Simulator for Articulate Robots

Victor Oloworaran

Supervisor: Professor Thomas Bräunl

GENG5511/GENG5512 Engineering Research Project

Submitted: 7th of June, 2020

Word Count: 7997

Faculty of Engineering and Mathematical Sciences

School of Electrical, Electronic and Computer Engineering

Thesis Declaration

I, Victor Oloworaran, declare that:

This thesis has been composed solely by myself and is solely the result of my own work. This thesis has been submitted to fulfill the requirements of the Master of Professional Engineering course at The University of Western Australia and not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

Date: 7th of June, 2020

Abstract

We currently live in an age where robot use and advancement is flourishing, may it be the physical technology of robots themselves or the extension of their software capabilities. With this, the simulation of robotic behaviour is paramount to efficiently designing and manipulating robots. The increase in robot usage has led to simulation software trending towards improved accessibility for non-specialists. Existing software include established or self-developed robot programming frameworks and complex libraries to encapsulate robot control theory. Though still, this can prove complicated to use for those without knowledge of robotics or programming.

This thesis paper presents a standalone articulate robot simulation system with an additional interface to a real small robot manipulator. This software package seeks to provide a free, small, and self-contained simulation system available for multiple platforms which is more suitable for educational purposes or informal small robot development. The system provides a convenient and easy to use experience, simulating robots which can easily be manipulated through the use of the graphical user interface. Furthermore, the simulation will also implement an interface to operate a real small robot manipulator to demonstrate the use of robot simulation for physical robots.

The simulator is based on creating an updated version of the application RoboSim, made in 1996. Implemented in Unity3D, the presented system replicates the movements of any user modelled articulate robot, accounting for physical interactions, and allows for manipulation of this robot. To that extent, a physical robot was constructed in reality and modelled in the simulator, which will be detailed in this paper. Denavit–Hartenberg parameters are used to fully define a robot pose and calculate transformations needed for each robot link in order to manipulate the robot. To test the quality of the system, the simulation of the physical arm will be completed and compared to results gained from the real-world robot through various tests and criteria.

Acknowledgements

For allowing me to complete, and providing guidance throughout my thesis, I’d like to thank Professor Thomas Bräunl. Additionally, I would also like to thank Chao Zhang for kindly agreeing to print all the 3D parts I needed.

Contents

List of Figures

List of Tables

Abbreviations

DH – Denavit-Hartenberg

IK – Inverse Kinematics

ODE – Open Dynamics Engine

OLP – Offline Programming

DOF - Degree of Freedom

GUI – Graphical User Interface

1 Introduction

1.1 Articulate Robots

No matter where we look in modern society, the influence of robots is ubiquitous. From basic robots who perform the same repeated action, such as a welding robot created by FANUC [1], to robots able to perform complex human-like activities such as cooking [2]. Ever since the creation of the first industrial robot, the Unimate, in 1961 to perform welding on car bodies for the General Motors assembly Line, robots have increasingly begun to take part in all aspect of modern life.

Based on the International Federation of Robotics and the ISO (International Organisation for Standardisation) standard the definition of a robot is an automatically controlled, reprogrammable multipurpose manipulator programmable in three or more axes [3].

In particular, one class of robot are articulate or industrial robots which contain rotary joints [3]. In general, these robots contain a number of either rotary or prismatic (changing in length) joints plus an end effector. The end effector will sit at the last joint on the robot and its' position depends on the orientation of every joint preceding it. To manipulate the external environment, the end effector will consist of some kind of tool such as a screwdriver attachment or a gripper for grasping objects. Between robots joints are robot links, which connect joints together. In practice, articulate robots tend to be robotic arms with six degrees of freedom (six joints).

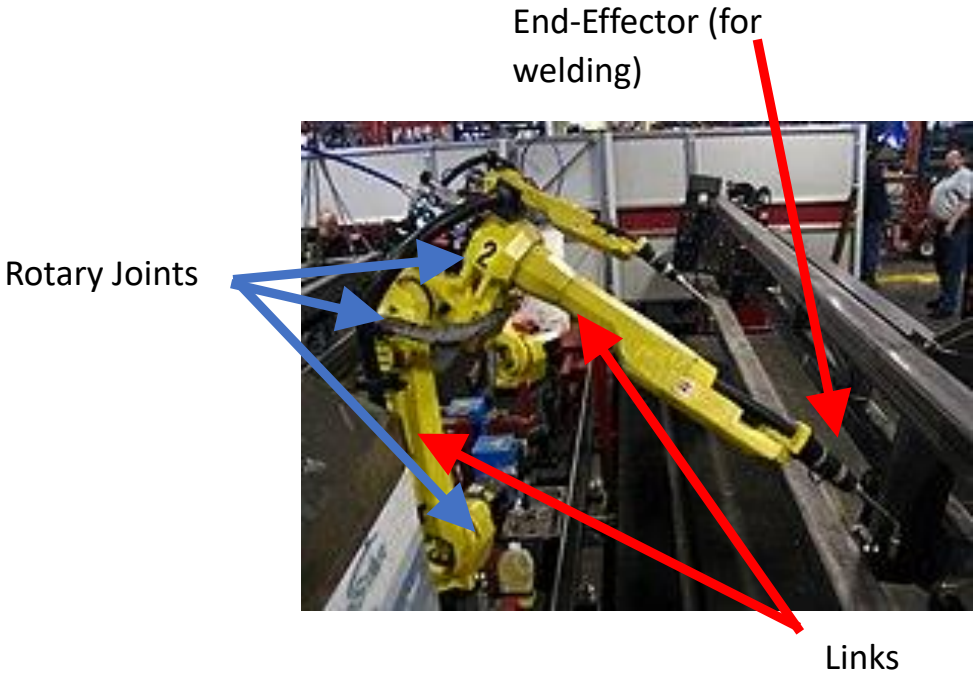


Figure 1: Articulate welding robot created by FANUC[1]

1.2 Robot Simulation

Simulation is the process of modelling a virtual counterpart to a real object or system and replicating its function in a virtual environment for analysing the output or results. The model aims to mimic reality as closely as possible and allows for one to gather data on a physical system without access to

the real system, or for a real system to exist. The very first robot simulations were introduced in the 1990s and included software such as RRS [4]. These were targeted towards industrial working robots [4] (the first type of widely used robot [5]) and were desired mainly by automotive companies and line builders [4]. At the time, robot cycle time (time to complete one loop of operation) and robot movement paths frequently deviated from actuality, leading to rework on real systems. Today, robot simulation is much more accurate and applies to a litany of different robots types like mobile and space robots [6]. Companies world-wide and academic researchers work on creating and enhancing them. One of the early main robot simulation systems was Stage [7] a free, 2D robot simulation system that could model mobile robots and sensors.

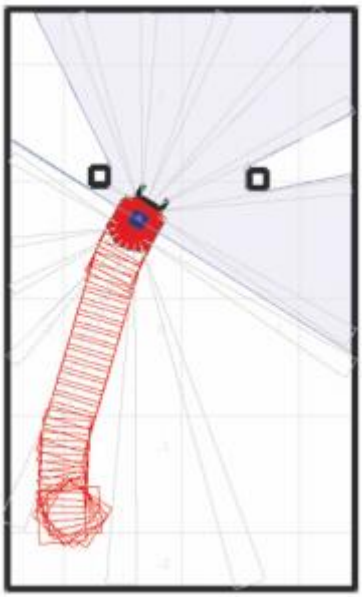


Figure 2: Stage simulation showing a robot (red square) as well as the trail (red hollow squares) and sonar and laser sensors showing the visible region (blue and green regions)[7]

From there, robot simulations have continued to grow in complexity, added various types of robots such as aerial, swimming and humanoid, and size, adding multi robot systems. Many of the largest robot simulators today allow for complex offline programming. Offline programming (OLP) is the ability to program a robot on an external PC and have that program be uploaded to the robot at a later time [6].

Though problems still persist, including the programming and modelling of realistic random errors (relating to motor control, senor accuracy, time delays), system programming [8], accurate modelling of physics for the environment and robots [8] and accuracy of the simulator compared to reality.

1.3 Aim

1.3.1 Motivation

At present, subjects like engineering, computer science/programming and robotics are being integrated into early stages of learning within the Australian curriculum. Simplified robotics allows for students as young as pre-primary to learn how to decompose a complex task into a set of steps [9]. While the curriculum uses the word “programming”, in early years this refers less to learning to code and more so the use of computers to solve a simple task [10].

Current robotic simulation packages mainly focus on use in commercial/research environments, with many aimed towards upper high school and university level users who may have had one or two years of education in the field. Meanwhile, existing education aimed robot simulation systems tend to be

either very basic systems, without modelling of articulate robots, or inaccurate representations of robot theory and many have a significant cost. These include packages such as LEGO MINDSTORMS robots and the virtual robotics toolkit or the root coding robot and iRobot Coding.

It is not feasible for all institutions to acquire the equipment, personnel, or funds to use high class physical robots. As such, a free simulation system which may realistically mimic robots is needed which is low complexity.

1.3.2 Intended Contribution

Thus, this paper proposes a standalone articulate robot simulation system with an additional interface to a real small robot manipulator. This software package seeks to provide a free, small, and self-contained simulation system available for multiple platforms which is more suitable for educational purposes or informal small robot development. The system provides a convenient and easy to use experience, simulating robots which can easily be manipulated both with high-level programming and through the use of the graphical user interface. Furthermore, the simulation also implements an interface to operate a real small robot manipulator to demonstrate the use of robot simulation for physical robots.

2 Literature Review

2.1 Denavit-Hartenberg Parameters

A kinematic chain, which may model a robot manipulator (or articulate robot), is composed of a set of robot links connected together by various joints.

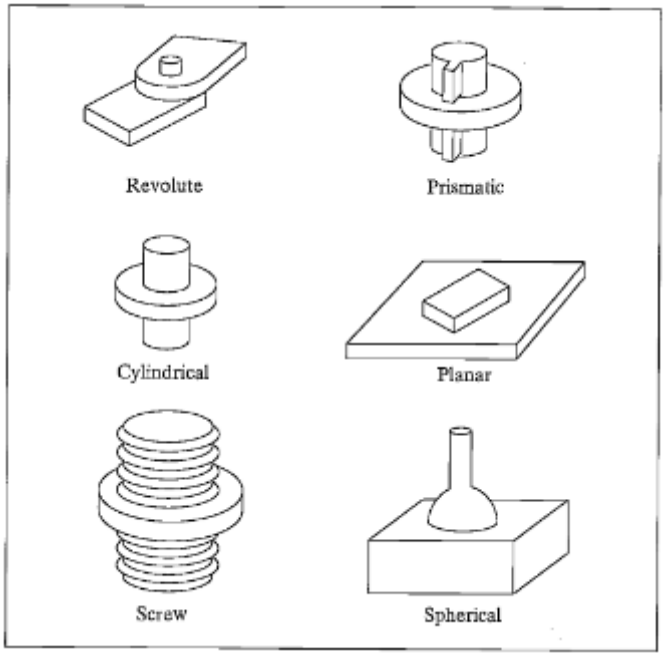


Figure 3: The six possible robot joints [11]

The figure below depicts 2 links in a generic chain, where i represents the link number (with 0 being the very first joint in the chain). It is evident that the position and orientation (pose) of a joint i directly depends on, and builds off of, the pose of the previous joint $i-1$. A compact way to represent this is through assigning each joint its own frame of reference and storing information on the difference between one joint's frame of reference and the next joint's.

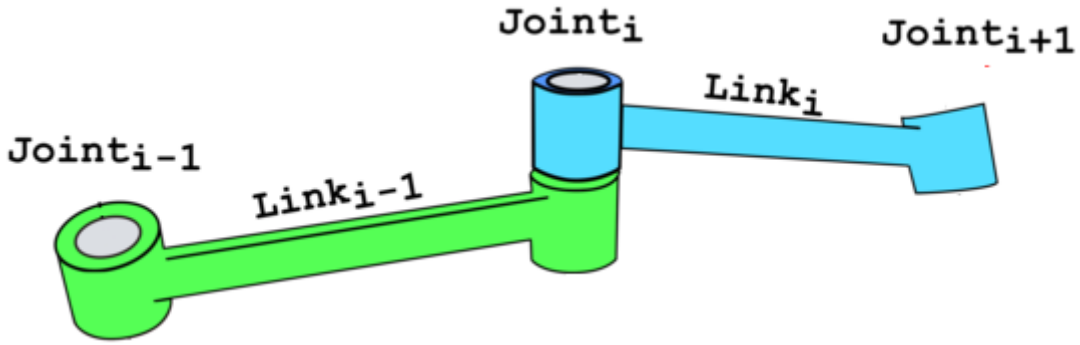


Figure 4: A generic 2-link kinematic chain with three joints [12]

Though other approaches exist to modelling kinematic chains, Denavit-Hartenberg is the most widely exercised in literature and academia [13]. The Denavit-Hartenberg (DH) parameters were developed in 1955 by Jacques Denavit and Richard Hartenberg as a way of assigning coordinate frames to joints [14]. Each joint will have a set of the four DH parameters, and this will define the reference frames for that joint. Using these parameters, one may full specify the relative pose of a joint.

