collisions in addition to detecting them. There are various research studies performed which study the implementation of collision avoidance algorithms and controllers for robotic systems [29] [30] [31]. Since the controller code for the

built environment can be written in multiple programming languages, such studies can be tested and be further developed in this environment.

The algorithm developed and implemented in this research can easily be applied to a different robotic arm with slight modification. Thus, the capabilities of various robotic arms can be evaluated for the specific application of assistive wheelchairs before implementation in hardware.

The trajectory generated by the inverse kinematics solver in this research was carried out in the simulation environment by a proportional controller. The Webots simulation provides the feature of testing the generated trajectory on controllers such as PI, PD, and PID. These controllers can be developed and implemented for obtaining smooth trajectory, desired damping or overshoot, and time constant for the arm motion.

APPENDIX A

Main Controller Code

```
% uncomment the next two lines if you want to use
% MATLAB's desktop to interact with the controller:
desktop;
keyboard;

SetPaths; % getting addresses of necessary files(motion files
    )

TIME_STEP = 64; % speed of the while loop in millisecond

wb_keyboard_enable(TIME_STEP) % Enable use of keyboard in
    simulation

GetDevices(); % get the wheels of chair activated for
    simulation

CallMotors(); % get the motors of the arm activated for
    simulation

CallCamera(); % get the camera activated for simulation

% main loop:
% perform simulation steps of TIME_STEP milliseconds
% and leave the loop when Webots signals the termination
while wb_robot_step(TIME_STEP) ~= -1
```

```
22
23      n = input('Keyboard modality=1 \nEmotiv modality=2 \
            nVoice modality=3 \nSelect the modality: ');
24
25      switch n
26          case 1
27              KeyboardModality(); % Enter Keyboard modality
28          case 2
29              EmotivModality(); % Empty 06/07/2021
30          case 3
31              VoiceModality(); % Empty 06/07/2021
32          otherwise
33              disp('Please select correct modality')
34      end
35
36    % if your code plots some graphics, it needs to flushed
          like this:
37    drawnow;
38
39  end
40
41  % cleanup code goes here: write data to files, etc.
```