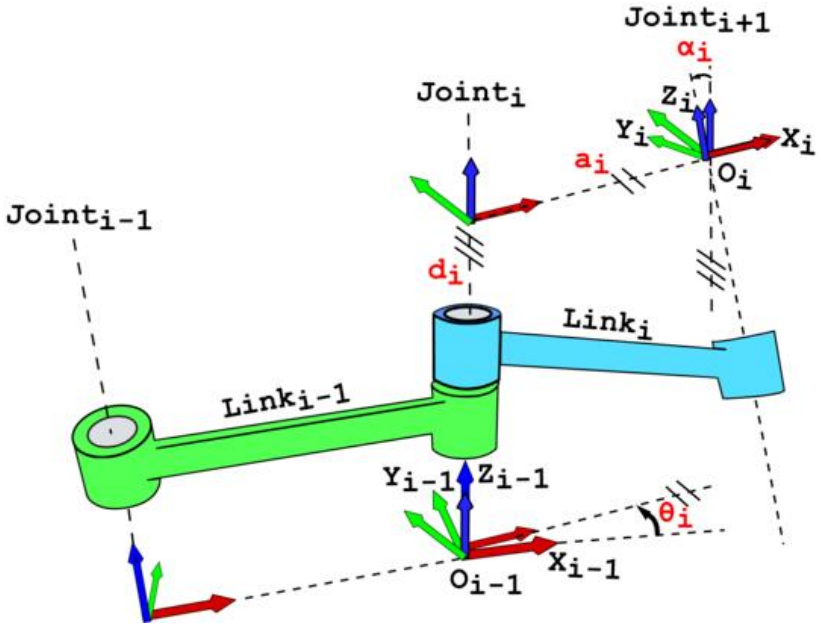


Figure 5: Classic DH parameters and reference frames for a generic 2-link chain [12]

Classic DH parameters will not be explored in lieu of modified DH parameters which have a slightly different (more consistent) method of assigning coordinate frames. The four parameters are:

- The hinge angle, θ, the angle between this joint's X-axis and the last joint's X-axis
- The link length, a, length of the common normal between this joint's Z-axis and the last joint's Z-axis
- The link offset, d, offset of this joint compared to the last, along this joint's Z-axis
- The link twist, α, angle between this joint's Z-axis and the last joint's Z-axis

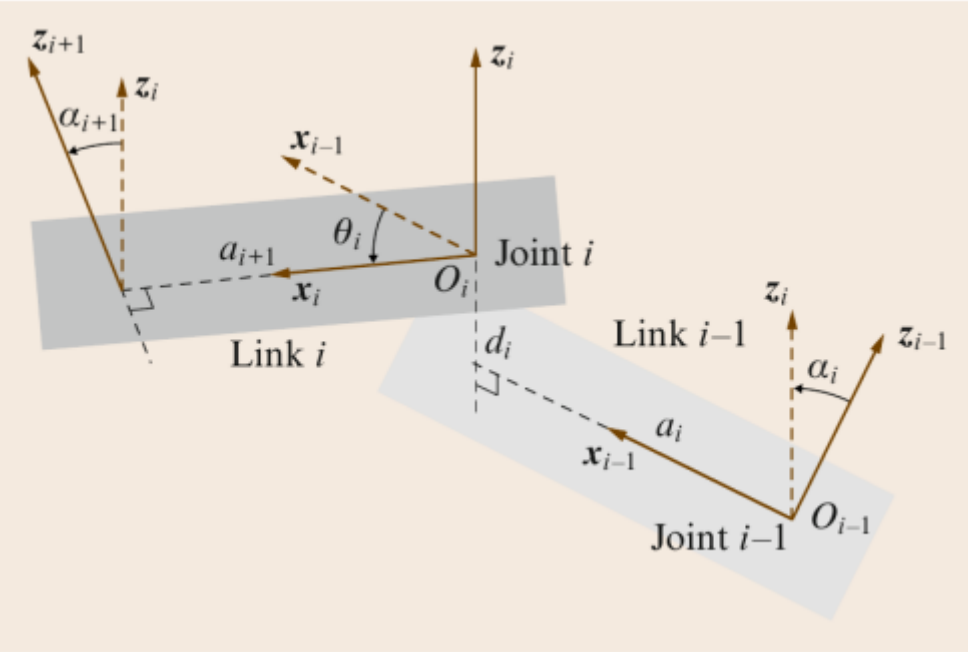


Figure 6: The modified DH parameters [15]

By applying a series of transforms, forward kinematics can be achieved, the act of finding the end-effector (final joint) position from the reference frames of every joint in the chain. The formula for finding the transformation matrix to move a joint to its proper pose based on DH parameters is given below:

$$^{n-1}T_n = \mathrm{Rot}_{x_{n-1}}(\alpha_{n-1}) \cdot \mathrm{Trans}_{x_{n-1}}(a_{n-1}) \cdot \mathrm{Rot}_{z_n}(\theta_n) \cdot \mathrm{Trans}_{z_n}(d_n) \qquad \text{(2.1.1.1) [15]}$$

(where n is equal to the joint number, or *i* previously)

Or in matrix form:

$$^{n-1}T_n = \left[\begin{array}{ccc|c} \cos\theta_n & -\sin\theta_n & 0 & a_{n-1} \\ \sin\theta_n\cos\alpha_{n-1} & \cos\theta_n\cos\alpha_{n-1} & -\sin\alpha_{n-1} & -d_n\sin\alpha_{n-1} \\ \sin\theta_n\sin\alpha_{n-1} & \cos\theta_n\sin\alpha_{n-1} & \cos\alpha_{n-1} & d_n\cos\alpha_{n-1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(2.1.1.2)[15]

While the DH parameters are wide-spread and generally accepted, other reference frame assignment methods do exist. The DH parameters are considered here as they are the most well established and one of the simplest to comprehend but it should be noted other methods, such as screw based methods, may out-perform it in various aspects such as inverse kinematics [13].

2.2 Inverse Kinematics

Inverse kinematics (IK) involves finding every joint pose (defined by DH parameters) needed to place an end effector in a desired position. The IK problem is much more complex and computationally

expensive than the forward counterpart. In general, there are two groups of methods for solving IK, numerical and analytical. Numerical methods often involve some form of iteration, where the end-effector and joints are transformed in steps and tend towards the target position. Conversely, analytical (or closed form) methods deal with devising solutions based on a specific robot's configuration (how the robot is physically defined) and forming a system of linear equations.

Both methods have their drawbacks. Numerical methods are often much more computationally expensive than analytical methods and are prone to being caught in local maxima while tending towards a solution [16]. Analytical method solutions only exist for certain types of robot geometries [11] (where the DOF of the robot is lower than that of the gripper). As there are 6-DOF for general 3D movement, three for rotation, three for translation, many robots are made to be 6-DOF.

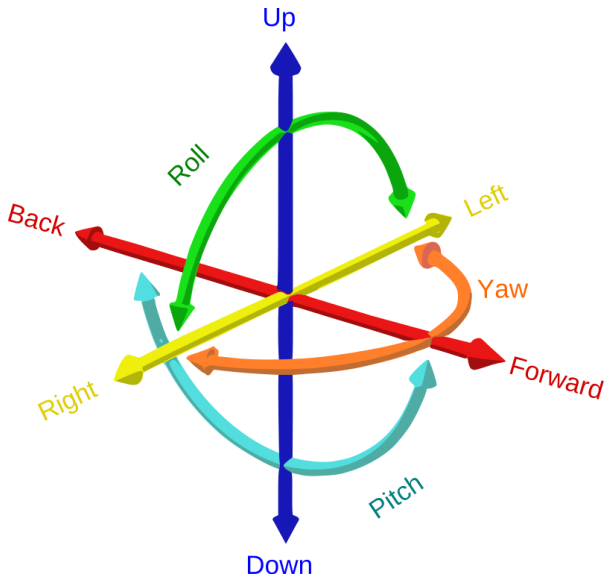


Figure 7: The 6-DOF for 3D space [17]

Due to this, there is no general analytical method as they cannot be applied to all types of articulate robots. In contrast, numerical algorithms can be generalized to accommodate additional constraints and objective functions, compared to analytical solutions which are restricted to 6-DOF systems [16]. Hence, analytical methods are not considered (note many robot manufacturers design robots to conform to the geometry requirements and hence have their own standard equations for IK).

One numerical method for solving the IK problem is through the use of the Jacobian transpose. The use of this matrix has been used in literature relate the force on an end effector to the torque on a joint [18] [19] [20]. Mathematically, the Jacobian inverse is used to solve the IK problem, but this incurs a high computational expense. Instead, the Jacobian transpose is much easier to calculate and serves as a good enough approximation [20].

Instead of considering a force-torque implementation of the IK problem, a more appropriate approach for this paper involves variables more closely aligned with the DH parameters. Hence, Luis Bermudez's IK method for the Unity game engine [21] which instead considers end-effector position and joint rotations is adapted instead.

Let us define T as the pose vector denoting the final orientation of each joint when the end-effector is in the desired position and O as the vector defining the initial orientation of each joint. Then:

$$T = O + dO \qquad (2.1.2.1)$$

Where dO is the change in O needed to reach the orientation solves the IK problem. While O is known, both T and dO are unknown. This equation will be used to iteratively move the robot end-

effector to the desired position, in order to avoid huge oscillations, a multiplier is added to limit the change in orientation, h, where h may be a number manually tuned and generally less than 0.1.

$$T = O + dO * h \quad (2.1.2.2)$$

The Jacobian, J, will be used to relate the spatial position of the end-effector to the angle of each joint. Where for a 3-DOF:

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_A} & \frac{\partial p_x}{\partial \theta_B} & \frac{\partial p_x}{\partial \theta_C} \\ \frac{\partial p_y}{\partial \theta_A} & \frac{\partial p_y}{\partial \theta_B} & \frac{\partial p_y}{\partial \theta_C} \\ \frac{\partial p_z}{\partial \theta_A} & \frac{\partial p_z}{\partial \theta_B} & \frac{\partial p_z}{\partial \theta_C} \end{bmatrix} \quad (2.1.2.3)[21]$$

Here, J relates the hinge angle of joints A, B and C to the position of the end-effector, p, in each axis, X, Y and Z. This can be calculated analytically as such:

$$J = \begin{bmatrix} R_A \times (E - A) & R_B \times (E - B) & R_C \times (E - C) \end{bmatrix} \quad (2.1.2.4)[21]$$

Where R_A is the axis of rotation of joint A (in terms of DH parameters, θ, is a rotation around this joints Z-axis) and likewise for R_B and R_C. E is the end-effector position and A, B, C are the joint positions as vectors.

With the Jacobian acquired it can now be applied to finding dO. If V is the change in spatial position needed to move the end-effector to the desired point, then V is calculated as follows:

$$V = T - E = J * dO \quad (2.1.2.5)$$

As J*dO gives the difference in end effector position for the dO changes in orientation. Now:

$$V = T - E = J * dO \quad (2.1.2.6)$$

$$J^{-1} * V = J^{-1} * J * dO \quad (2.1.2.7)$$

$$J^{-1}V = dO \quad (2.1.2.8)$$

And as noted before, the inverse can be approximated by the Jacobian transpose:

$$J^T V = dO \quad (2.1.2.9)$$

This will provide us with dO, which can then be applied to our robot manipulator and iteratively repeat this process until the end-effector is inside of a certain distance of our desired point.

Problems with the Jacobian transpose method and the optimization-based approaches include the possibility of stops in local minima (minima is used here instead of maxima as the distance to the desired position is minimised). Numerical instability due to poor reliability near singularities of the Jacobian can prevent convergence towards a solution [16]. It is also liable to oscillation around the optimal solution if the step resolution (and hence the distance the end-effector moves each step) is too large. Often multiple optimisation methods are needed to make this method reliable.

2.3 Existing Simulators

Below, currently available robot simulators are considered with their architecture and features, who provide similar functionality to the proposed system. For a more comprehensive list of simulation systems and their features (and comparison to the proposed system) see Appendix A.

2.3.1 Webots

Originally developed in 2004 for Khepera robots [22], Webots is one of the major robot simulations used in research and industry at the moment and recently became free and open source. Webots can model any mobile robot included legged, wheeled and flying [22]. Containing accurate models for sensors and actuators, Webots has been successfully used in industrial applications to accurately model robots [23]. Webots is mainly programmed in C++ and Robots are programmed in C, C++/Java or third-party software such as MATLAB and these programs can be transferred directly onto real robots. The physics engine used is the Open Dynamics Engine (ODE) which allows for accurate simulation of collisions, friction and other forces between the rigid body (a solid body affected by physics but unable to be deformed) robots and the environment. Webots also uses its own 3D rendering engine (WREN) to show 3D environments and robots to the user. Multi-agent simulations and multiple different types of robots as well as customisable environments are all features. Creating and controlling a robot involves forming a chain of servo nodes which can be controlled from a user made controller program. This allows for custom programming of user defined robots. Making the system incredibly flexible. In addition to these features, Webots has toolboxes for complex computing such as artificial intelligence and computer vision [23] and allows users to write their own plugins.

Similarly, to other popular robot simulators, Webots is aimed towards professional programming of robots and academia. This makes it difficult for one without programming experience to pick up easily. In addition, while the package is free, user support or consulting has a fee. Though in terms of accuracy and flexibility, Webots is one of the top-of-the-line simulation systems.

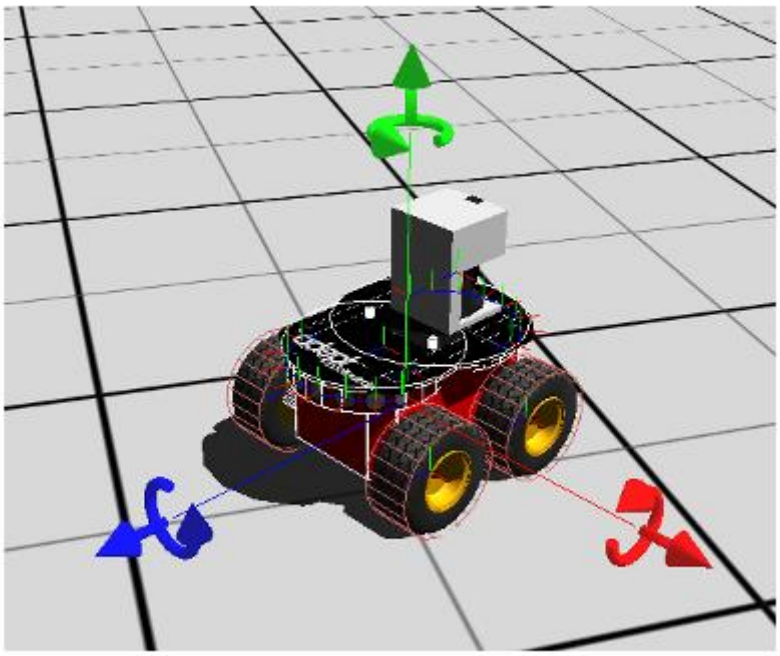


Figure 8: Image of the Webots 3D Scene editor with a mobile robot selected and axis for translating the robot shown [23]

2.3.2 RoboDK

RoboDK is another large, popular robot simulation system, widely used in industry and academia. The biggest difference between RoboDK and Webots is RoboDK is a paid service and as such is much more geared towards industry. RoboDK allows for simulation of industrial robots and allows for programming of various robot controllers including ABB RAPID, Fanuc and Universal Robots [24]. Similarly, to Webots, RoboDK also allows for programming in C/C++, Java and many third-party languages (including manufacturer specific robot programming languages) and allows for user-written plugins. The system was made in Python and uses OpenGL for 3D rendering and the Gravity Plugin for physics. Where it separates itself from Webots is through the ability to perform CAD to Motion functions, allowing for robot machining to be programmed [25]. Examples of RoboDK use involve by NASA to program collaborative robotic arms [26] and Pen State University for education and research [27].

Again, RoboDK is a large, flexible and accurate robot simulation system. With many of the same strengths as Webots and the same flaws. Though it has the added disadvantage of being a paid service.

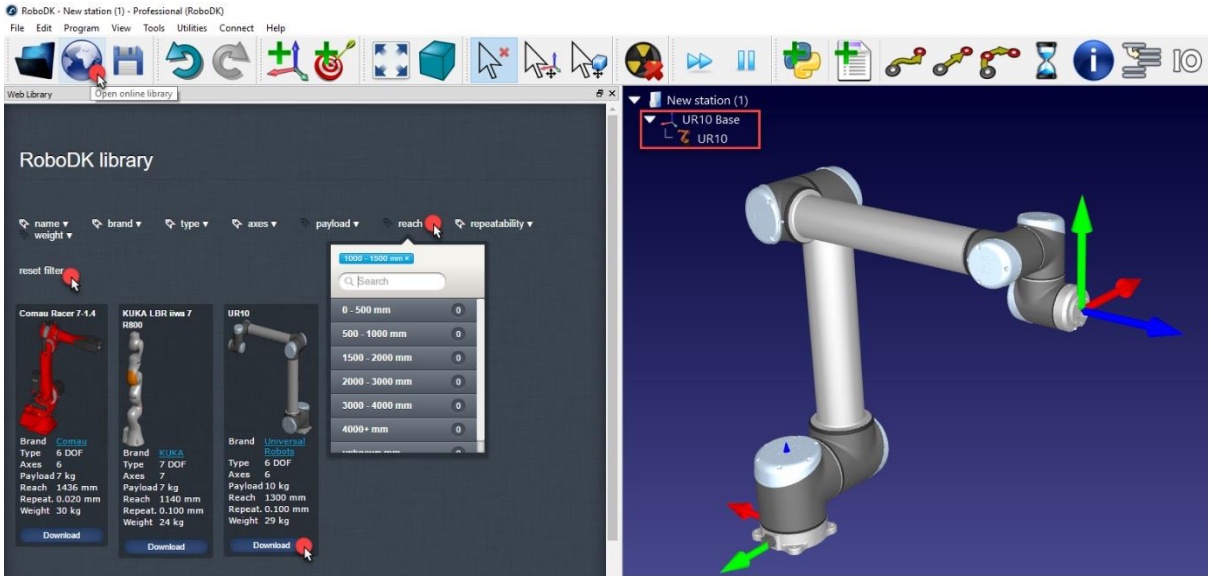


Figure 9: Picture of the RoboDK GUI showing the RoboDK library containing various robot arms that can be loaded into the editor (left) and the RoboDK editor showing a UR10 6-DOF robotic arm [24]

2.3.3 RoboSim

RoboSim is designed for a single pre-defined articulate robot arm that can be manipulated by a user [28]. Created in 1996 and programmed with java and C as both a Java applet and Java application [29] the program was made by Rainer Pollak (original version) and Johannes Schützner (Java version) [30]. RoboSim is the predecessor to the proposed system, forming the outline for what this system aimed to provide. The robot has 6-DOF and consists of 3 links connected by 2 joints to a gripper which may open and close. The system is very limited, it has no physics engine and is rendered as a java applet, collisions are avoided by giving hard-coded limits to robot angles. The system is able to save robot positions and load them, as well as loading of simple programs using its own defined programming language in .file files (see Appendix B for commands). As the robot is a fully defined 6-DOF robot, the system may use analytical inverse kinematics to move the robot to a specified position. Included was a physical counterpart to RoboSim which could be programmed with the same .file files.

RoboSim is a very limited and older system for simulating articulate robots. Though the attractive parts, which were adapted to the proposed system, include its simple design and use of the GUI to control a robot in addition to a simplified programming language. Overall, the system was much too limited to be very useful for applications outside of programming its real counterpart.

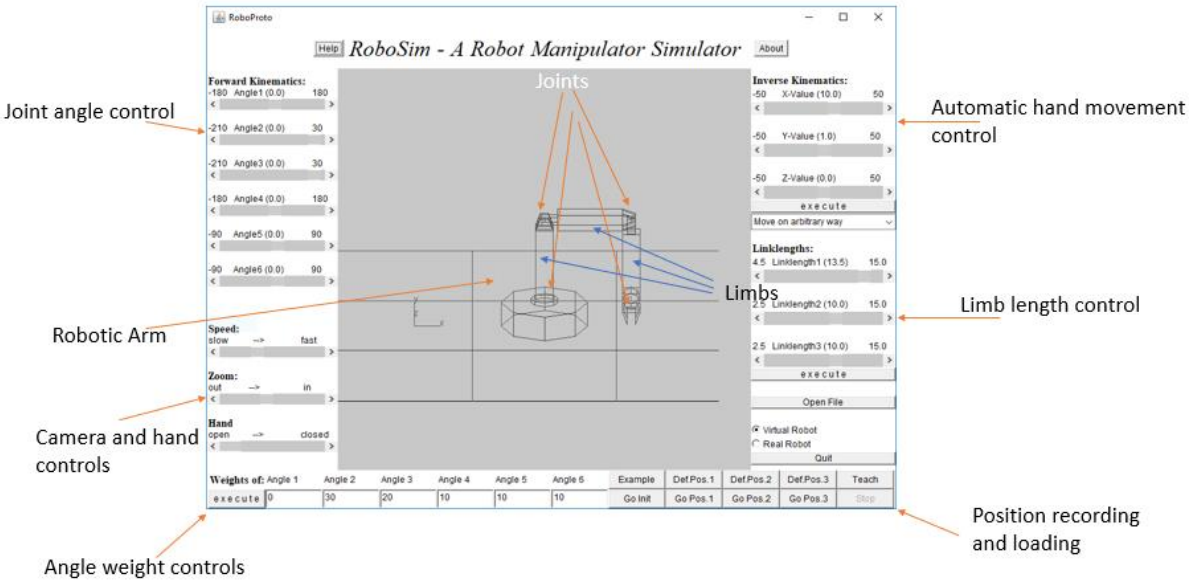


Figure 10: Picture of the RoboSim GUI depicting the robotic arm and user controls

3 Software Framework

3.1 Requirements

In order to achieve a useful articulate robot simulation system that is low complexity, the system aimed to meet the following requirements:

- Users should be able to load in and customise their own articulate robot and see it rendered
- The user should control their robot through the use of a graphical user interface (GUI) and view the resulting transformations through forward kinematics
- The system should include realistic physics and allow for collisions
- The system should implement some form of IK
- The system simulated robots should be able to mimic a real robot arm
- The system shall be free and available on any major platform

3.2 Constraints

Constraints included:

- Cost for real robot parts
- Choice of real robot design (detailed below)
- Computing power, assuming the system should run on any common computer
- Choice of system engine, as it has to be widely supported

3.3 Real Robot

In order to prove the usefulness of the program, a physical robot was needed which could be modelled in the simulation system. A robot design which is not too complex yet adequately shows the capabilities of the system should be chosen and built. When deciding on the specific design to use, prioritise included:

- Moderate price (under 200$)
- Mechanically easy to construct
- Made of readily sourced parts
- At least two limbs and a gripper
- Easily programmable

After considering multiple alternatives the Robot Arm MK2 Plus designed by Jacky Le and found on Thingiverse was used which can be found online at https://www.thingiverse.com/thing:2520572 for free. The final design, specification and parts list of the robot are presented in section 4.

3.4 Resources Used

3.4.1 Unity

Unity is a cross-platform game engine for the creation of 2D or 3D games as well as simulations and animation. Unity allows for a user to create and edit 3D object and environments and allows for scripting using C#. Unity projects are supported by a wide range of platforms including iOs, Andriod, Windows, MacOS, Linux and many more. Unity presents the rendered scene to the user when the game is launched and uses its own physics engine to deal with rigid body objects.

3.4.2 Blender

Blender is a free open source 3D creation suite. It allows for modelling, rigging, animation, simulation, rendering and compositing of 3D meshes. While Unity was also capable of making custom objects though the object editor, it is not well optimised for creating complex meshes and has a much narrower range of accepted file types when importing non-Unity models.

3.4.3 3D Printer and Work Tools

The 3D printer and work tools were needed in the construction of the physical arm. Tools used included screwdrivers, files and a soldering iron.

3.4.4 Non-Convex Mesh Collider

The Non-Convex Mesh Collider is a Unity asset created by Productivity Boost software development [31]. The asset is available on the Unity asset store and allows a user to create concave collision meshes. Unity, natively, only allows for convex collision meshes, meaning any holes in models would be considered as part of their collision mesh. This posed an obvious problem for importing complex robot parts which may have to fit together and hence this Unity asset allows for accurate collision meshes for many varieties of complex objects.

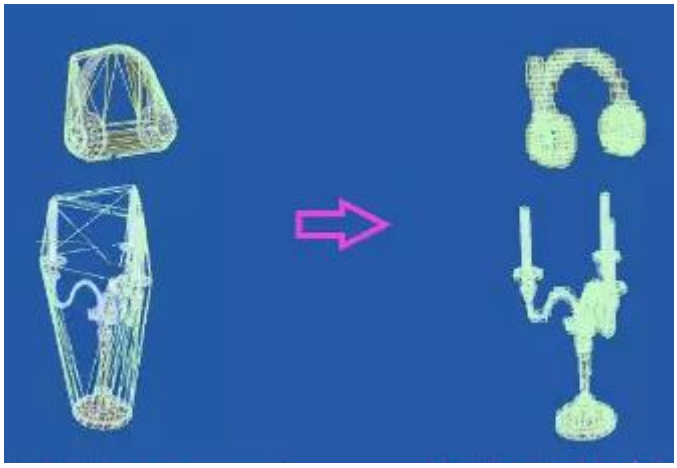


Figure 11: Concave collision mesh for imported objects in Unity (left) vs collision meshes using the Non-Convex Mesh Collider asset (right)[32].

3.5 Design Philosophy

As the goal was creation of a simulator for users new to the field of robotics, the design philosophy reflected the need for simplicity and intuitiveness over technical complexity. First and foremost, the design was meant to be easy to use and understand. Users should be able to use the system with little to no knowledge of robot control theory. In accordance, the system is not designed for rigorous use or commercial applications. There was no requirement to be as accurate or expansive as leading robot simulators, but rather create a system which could display the process of simple robot manipulation. The second highest priority was creating a customisable system, one which was able to take in any valid user model and allow the user to manipulate it and whatever way it was configured. To that end the system was built with little hard limits on robot configuration and the system was made modularly as so it would easily be extensible.

3.6 Metrics Evaluating Success

In order to measure the success of the application, its functionality is tested to ensure the system achieves its goal of creating a useful articulate robot simulation. Hence, the main metric for evaluating success will measure the difference between end-effector positions of the virtual and real counterparts when both are manipulated the same way. To achieve this, both the constructed physical arm and its model in the system have the ability to undergo the same basic commands which are sent from the user and change the robot's orientation. Each movement will be considered in isolation and joints will be tested to expose if there is any difference in the accuracy of different joint types. Measurements for repeatability of the simulation will be included to show if there is any accumulated error or constant error in end-effector position.

As a subset of this, a modelled robot should be to scale and be able to complete the same poses as the physical robot without self-collisions. Therefore, the robot's scale must be tested to ensure it provides an accurate representation of the physical robot. Through these three measurements, a comprehensive metric for how closely the simulation mimics the robot arm will be calculated.

Additionally, IK tests will be done to find the percentage of time the robot is able to reach its desired point. Or conversely, find its position and response to non-reachable points. Two different robots with 2 and 3 links are considered to see the difference in their performance. The robot will start the test from a "untangled" position, meaning the robot will either be straight or close to straight, initially, and then use IK to attempt reach several random positions. The two robots should also be given different start and desired positions as it is impossible to give them the same starting pose (due to the differences in robot configuration) and that pose may influence how easily they reach the desired point.

4 Final Design

4.1.1 General

The final system was created in the Unity game engine and completely coded in C#. The figure below shows the general way the user and the system interact. A user will be able to open the executable, from there they will have access to the system GUI. The user may load in robots from the GUI or send commands to the system. If a user loads a robot, they will need to provide both the models (.obj files) to all parts of the robot and a configuration file (text file) which details how that robot is arranged and joints are configured. These files are then loaded into the system and sent to the game engine which will then load them as objects into the scene.

Once a robot has been loaded a user may manipulate a robot through the GUI, changing the DH parameters of hinge angle or link twist of the robot. These changes are relayed back to the game engine which will recalculate the pose of the robot and send the new pose to the scene for rendering. In this way, users will be able to manipulate the scene in real time.

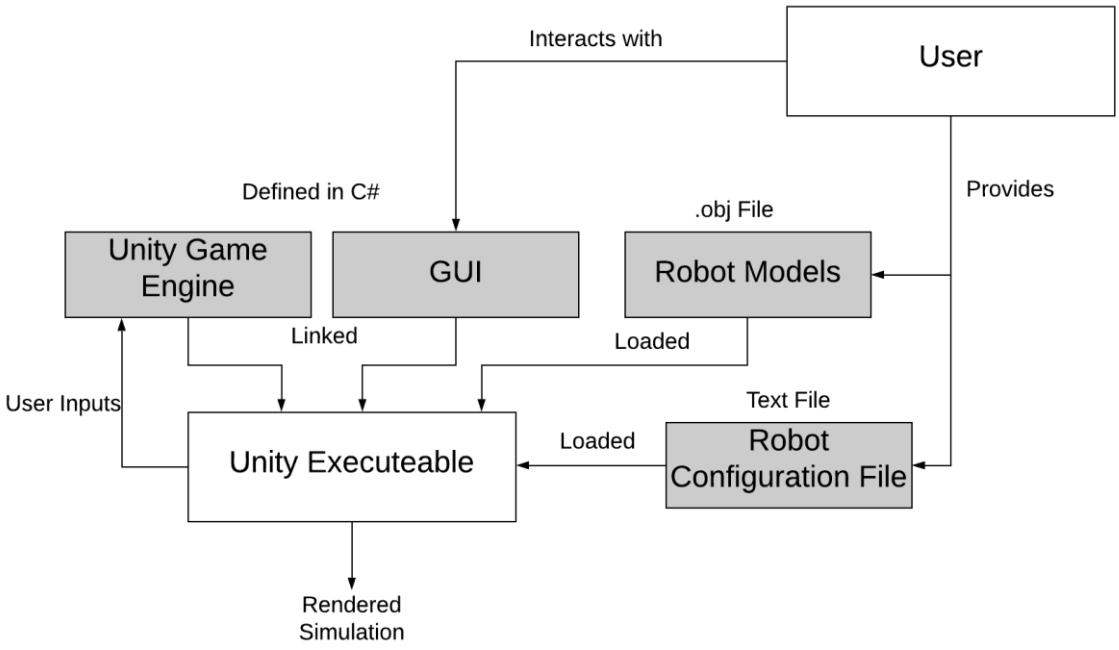


Figure 12: Simplified Block Diagram of Proposed System

4.1.2 Robot and Objects

The robot itself will be composed of five different types of objects, each of a different class. A simple robotic arm, modelled in the simulator, makes up the figure below.

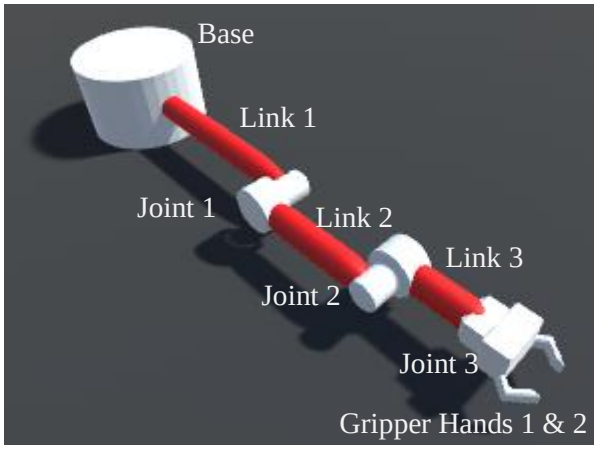


Figure 13: 3-link robot arm, modelled in the proposed system. Annotated with all object classes (note Joint 0 is concealed by the base object).