APPENDIX B

Webots Interface

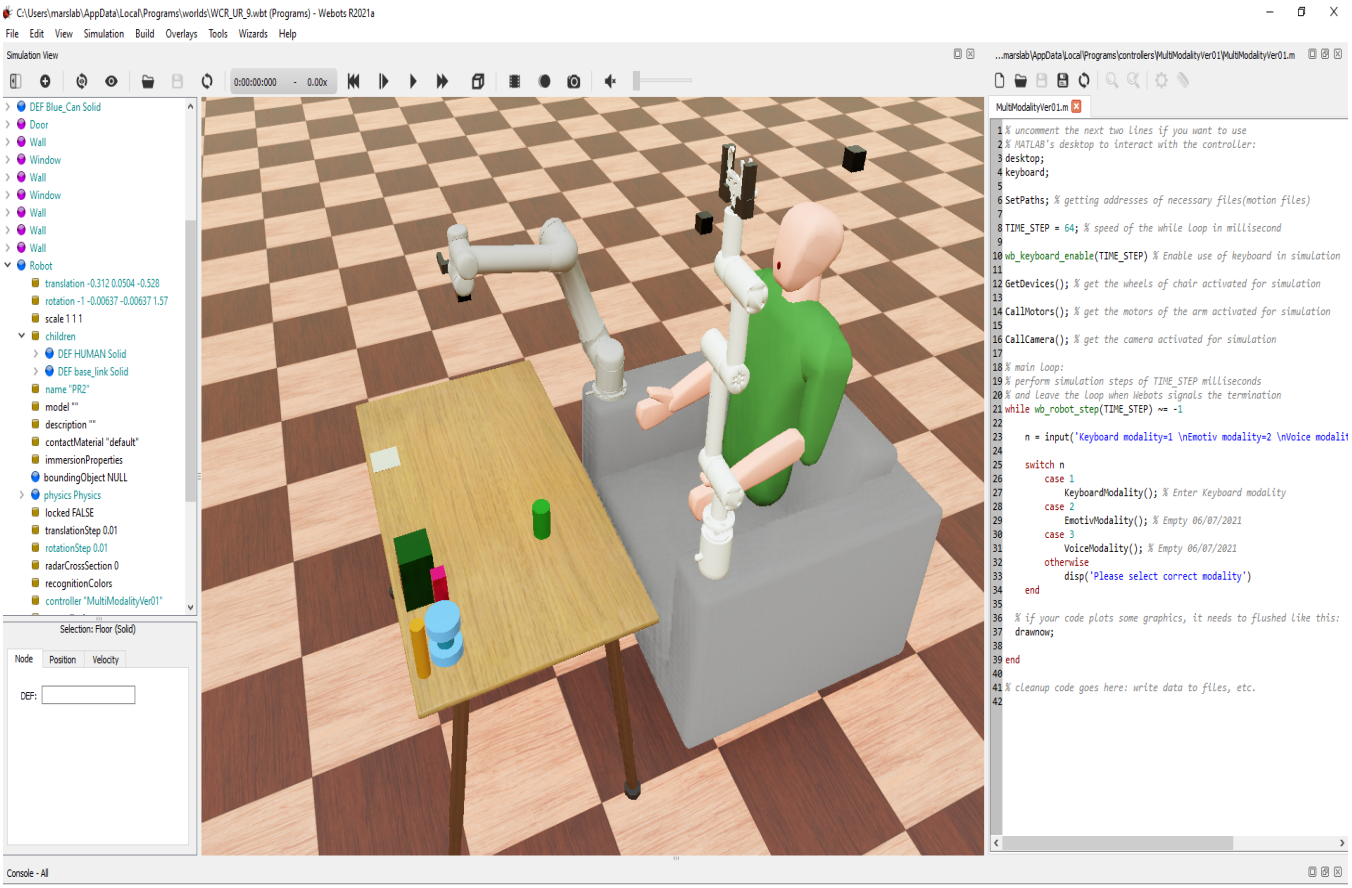


Figure B.1: Webots Interface

REFERENCES

[1] (2021) Matlab robotic system toolbox. [Online]. Available: https://www.mathworks.com/products/robotics

[2] M. A. Goodrich and A. C. Schultz, Eds., Human-Robot Interaction: a survey. Boston - Delft: now, 2007.

[3] S. Lemaignan, M. Warnier, A. Sisbot, A. Clodic, and R. Alami, "Artificial cognition for social human–robot interaction: An implementation," Artificial Intelligence, vol. 247, pp. 45–69, June 2017.

[4] M. A. Sassi, M. J.-D. Otis, and A. Campeau-Lecours, "Active stability observer using artificial neural network for intuitive physical human–robot interaction," International journal of advanced robotic systems, vol. 14, August 2017.

[5] S. Fani, S. Ciotti, M. G. Catalano, G. Grioli, A. Tognetti, G. Valenza, A. Ajoudani, and M. Bianchi, "Simplifying telerobotics: Wearability and teleimpedance improves human-robot interactions in teleoperation," IEEE robotics automation magazine, vol. 25, March 2018.

[6] Y. Wu, P. Balatt, M. Lorenzini, F. Zhao, W. Kim, and A. Ajoudani, "A teleoperation interface for loco-manipulation control of mobile collaborative robotic assistant," IEEE robotics and automation letters, vol. 4, October 2019.

[7] P. Polygerinos, N. Correll, S. A. Morin, B. Mosadegh, C. D. Onal, K. Petersen, M. Cianchetti, M. T. Tolley, and R. F. Shepherd, "Soft robotics: Review of fluid-driven intrinsically soft devices;manufacturing, sensing, control, and applications in human-robot interaction," Advanced engineering materials, vol. 19, December 2019.

[8] L. Cappello, J. T. Meyer, K. C. Galloway, J. D. Peisner, R. Granberry, D. A. Wagner, S. Engelhardt, S. Paganoni, and C. J. Walsh, "Assisting hand function after spinal cord injury with a fabric-based soft robotic glove," Journal of neuroengineering and rehabilitation, vol. 15, June 2018.

[9] A. G. Kravets, Ed., Robotics: Industry 4.0 Issues New Intelligent Control Paradigms. vol. 272: Springer, 2020.

[10] J. P. Vasconez, G. A. Kantor, and F. A. Cheein, "Human–robot interaction in agriculture: A survey and current challenges," Biosystems Engineering, vol. 179, pp. 35–48, March 2019.