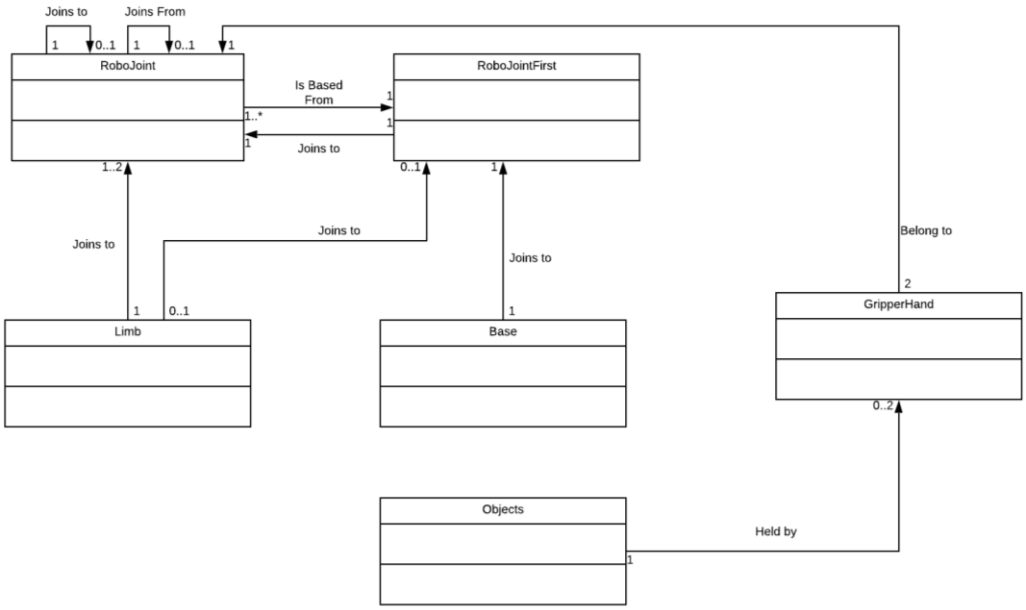


Figure 14: Simplified UML diagram for objects and the robot

Both RoboJoint and RoboJointFirst define behaviour for the robotic joints. The only difference between the two is RoboJointFirst is the first joint in the robot and hence has no previous joint it is positioned relative to. These joints will each contain their own set of DH parameters and will have access to the last joint's DH parameters. By implementing the equations seen in section 2.1.1, the system uses forward kinematics from the DH parameters to fully define the robot pose. Their model is specified by the user.

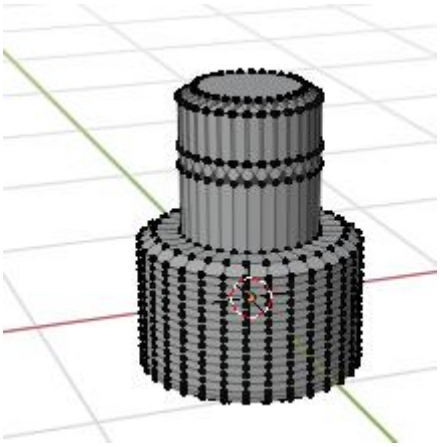


Figure 15: Example robot joint model, made in Blender

Each limb object simply acts to place themselves between their start and end joints. As the joints will be positioned as far apart as their link length parameter dictates, links will automatically fill this distance. As joints translate and rotate, a link will follow the angle of its start joint and place itself in the middle of their start and end joints.

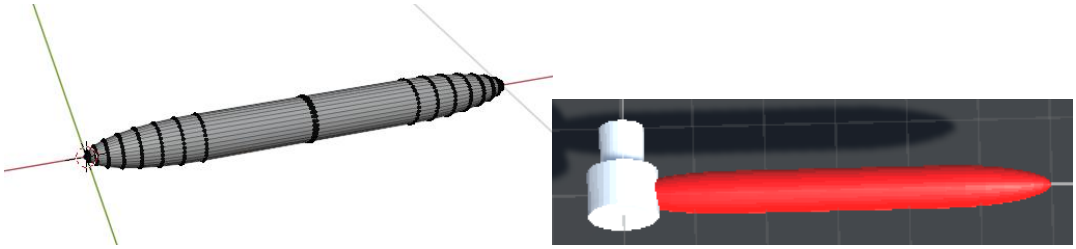


Figure 16: Example robot link model, made in Blender (left) and a robot link and start joint, connected in the simulator (right).

A base object must be defined for all robots, this part anchors the robot to the ground and undergoes no transformations. As there is no need for transformations, this object does not have any DH parameters and is simply a visual formality to make the robot look accurate.

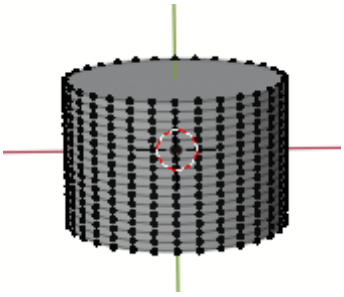


Figure 17: Example robot base model, made in blender

To complete the robot, a pair of gripper hands should be defined (though can be configured to be excluded). These are programmed to place themselves at the last joint, directly under and in front of the joint. Hence, to form the gripper, the last joint model should be for the gripper. The gripper hands initialise as open, separated by the width of the joint. They can then be controlled to either close or open. As the gripper is two pronged, the gripper hands should have two models.

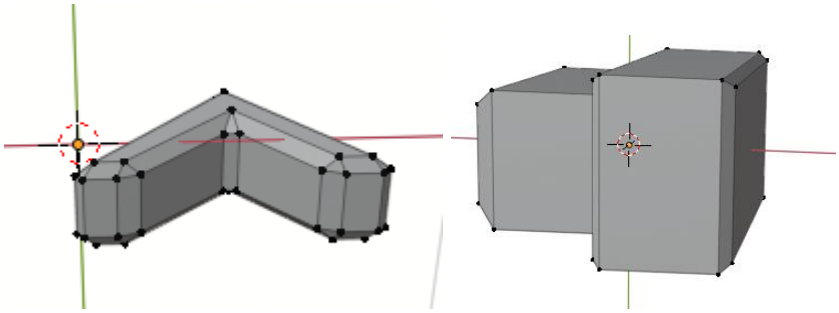


Figure 18: Example gripper hand model (left) and example model for final robot joint (right). Both made in Blender

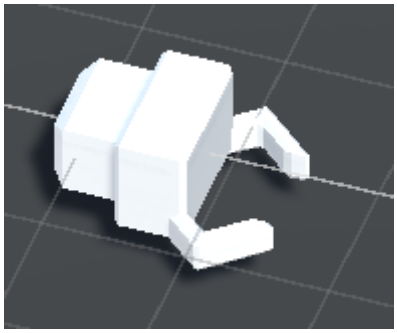


Figure 19: Final gripper with hands and joint, in the simulator

In addition, the object class defines any static objects not connected to the robot. These objects are still physics enabled which allows for the robot to interact with them (through gripping or collisions). They are also able to be dragged around by the mouse so the user can manipulate them easily.

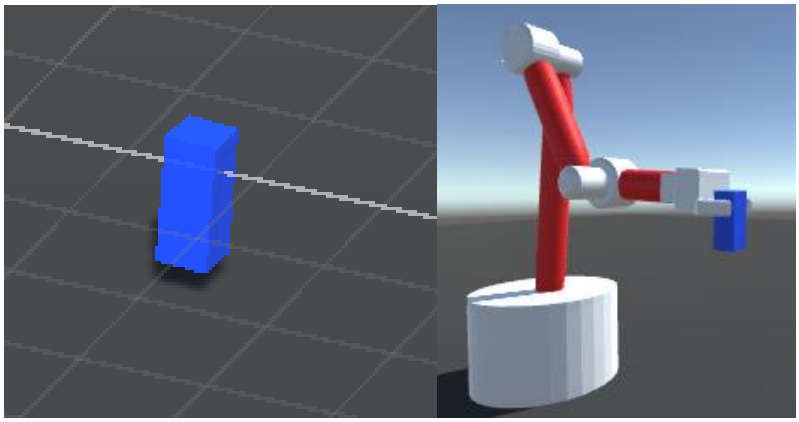


Figure 20: Blue box which is of the object class (left) and a simulated robot arm gripping and holding up the object (right).

Hence, a virtual robot fully specified by its DH parameters has been created. When initialised, robots will be configured according to the user's input file (explained in section 4.2). Here the example robot from the above figures with no rotations is considered to demonstrate its DH table.

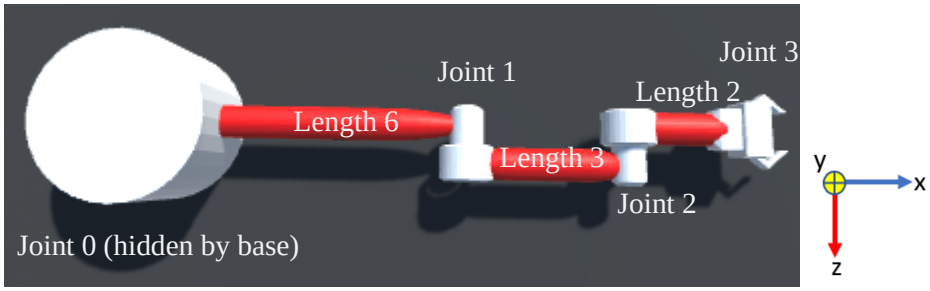


Figure 21: Example robot in initial position. Note the axis on the right, with y coming out of the page.

Joint number	θ	a	d	α
0	0	0	0	0
1	0	6	0.7	0
2	0	3	-0.7	0
3	0	2	0	0

Table 1: Figure 21 corresponding DH table taken from the simulation (bottom).

As seen above, the robot has no rotations in any axis and is facing along the X-axis. The DH parameters also have 0-valued θ (hinge angle) and α (link twist) for each joint. Each link has the corresponding link length (note this defines the length of link connecting to that joint). In addition, joints 1 and 2 have d-values (link offsets) of 0.7 and -0.7 respectively. This causes their displacement along the Z-axis, as seen in Figure 21. For simplicity, this robot initially has all its joint reference frames aligned with the world coordinate system. As a joint is transformed, that frame of reference will transform as well, so a rotation about the hinge angle is always around that joint's Z-axis, not the world's Z-axis.

Once the DH parameters describing orientation are changed, the user will see the robot change its pose accordingly.

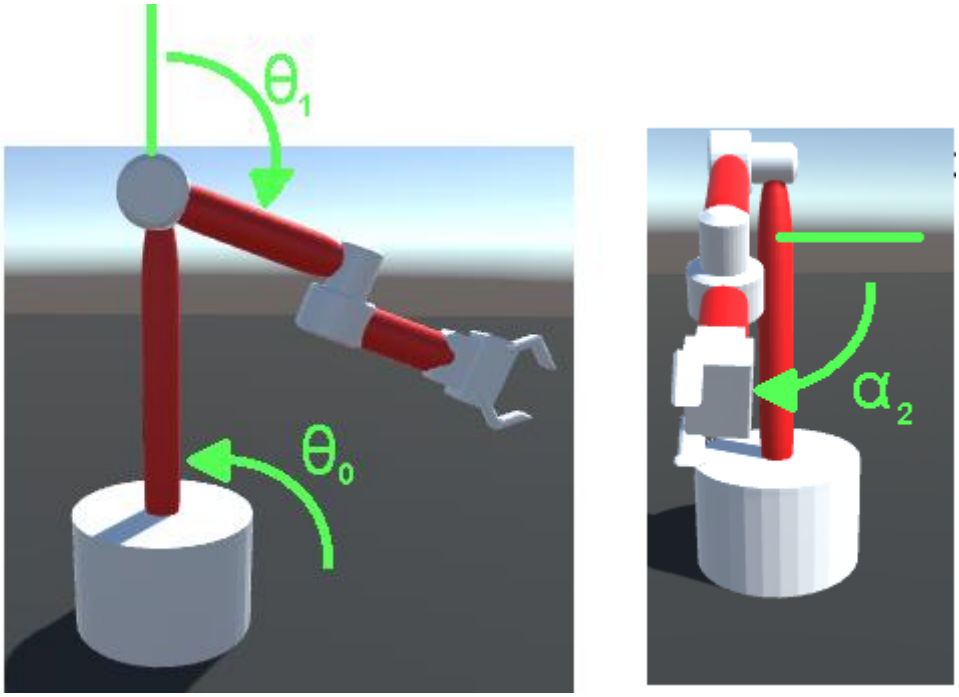


Figure 22: Example robot in rotated position with side view, facing negative z-axis (left) and frontal view, facing negative x-axis (right).

Joint number	θ	a	d	α
0	90	0	0	0
1	-110	6	0.7	0
2	0	3	-0.7	90
3	0	2	0	0

Table 2 : Corresponding DH table taken from the simulation for Figure 22.

Thus, forward kinematics have been achieved as the end-effector (gripper) has been placed in the correct orientation and position based on the previous joints' DH parameters and the robot has been fully specified.

4.1.3 Physics

Every object in the scene is defined as a rigid body, allowing them to experience forces and interact with Unity's physics engine without deformation. The main use of the physics engine is to detect collisions between two objects, but other forces also exist such as gravity. Objects can be collided with, including self-collisions for robots, and pushed.

For the purpose of this simulation, many physics forces (such as friction) exist in the system but are either ignored or unhandled. This simplification is appropriate as the system is not intended for intense analytical use but rather as a demonstration or informal testing of custom robots.

4.1.4 GUI

The GUI provides an intuitive interface for the user to interact with the simulated robot. As the system was aimed towards users without robotic experience, the GUI has been styled simply with a minimal number of sliders and buttons which perform high level functions. These control the functionality to load robots, restart the scene, quit the scene, and invoke inverse kinematics. When the user first loads the program, they will see the following base screen:

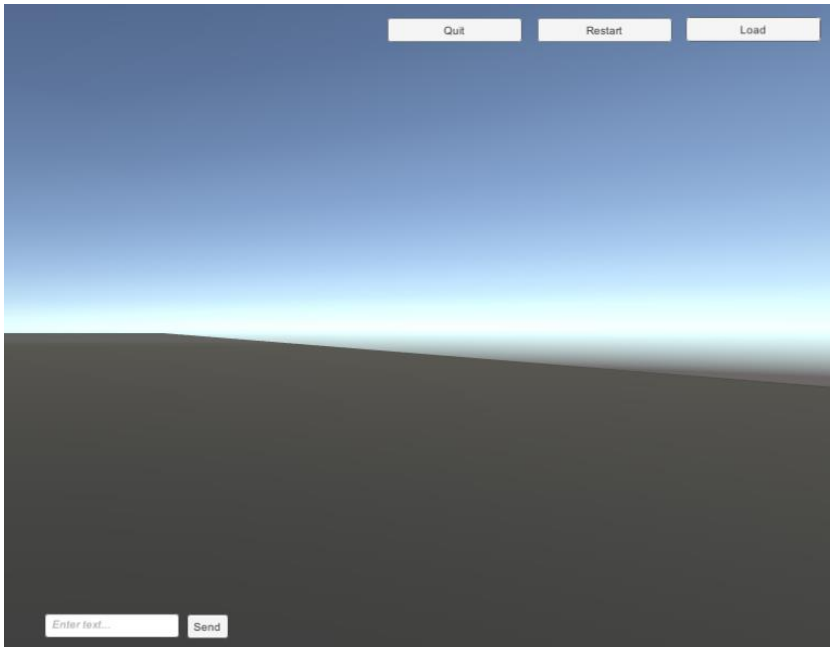


Figure 23: Basic GUI screen

This contains three buttons for different user functions. The quit button will exit the system, while the restart button will delete any previously loaded objects and display Figure 23 to the user once again. The send command feature is used for sending commands to the real robot through a serial port, as well as controlling the robot's virtual counterpart. Refer to section 4.3 for more details.

Pressing the load button will open a file selection window. The user will use this to select their robot configuration text file.

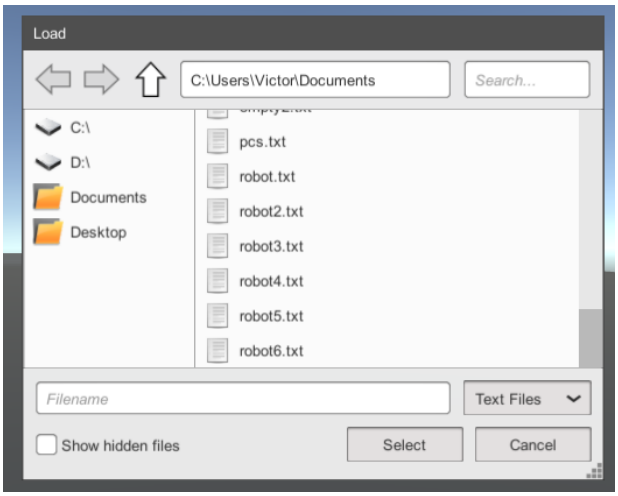


Figure 24: Simulator file selection window

Once a valid robot file is selected and loaded, added controls will be presented to the user in the form of sliders for controlling the robot DH parameters and an inverse kinematics button.

Figure 25: Simulator GUI with DH sliders and IK button

Outside of the GUI, camera navigation with arrow keys is also provided for changing view of the scene.

4.1.5 Inverse Kinematics

IK is implemented using the method described in section 2.12, though it has been adapted in multiple ways to suit this system. Firstly, it has been generalised to any number of joints and also includes calculations for the link twist angle, in addition to the hinge angle as originally defined. Because of this, IK can be performed in three dimensions rather than two (changing hinge angle would only rotate joints in around the X-Y plane).

Currently, the IK algorithm tries to direct the arm to a pre-loaded cube in the scene within the executable system. But this target can be edited within the Unity editor to see IK for any target position in the scene.

4.2 Specification of Robots

Robots will be specified through use of a structured text file. This file will contain 20 sets of inputs including information on how the robot is built, system path to robot .obj file models, its initial DH parameters and frames of reference, and many other parameters about the robot. To view the structure of the document, see Appendix C.

4.3 Real Robot

4.3.1 Construction

The real robot was constructed with intent of having it simulated in the system and compared to for data collection. From here on, the original online design this robot was adapted from will be referred to as the online specification. The only special tool needed for construction was a 3D printer for fabricating all the online model files and a soldering iron for the circuitry. Otherwise all other tools used were commonplace. While the physical design is completely the same as the online specification, the electrical parts were slightly changed which lead to the wiring being reworked. The online specification's program was not used in favour of a custom one that more accurately represented basic robot control. For full construction images (including final set up) see Appendix D.

4.3.2 Specifications

The chosen robot has three DOF, the first joint allows the robot to rotate its base about the Y-axis. The next two joints allow the robot to rotate its arm about the Z-axis. Finally, a two-pronged gripper exists at the end of the last robot limb. Note here, while the point which connects the gripper to the rest of the robot is technically a hinge joint, the gripper hinge is controlled completely mechanically to force it to stay horizontal. Hence, the joint is considered as essentially constrained and not a DOF.

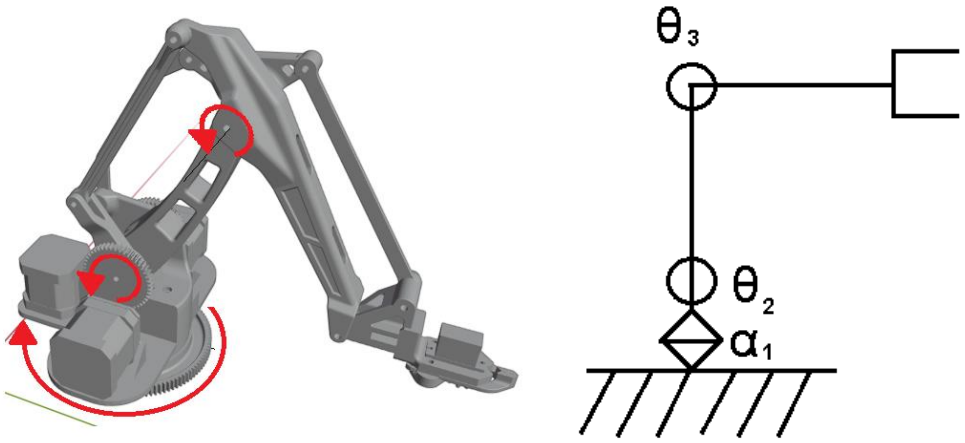


Figure 26: 3D model of the full physical arm source from the online specification, with red arrows showing the robot joints and their axes of rotation (left) and the corresponding robot notation diagram (right)

(Refer to Appendix E for details on robot notation)

The robot will have a total horizontal reach of 38.2cm and total vertical reach of 36cm. See Appendix F. for robot dimensions.

All three robot joints will be controlled from their own stepper motor, while a servo will be incorporated in order to control the gripper fingers. The stepper motors each drive gears that are connected to the robot and this allows them to turn the robot. The servo works similarly, with a gear attached to the servo and to the gripper fingers, as the servo rotates so too do the prongs which can open or close the gripper. The three stepper motors will have their own motor controller modules, and these will be controlled by an Arduino Nano. For a full parts list, refer to Appendix G.

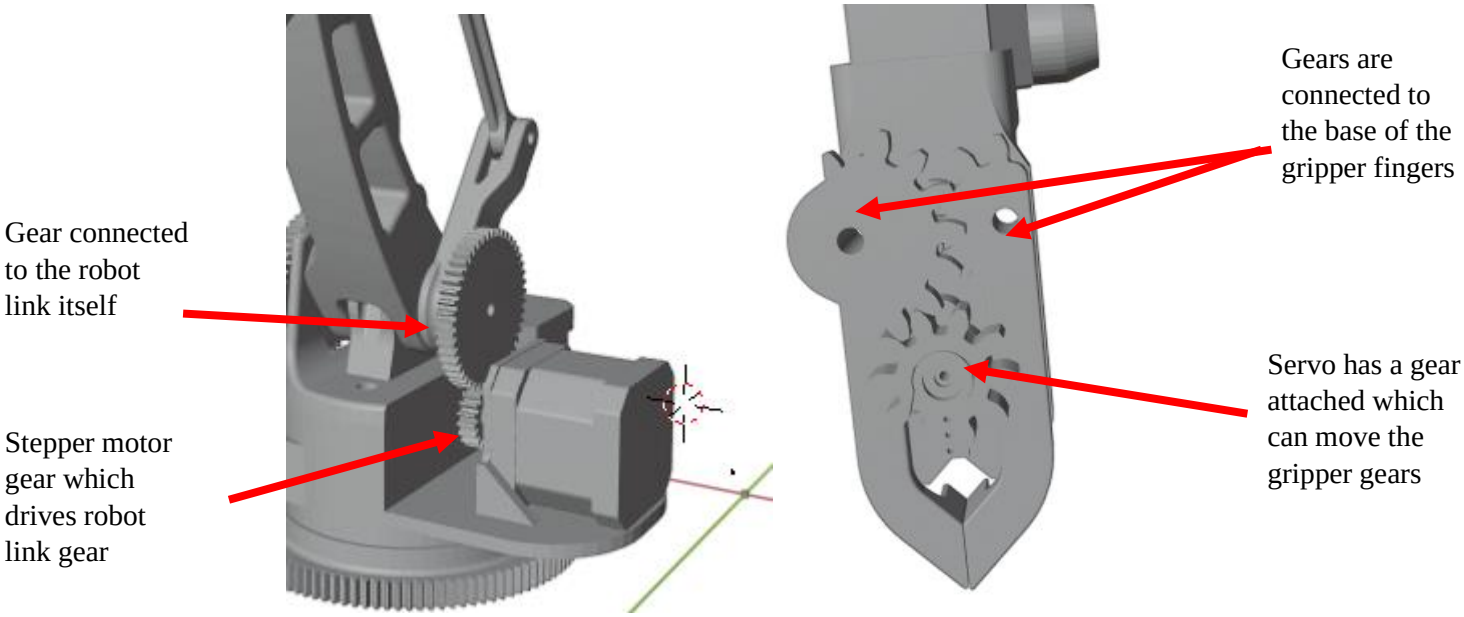


Figure 27: Gears for stepper motor (at side of robot base) (left) and servo (underneath the gripper) (right)

4.3.3 Electrical Wiring and Programming

The system was powered by a 12V DC adapter, as well as through the use of a DC/DC buck converter to gain a 5V supply. The stepper motors needed to operate on the 12V supply as they draw high amounts of power and current. In comparison, the Arduino and servo both operate off 5V inputs. In order to accurately control the stepper motors, L298N modules were incorporated which took input signals from the Arduino and turned the stepper motor accordingly. All components are either soldered together or use female to male wire plugs.

To see the full electrical diagram, please see Appendix H.

The robot was programmed as to constantly read data coming in off a computers serial port which was directly connected to the Arduino through a USB 2.0 cable A-B. Commands it receives must be of the structure:

“L” [joint number] [rotation amount]

Or

“S” [0 or 1]

The first command will rotate a robot joint by a certain amount of degrees (by rotating the stepper motor by a certain number of steps). The second command sends a signal to the servo which either turns the servo counter-clockwise for 1 (which closes the gripper) or clockwise for 0 (which opens the gripper). Any other commands are ignored. The program also keeps track of the current robot angle by recording changes made through commands, though this assumes the robot started from a set position and only moves due to the commands.

4.3.4 Final Product and Virtual Model

By using the models provided in the online specification and using Blender to convert them from .stl files to .obj files, the system was able to recreate the physical arm in a virtual environment.

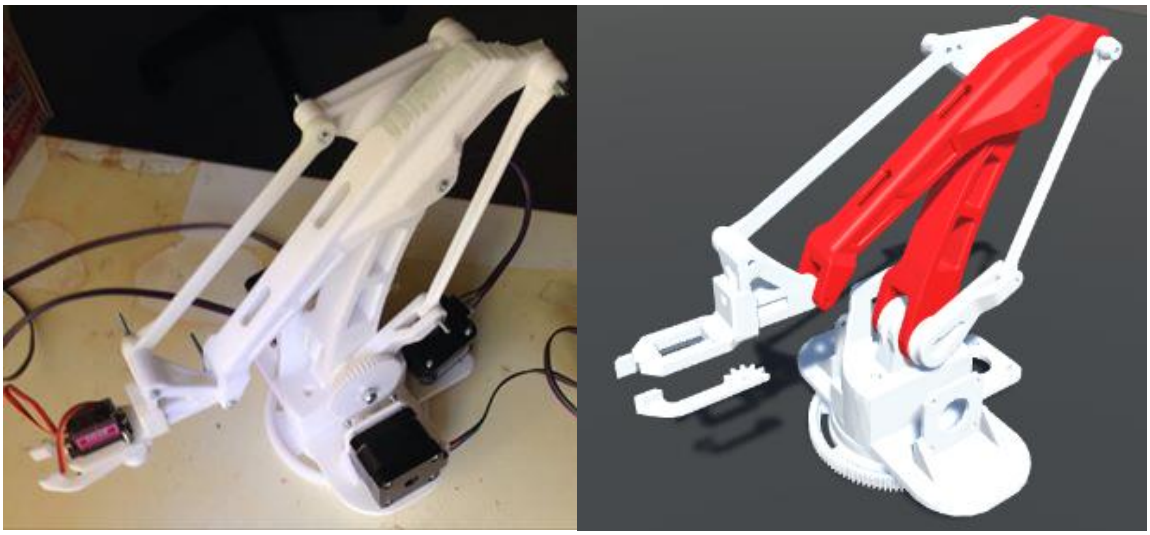


Figure 28: Final physical robot (left) and the simulated counterpart (right)

While the two systems appear visually similar there is one main difference. The simulated robot is unable to model the stepper and servo rotations, so they have been omitted. Due to this the gripper in the simulated robot appears to be floating when it should really be together with the servo. The gripper moves closer and further apart but does not rotate open and closed as the physical robot does.

To ensure a realistic representation, virtual robot has been configured to move with the same DOF as the physical robot.

5 Results

5.1 Scale

Firstly, to ensure our simulation will be useful, the accuracy of the scaling for this robot is shown. The specified lengths of our robot are known from measurements of the 3D models in Blender (see Appendix F) but they have been scaled down to make them more manageable for Unity. Of importance, here, are the lengths of the height from the ground to the first joint, the link length from the first joint to the second and finally the link length from the second joint to the gripper. In Unity these distances between each joint are checked to get our scale factors (actual length divided by simulated).

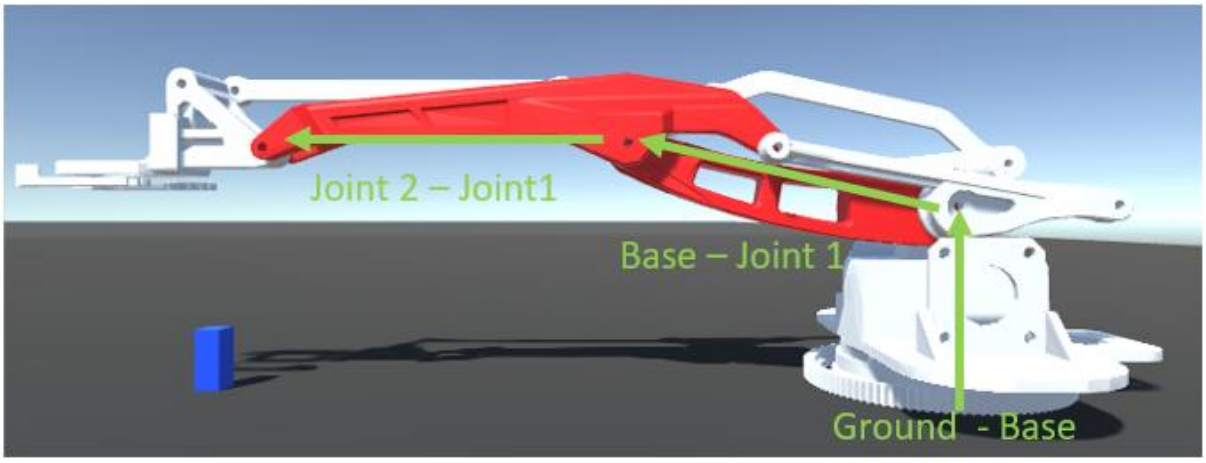


Figure 29: Simulated robot joint differences

Distance Label	Blender Measurement	Simulated Measurement	Scale Factor
Ground – Joint 1	7.8cm	2.97	2.63
Joint 1 – Joint 2	13.5cm	5.23	2.58
Joint 2 - Gripper	14.7cm	5.73	2.57

Table 3: Joint to Joint distances and scale factors in the simulated robot

This result is not completely unexpected, while the two links have a very close scale factor, the base is slightly higher. The base was constructed in Blender by joining three individual models together so it is likely small errors could have occurred in the process. In light of this, the average of the non-ground – joint 1 measurements, 2.575, shall be taken as the scale factor from the virtual robot to reality and it is assumed the simulated height from ground to joint 0 is actually $2.97 \times 2.575 = 7.65$cm.