[11] R. Patel, "Human robot interaction with cloud assisted voice control and vision system," M. S. thesis, The University of Texas at Arlington, Arlington, Texas, May 2018.

[12] U. Shah, "Human robot interaction using knowledge base approach," M. S. thesis, The University of Texas at Arlington, Arlington, Texas, May 2016.

[13] A. H. A. Rahaman, "On the development of a human thumb tracking device for telemanipulation," M. S. thesis, The University of Texas at Arlington, Arlington, Texas, May 2018.

[14] (2021) Gazebo. [Online]. Available: http://gazebosim.org/

[15] (2021) Webots: Open source robotic simulator. [Online]. Available: https://www.cyberbotics.com/

[16] (2021) Coppeliasim. [Online]. Available: https://www.coppeliarobotics.com//

[17] B. Zhang and P. Liu, "Control and benchmarking of a 7-dof robotic arm using gazebo and ros," PeerJ Computer Science, March 2021.

[18] J. Oyekan, M. Farnsworth, W. Hutabarat, D. Miller, and A. Tiwari, "Applying a 6 dof robotic arm and digital twin to automate fan-blade reconditioning for aerospace maintenance, repair, and overhaul," Sensors, vol. 20, August 2020.

[19] M. Ginesi, D. Meli, A. Roberti, N. Sansonetto, and P. Fiorini, "Dynamic movement primitives: Volumetric obstacle avoidance using dynamic potential functions," Journal of Intelligent and Robotic Systems, vol. 101, April 2021.

[20] J. Zhang, W. Li, J. Yu, X. Feng, Q. Zhang, and G. Chen, "Study of manipulator operations maneuvered by a rov in virtual environments," Ocean engineering, vol. 142, September 2017.

[21] J. Zhang, W. Li, J. Yu, Q. Zhang, S. Cui, Y. Li, S. Li, and G. Chen, "Development of a virtual platform for telepresence control of an underwater manipulator mounted on a submersible vehicle," IEEE transactions on industrial electronics, vol. 64, February 2017.

[22] W. Touzout, Y. Benmoussa, D. Benazzouz, E. Moreac, and J.-P. Diguet, "Unmanned surface vehicle energy consumption modelling under various realistic disturbances integrated into simulation environment," Ocean engineering, vol. 222, February 2021.

[23] (2021) Open dynamics engine. [Online]. Available: http://www.ode.org/

[24] (2021) Ur5e collaborative robot arm. [Online]. Available: https://www.universal-robots.com/products/ur5-robot/

[25] J. Craig, Ed., Introduction to robotics: mechanics and control. New Jersey: Pearson Education, 2005.

[26] Y. Mishchenko, M. Kaya, E. Ozbay, and H. Yanar, "Developing a three- to six-state eeg-based brain–computer interface for a virtual robotic manipulator control," IEEE transactions on biomedical engineering, vol. 66, April 2019.

[27] N. Richer, R. J. Downey, D. Hairston, D. P. Ferris, and A. D. Nordin, "Motion and muscle artifact removal validation using an electrical head phantom, robotic motion platform, and dual layer mobile eeg," IEEE transactions on neural systems and rehabilitation engineering, vol. 28, August 2020.

[28] S. Portol´es, G. Borghesan, L. Joyeux, C. Meuleman, D. Stoyanov, S. Ourselin, T. Vercauteren, D. Reynaerts, and V. Poorten, "Evaluation of haptic feedback on bimanually teleoperated laparoscopy for endometriosis surgery," IEEE transactions on biomedical engineering, vol. 66, May 2019.

[29] S. Mauro, S. Pastorelli, and L. Sabatino, "Collision avoidance algorithm for collaborative robotics," International Journal of Automation Technology, vol. 11, pp. 481–489, May 2017.

[30] S. Moe, K. Y. Pettersen, and J. T. Gravdahl, "Set-based collision avoidance applications to robotic systems," Mechatronics, vol. 69, August 2020.

[31] V. Sunkara, A. Chakravarthy, X. Yi, W. Zuo, and Z. Chen, "Cooperative optimal collision avoidance laws for a hybrid-tailed robotic fish," IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, vol. 28, July 2020.