5.2 Single Joint Movements

For this test, The final end effector positions are compared after one movement for a single joint, performed over every joint. Measurements of position will be taken from the middle of the joint which connects the last robot link to the gripper. For reference, the measurement will be relative to the middle of the base of the robot and presented in X-Y-Z coordinates. The test was performed in respect to each axis and for each joint (where joint 0 is the base twisting, joint 1 is the first robot hinge joint and joint 2 is the second) and the resulting error in virtual robot end-effector position, compared to the physical robot, is portrayed below.

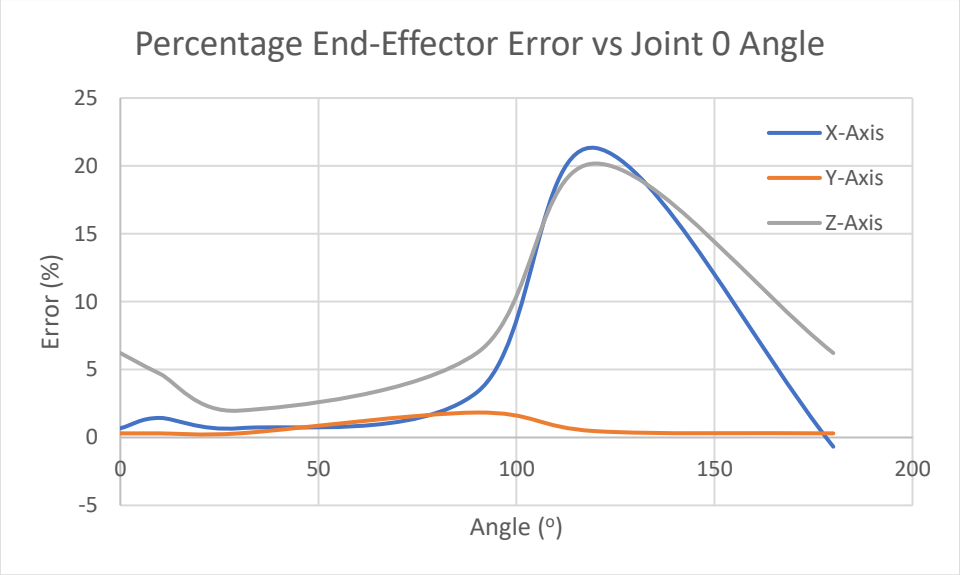


Figure 30: Percentage End-Effector Error (between virtual and physical robot) vs Joint 0 Angle

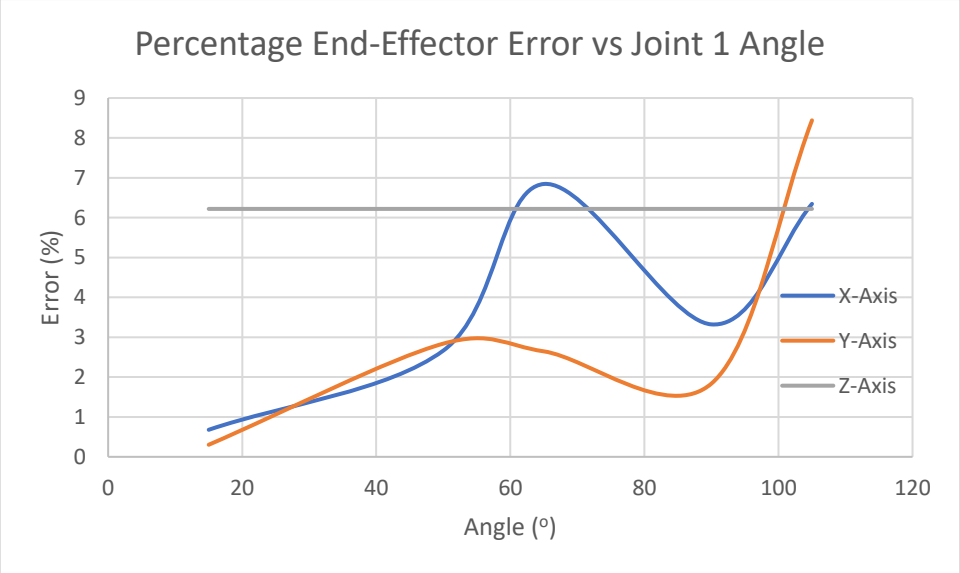


Figure 31: Percentage End-Effector Error (between virtual and physical robot) vs Joint 1 Angle

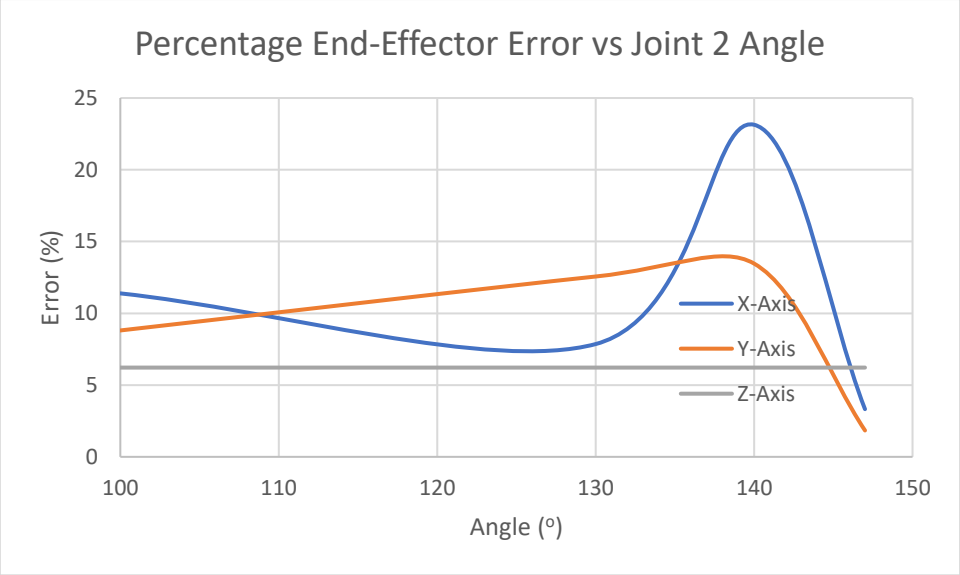


Figure 32: Percentage End-Effector Error (between virtual and physical robot) vs Joint 2 Angle

Note for the Y-axis with joint 0 and Z-axis for joints 1 and 2, the end effector did not move in that axis due to those joints rotation types, hence the near constant error values. Due to the mechanical constraints of the robot, some joints have more results and reachable angles than others.

For raw error value graphs and tabulated data, see Appendix I and K.

The maximum and average errors for each test are presented below.

	Joint 0			**Joint 1**			**Joint 2**		
Axis	Max Error (%)	Max Error (cm)	Average Error (cm)	Max Error (%)	Max Error (cm)	Average Error (cm)	Max Error (%)	Max Error (cm)	Average Error (cm)
X	21.3	0.38	0.11	6.84	0.49	0.27	13.5	1.80	1.10
Y	1.84	0.15	0.07	8.44	0.49	0.28	23.1	1.03	1.03
Z	20.2	0.2	0.07	6.22	0.05	0.05	6.22	0.05	0.05

Table 4: Maximum and average errors for the single movement test

5.3 Repeatability

To test for repeatability, the simulated robot was put through increasingly long chains of commands (up to 5), recording the end-effector position and orientation, then put through the same commands in reverse order and again recording the end-effectors, position and orientation (i.e. the test first involves doing one movement and then returning to the start, then two movements and returning to the start etc.). This test also includes some (manually forced) self-collisions to ensure no parts are dislodged and the robot configuration is altered, after these the robot was set back to the same angle and the test continued.

The 5 movements were:

- Rotate Joint 0 by +20
- Rotate joint 1 by +100
- Rotate joint 2 by -100
- Rotate joint 0 by -50
- Rotate joint 1 by -50

Number of movements completed vs error in end-effector position at start and destination

Number of movements	Average error on end-effector final destination (cm)	Average error on end - effector return to start position (cm)
1	0.00	0.00
2	0.00	0.00
3	0.00	0.00
4	0.00	0.00
5	0.00	0.00

Table 5: Table on of repeatability results

5.4 Inverse Kinematics Test

To run the IK test, two types of robots where considered, the physical robot model which has 3-degrees of freedom and 2 links and a 3-link 8-DOF robot with very simple architecture. Each robot was given 10 random positions to move towards and would generally start from the position the last run of IK ended in (if it was not too disadvantageous). The IK was considered successful if any part of the gripper was touching the target location, or if the target was out of reach, the robot must be pointing towards it with close to its maximum extent.

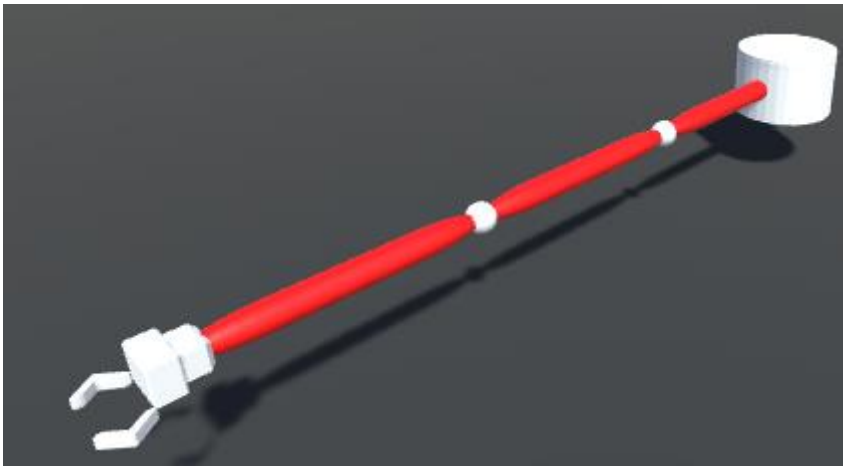


Figure 33: Simulated simple, 3-link, 8-DOF robot

Run	Success
1	Yes
2	Fell in line with destination
3	Fell in line with destination
4	Yes
5	Fell in line with destination
6	Collision loop
7	Collision loop
8	Yes
9	Yes
10	Yes

Table 6: Results of running IK on the modelled physical robot

Run	Success
1	Collision loop
2	Collision loop
3	Yes
4	Collision loop
5	Overtime
6	Local Minima
7	Collision loop
8	Collision loop
9	Yes
10	Collision loop

Table 7: Results of running IK on the 3-link, 8-DOF robot

5.5 Analysis of Results

From these tests, the accuracy of the system can be determined. Both the scale test and single movement tests show the ability of the robot to mimic a real robotic arm. While the scale test shows the ability of the simulation to accurately configure a robot, the movement test displays the system's ability to accurately manipulate the robot. The scale test gave good results, with the robot being able to be configured to a (nearly) consistent scale. The movement test exposed certain angles which were much more inaccurate than others, with joint 2 being highly inaccurate at most points. Otherwise, joints 0 and 1 had relatively small average errors. The repeatability test shows the system is completely repeatable, meaning the errors from the movement test are not accumulated or random. In this test, collisions encountered (once adjusted back to the pre-collision angle) had no impact on the final position of the end-effector.

It is important to note the real robot was likely not the most accurate itself, the stepper motor accuracy was 200 steps/revolution. This means for a 360^{o} there are only 200 steps, giving an error of $\pm 0.9^{o}$. Depending on the pose of the robot this can have noteworthy influence on the end-effector position in a certain axis. It may also be that this robot, who has many mechanical limits, is not suitable to be modelled in a simulation which cannot model them. Considering these factors, the system is still able to simulate robot movement well enough for our purpose, to allow new users to easily manipulate an articulate robot using real robot theory. It is not designed to compete with much larger and more accurate simulations.

One conclusion that can be drawn from the (albeit limited) IK test is that the more complex the robot, configuration-wise, the worse the Jacobian Transpose method will work. The 2-link physical had a much smaller chance of colliding with itself (causing the robot to be unable to progress), both due to its lower number of links and its lower DOF. The 3-link robot could easily tie itself into a knot while trying to reach a certain position. This resulted in two collision loop failures for the 2-link robot, compared to six for the 3-link robot. In general, the 2-link robot performed better, achieving its goal (or pointing towards unreachable ones) in five of the tests. The 3-Link robot performed more poorly, only achieving its goal twice in ten tests. See appendix J for more information on tests and result categories.

5.6 Limitations

As the system allows for such a wide degree of customisation, there is no guarantee that these tests will hold generally true for all robots. The system aims to expose and allow users to interact with the functionality of an articulate robot but does not compare to much larger systems which focus on accuracy. This is a form of simplification for the system but also a downside. Robots can be created and configured easily without many parameters outside of how they can move and how the parts are positioned, but they lack accurate functionality such as that of simulating motor control, sensors or rigorous IK methods.

The system is not suited for any formal robot development, the amount of error is too high for a robot whose main benefits include being more accurate than humans. As such, only informal small robots should be modelled with this system. If one wanted to quickly see the DH parameters, they needed to create a robot and visualise how they would work this system would be.

To allow for collision detection, the Unity physics engine is used but this also applies unwanted forces on the robot. The system is not designed to have the robot be hit or put into physically impossible poses, in such situations the physics engine has to deal with improper interactions causes the system to lag until it is unusable.

6 Conclusions

6.1 Topics for Further Investigation

For the most part, the system is closer to a skeleton than a fully-fledged simulation system. Both accuracy and performance can both stand to be improved, as well as the addition of more features common to robot simulators. Though the fact that IK could be programmed in the system bodes well for the extensibility of the system. In the long term, this may include having different types of robots added to the simulation, allowing for multi-robot systems and giving the user the ability to model environments as well as robots.

In the near term, there are still many features that need to be improved or implemented. While the collision detection works for detecting and repelling most collisions, if two objects happen to become stuck inside one another the program will often begin to lag, due to all the added physics, and becomes unusable. Currently, robots are moved and forced to follow their DH parameters but to make the robots more realistic, they should move using forces and physics and allow for their DH parameters to change due to events like a collision or falling down. This would allow for a much more realistic environment and allow for many situations to be left to the physics engine to deal with. Currently, if a force is applied to the robot, a robot joint is displaced and then will try snap back into place. Ideally, it this should be dealt with physics and if a collision occurs the robot should rotate or move appropriately with the collision.

6.2 Conclusions

The proposed system has created a virtual environment in which users may easily load, manipulate and test articulate robots. Compared to current robotic simulation packages, the proposed system suffers from accuracy and performance drawbacks. But in terms of creating an intuitive simulation environment which can easily be picked up for users new to robotics, engineering or computer science, the system boasts multiple features which help users quickly and easily begin robot simulation.

The systems strength lies in the fact that it can accurately represent the DH parameters in an interactive environment and the customisation of robots. Any user wishing to explore articulate robots will be able to manipulate an accurately configured robot or customise and load in their robot with any DOF and amount of links. As Unity is a free system, anyone will also be able to continue to contribute to this project.

In future, the hope is for robots and their simulation systems to become more widespread and a fundamental skill taught in schools, such as programming is becoming. Available simulation systems are becoming more accessible by the day to the average person and robotics libraries are growing in functionality while improving useability. With robots continuing to become a major part in our lives, it is only appropriate we strive to increase the number of people with understanding and interest in robots.

7 References

[1] "Bras robot industriel 6 axes de soudure, ARC Mate 120iC/10L de FANUC," 2009. [Online]. Available: https://en.wikipedia.org/wiki/Robot#/media/File:FANUC_6-axis_welding_robots.jpg. [Accessed: 09-Sep-2019].

[2] K. Junge, J. Hughes, T. G. Thuruthel, and F. Iida, "Improving Robotic Cooking Using Batch Bayesian Optimization," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 760–765, 2020.

[3] I. F. of Robotics, "Robots and robotic devices - Vocabulary," *ISO 8373:2012*, 2012. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en. [Accessed: 10-Sep-2019].

[4] R. Bernhardt, G. Schreck, and C. Willnow, "The Realistic Robot Simulation (RRS) Interface," *IFAC Proc. Vol.*, vol. 27, no. 4, pp. 321–324, 1994.

[5] P. Menzel and F. D'Aluisio, *Robo sapiens : evolution of a new species*. Cambridge, Mass.: MIT Press, 2000.

[6] L. Žlajpah, "Simulation in robotics," *Math. Comput. Simul.*, vol. 79, no. 4, pp. 879–897, 2008.

[7] A. Whitbrook, Ed., "Stage Simulations BT - Programming Mobile Robots with Aria and Player: A Guide to C++ Object-Oriented Control," London: Springer London, 2010, pp. 93–108.

[8] F. Rodriguez Lera, F. Casado Garcia, G. Esteban, and V. Matellan, "Mobile Robot Performance in Robotics Challenges: Analyzing a Simulated Indoor Scenario and Its Translation to Real-World." IEEE, pp. 149–154, 2014.

[9] Australian Curriculum Assessment and Reporting Authority (ACARA), "Digital Technologies." [Online]. Available: https://www.australiancurriculum.edu.au/f-10-curriculum/technologies/digital-technologies/?year=12983&year=12984&year=12985&year=12986&year=12987&strand=Digital+Technologies+Knowledge+and+Understanding&strand=Digital+Technologies+Processes+and+Production+Skills&capability=ignore&capability=Literacy&capability=Numeracy&capability=Information+and+Communication+Technology+%28ICT%29+Capability&capability=Critical+and+Creative+Thinking&capability=Personal+and+Social+Capability&capability=Ethical+Understanding&capability=Intercultural+Understanding&priority=ignore&priority=Aboriginal+and+Torres+Strait+Islander+Histories+and+Cultures&priority=Asia+and+Australia's+Engagement+with+Asia&priority=Sustainability&elaborations=true&elaborations=false&scotterms=false&isFirstPageLoad=false. [Accessed: 10-May-2020].

[10] S. Blackley and J. Howell, "The Next Chapter in the STEM Education Narrative: Using Robotics to Support Programming and Coding.," *Aust. J. Teach. Educ.*, vol. 44, no. 4, p. 4, 2019.

[11] J. Craig, *Introduction to Robotics: Mechanics and Control*. Pearson/Prentice Hall, 2005.

[12] User:Pushpendra050, "Classic DH Parameters Convention," *Wikimedia Commons*, 2020. [Online]. Available: https://commons.wikimedia.org/wiki/File:Classic_DH_Parameters_Convention.png. [Accessed: 06-May-2020].

[13] C. R. Rocha, C. P. Tonetto, and A. Dias, "A comparison between the Denavit–Hartenberg and the screw-based methods used in kinematic modeling of robot manipulators," *Robot. Comput. Integr. Manuf.*, vol. 27, no. 4, pp. 723–728, 2011.

[14] J. Denavit, "A kinematic notation for low pair mechanisms based on matrices," *ASME J. Appl. Mech.*, vol. 22, pp. 215–221, 1955.

[15] B. Siciliano and O. Khatib, Eds., *Springer handbook of robotics* . Berlin ; Springer, 2008.

[16] D. Tolani, A. Goswami, and N. I. Badler, “Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs,” *Graph. Models*, vol. 62, no. 5, pp. 353–388, 2000.

[17] User:GregorDS, “6DOF,” *Wikimedia Commons*, 2015. [Online]. Available: https://commons.wikimedia.org/wiki/File:6DOF.svg. [Accessed: 06-Jun-2020].

[18] W. A. Wolovich and H. Elliott, “A computational technique for inverse kinematics,” *CDC*. IEEE, pp. 1359–1363, 1984.

[19] C.-H. Wu and R. P. Paul, “Resolved Motion Force Control of Robot Manipulator,” *IEEE Trans. Syst. Man. Cybern.*, vol. 12, no. 3, pp. 266–275, 1982.

[20] A. Balestrino, G. De Maria, and L. Sciavicco, “Robust control of robotic manipulators,” *IFAC Proc. Vol.*, vol. 17, no. 2, pp. 2435–2440, 1984.

[21] L. Bermudez, “Overview of Jacobian IK,” *medium*, 2017. [Online]. Available: https://medium.com/unity3danimation/overview-of-jacobian-ik-a33939639ab2. [Accessed: 10-Sep-2019].

[22] O. Michel, “WebotsTM: Professional Mobile Robot Simulation,” *Int. J. Adv. Robot. Syst.*, vol. 1, Mar. 2004.

[23] Cyberbotics Ltd, “Webots.” [Online]. Available: https://cyberbotics.com/. [Accessed: 15-Mar-2020].

[24] RoboDK, “Simulate Robot Applications,” 2019. [Online]. Available: https://robodk.com/index. [Accessed: 10-Sep-2019].

[25] RoboDK, “Robot Machining,” 2019. [Online]. Available: https://robodk.com/doc/en/Robot-Machining.html#RobotCNC. [Accessed: 10-Sep-2019].

[26] J. Hider, “Software Helps NASA Automate Robot Programming Process,” *Mod. Mach. Shop*, vol. 91, no. 2, pp. 94–100, 2018.

[27] “SOFTWARE GIFTS ADVANCE STUDENT LEARNING AT PENN STATE BEHREND,” *US Fed News Service, Including US State News*. HT Digital Streams Limited, Washington, D.C., 2019.

[28] T. Braunl, *Robotics Education using Embedded Systems and Simulations.* 2007.

[29] J. Schützner, “Transfer of RoboSim from C to Java,” 1996.

[30] “RoboSim- A Simple 6-DOF Robot Manipulator Simulation System.” [Online]. Available: https://robotics.ee.uwa.edu.au/robosim/. [Accessed: 20-Aug-2020].

[31] F. Standhartinger, “Welcome to Productivity Boost.” [Online]. Available: http://www.productivity-boost.com/index.html. [Accessed: 06-Jun-2020].

[32] F. Standhartinger, “Non Convex Mesh Collider.” [Online]. Available: https://assetstore.unity.com/packages/tools/physics/non-convex-mesh-collider-84867#description. [Accessed: 06-Jun-2020].

[33] “Gazebo.” [Online]. Available: http://gazebosim.org/. [Accessed: 05-Jun-2019].

[34] “Simulation: Using the Gazebo interface.” [Online]. Available: http://library.isr.ist.utl.pt/docs/roswiki/sr_hand(2f)Tutorials(2f)Simulation(20)in(20)Gazebo.html.

[35] Cogmation Robotics, “SIMULATE LEGO® MINDSTORMS® ROBOTS.” [Online]. Available: https://www.virtualroboticstoolkit.com/. [Accessed: 03-Apr-2020].

[36] iRobot Corp, “Using Robots to Empower the Next Generation of Innovators.” [Online]. Available: https://edu.irobot.com/. [Accessed: 03-Apr-2020].

[37] E. Ackerman, “iRobot Launches Robot Simulator, Free Online Curriculum for Robotics Education,” 2020. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/home-robots/irobot-launches-robot-simulator-free-online-curriculum-for-robotics-education. [Accessed: 03-Apr-2020].

[38] T. Bräunl, “No Title,” 2020. [Online]. Available: https://robotics.ee.uwa.edu.au/courses/robotics/notes/. [Accessed: 28-Apr-2020].

8 Appendices

8.1 Appendix A

Comparison of currently available robot simulation systems

System	Main Use	Complexity	Cost (AUD)	Articulate Robots	Physics	Programmable robots	Other robot types	Sensor Simulation
Proposed System	Education/ Informal	Very Low - Low	Free	Yes	Yes	No	No	No
Gazebo	Formal/ Commercial	Medium -High	Free	Yes	Yes	Yes	Yes	Yes
Webots	Formal/ Commercial	Medium -High	Free	Yes	Yes	Yes	Yes	Yes
RoboDK	Formal/ Commercial	Medium -High	243.29	Yes	Yes	Yes	Yes	Yes
Virtual Robot Toolkit	Education/ Informal	Medium - Low	77.25	Yes	Yes	Yes	Yes	Yes
Robosim	Education/ Informal	Very Low	Free	Yes	No	No	No	No
iRobot Coding Platform	Education	Very Low	Free	No	Yes	Yes	Yes	Yes

[33][34][8][35][36][37][24][23]

8.2 Appendix B

RoboSim Commands

Writing own control programs: You can write control programs such like these ones being produced by the "Teach button", but you have got more command power. Every command starts with a command character and is completed by the arguments needed for the command. Every input, e.g. command character and arguments, must be in a seperate line. Otherwise the required motion of RoboSim cannot be guaranteed. The possible commands are as follows:

- *m* This character stands for manual control. It has to be followed by six double numbers representing the new angles. Remember that every command and every number has to be in a row of its own.
- *a* indicates auto control. In the next three lines there must be the three coordinates of the target point.
- *l* By choosing this character, which represents linklengths, you can determine new lengths for the links. These lengths has to be in the allowed ranges.
- *w* is the abbreviated form for angle weights. The arguments of this command are the six new angle weights which are integers.
- *s* represents speed. It has only one argument - the new speed. The range of possible values starts at 1(slow) and goes up to 82 (fast).
- *h* signifies that the content of the following line should be in the range of the aperture of the robot's hand. 0.0 means a fully open hand and 0.5 a totally closed one.
- *z* indicates the zooming factor. The zooming factor range is between 0.1 (very tiny) and 3.5 (large).
- # Comments start with # and last till the end of line.

The system is not case sensitive.

[29]

8.3 Appendix C

Structured text file for configuring robots

Where [] contains and actual value in the text file and 1# denotes what number that set of variables is for ordering.

In order, fill in each of these lines in a text file in order to create a robot configuration

1# [Number of arms]

2# [Scale of each arm model in the y-axis] [Scale of each arm model in the z-axis] [scale of the joint model] (space separated)

2# One line per arm

2# ...

3# Initial DH Parameters [length] [offset] [theta] [alpha]

3# One line for each joint

3#...

4# rotation of each arm model in degrees [rotation x] [rotation y] [rotation z]

4# One line per arm

4# ...

5# Is there a hand [1 or 0]

6# [Path to arm model]

6# One line per arm or [-] for the same as the last

6# --- NOTE arm must have -z as up and y as forward in blender, also align where the arm may interconnect on the x-z plane intersect

6# ...

7# [Path to base model]

8# [Path to joint model]

8# For each joint or leave as [0] to use first joint model

8# ...

9# DH parameters allowed to be changed for each joint, can either be 0 or 1 [theta] [alpha] [length]

9# One line per joint (note the length parameter is not functional currently)

9# ...

10# Restrictions on movement (not used yet)

11# [Path to final joint model]

12# [Path to hand model 1]

12# [Path to hand model 2]

13# Amount of arm overlap in length [scaling factor], allows for models to be longer than DH length

13# One line per arm

13# ...

14# Offset of each arm in x axis [offset]

14# One line for each arm -- NOTE offset + length is real length of part (takes into account real not DH length of preceding joint so add that offset to this)

14# Ensure things are to scale in the blender

14# ...

15# Offset of base position from joint 0 [x] [y] [z]

16# Base scale [scale]

17# Option to show and use joints for, each, joint [0 or 1] [... one number for each joint]

18# Rotate gripper hand or keep at horizontal [0 or 1]

19# Number of non-kinematic models (called bits) [number]

20# Bits have no physics and are for visuals, they are attached to a certain joint

20# [Path to bit] [joint number to follow position of] [joint number to follow rotation of] [x position offset] [y position offset] [z position offset] [whether to use position relative to world or to the joint this bit is attached to]

8.4 Appendix D

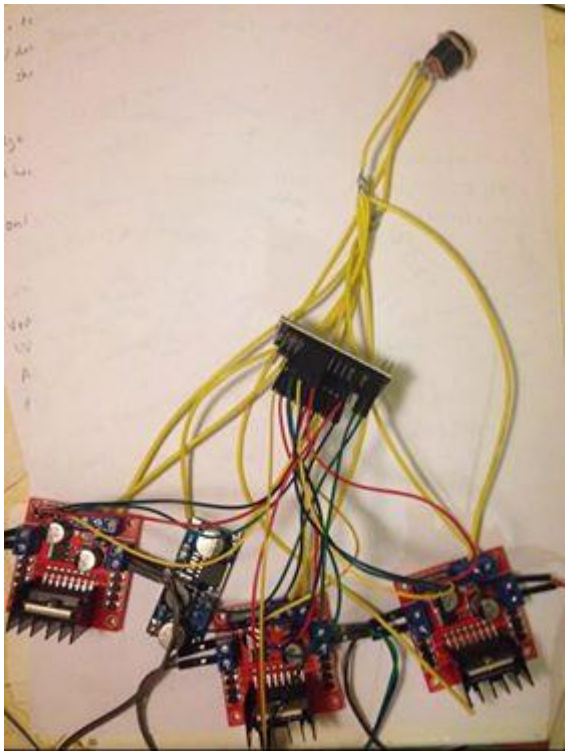
Physical robot arm construction images

All parts from 3D printer:

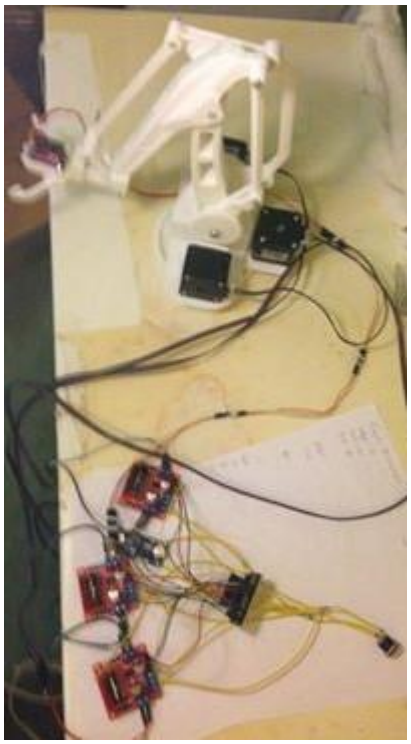
Construction with no electronics and no base:

Construction including base and motors, with no electronics attached:

Final Electronics:

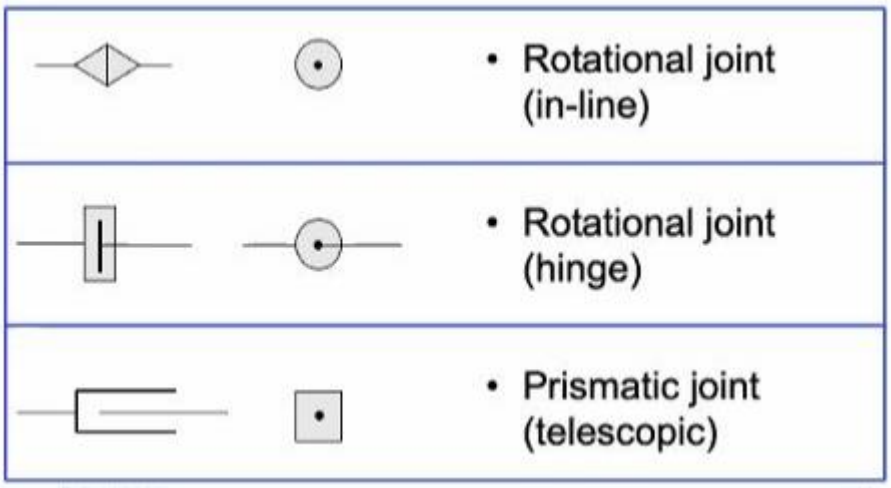


Final robot with electronics:



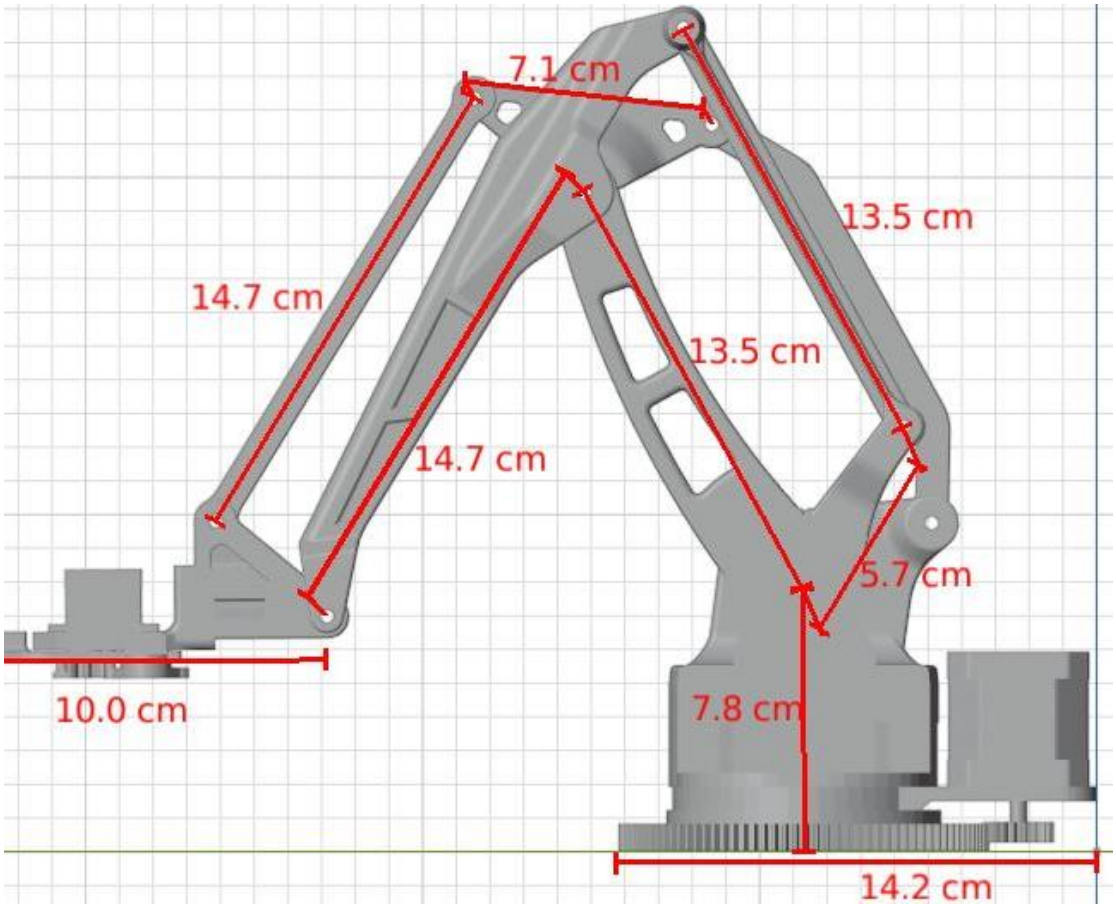
8.5 Appendix E

Robot notation diagram [38]



8.6 Appendix F

Robot dimensions



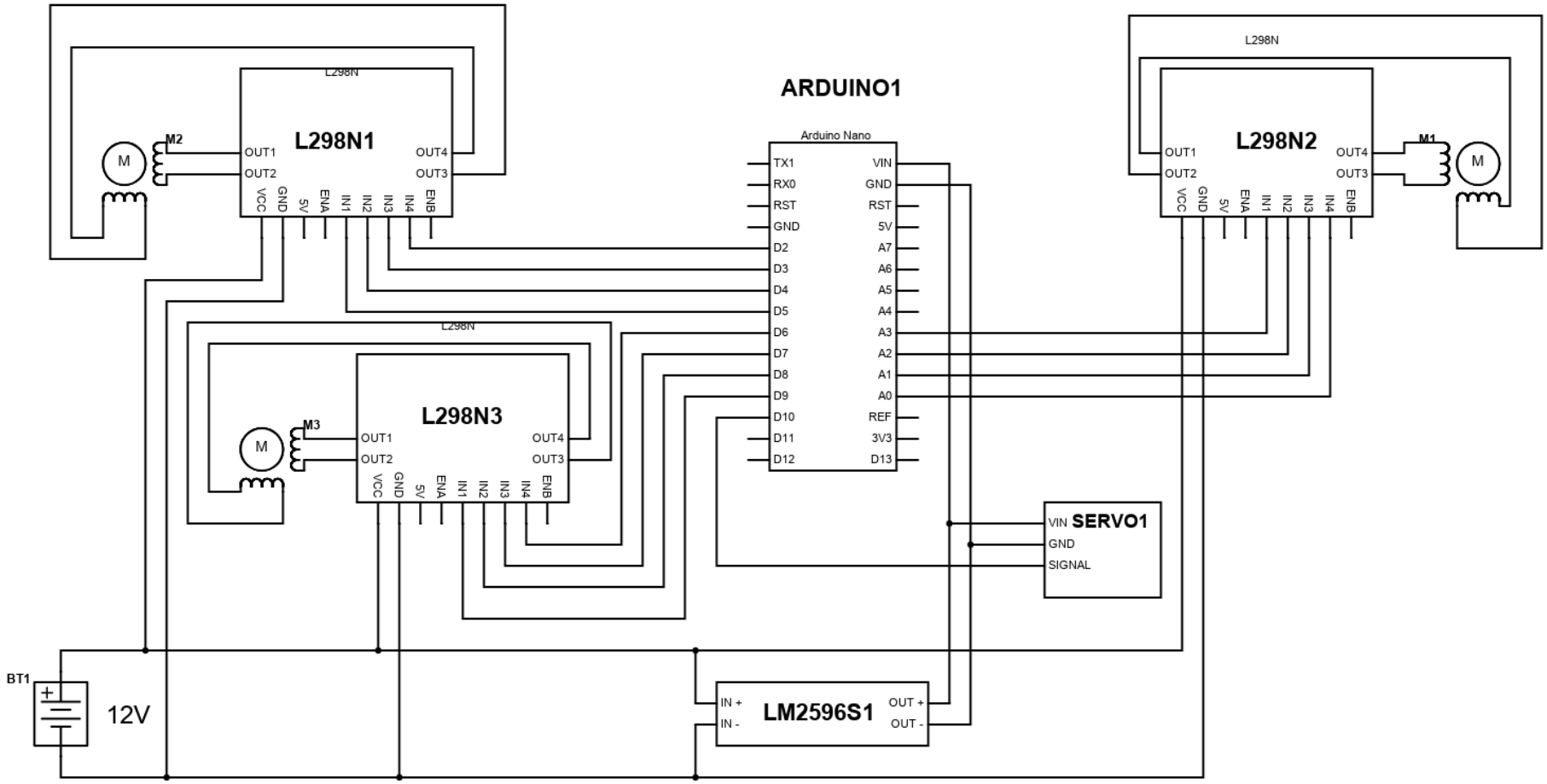
8.7 Appendix G

Parts List

Part Name	Quantity	Link
12V DC 3A Appliance Power Supply Adapter	1	https://www.altronics.com.au/p/m8937d-powertran-12v-dc-3a-2.1mm-fixed-tip-positive-appliance-plugpack/
2.1mm Female Plastic Chassis Mount DC Power	1	https://www.altronics.com.au/p/p0628-2.1mm-female-plastic-chassis-mount-dc-power-socket/
DC-DC Buck Module 3-40V In / 1.5-35V Out	1	https://www.altronics.com.au/p/z6334-dc-dc-buck-module-3-40v-input/
L298N Dual Motor Module for Arduino	3	https://www.altronics.com.au/p/z6442-l298n-dual-h-bridge-motor-module-for-arduino/
NEMA17 200 Step Stepper Motor & Lead	3	https://www.altronics.com.au/p/j0070-200-step-nema17-stepper-motor-and-lead/
Plastic Servo For Arduino	1	https://www.altronics.com.au/p/z6392-9g-180deg-plastic-servo-for-arduino/
Arduino Nano 3.0	1	https://www.altronics.com.au/p/z6372-funduino-nano-3.0-compatible-development-board/
3D Printed Parts	-	
M4 Threaded Rod 30cm	-	
M3 Screws and bolts	23	

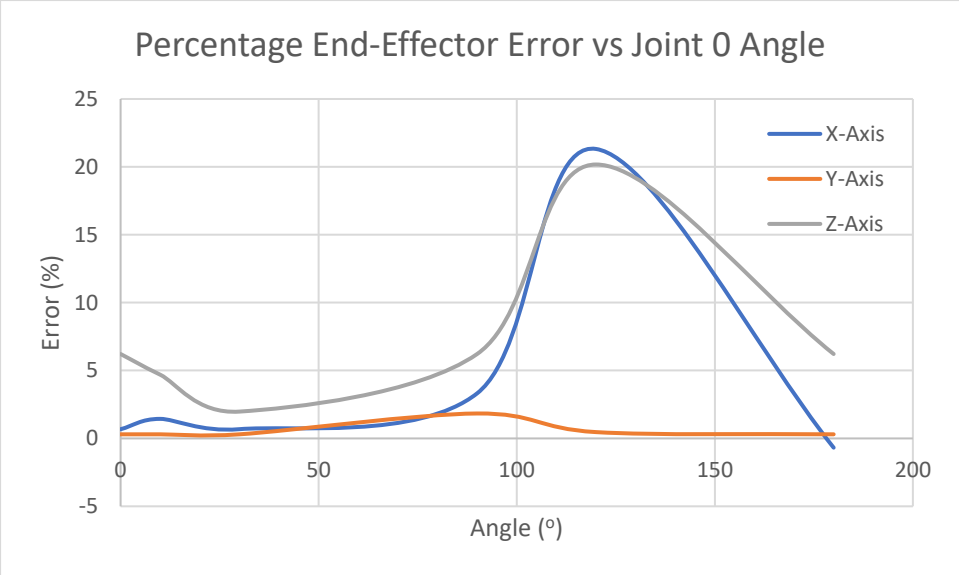
8.8 Appendix H

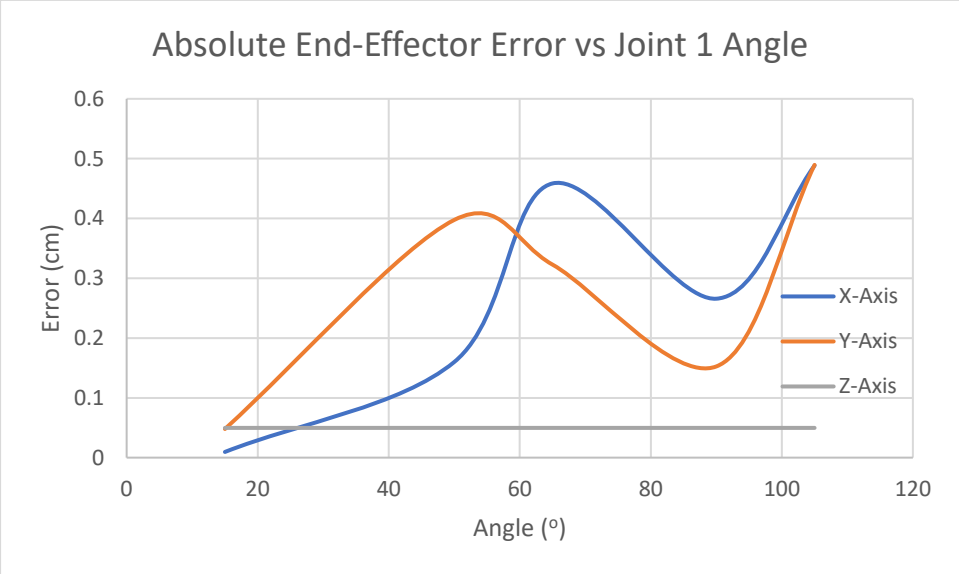
Wiring Diagram for the physical arm

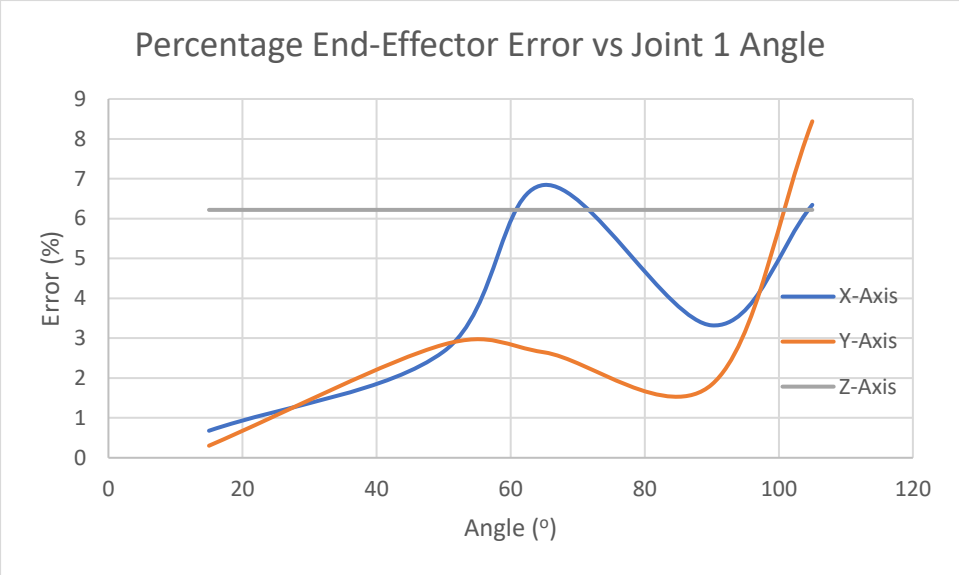


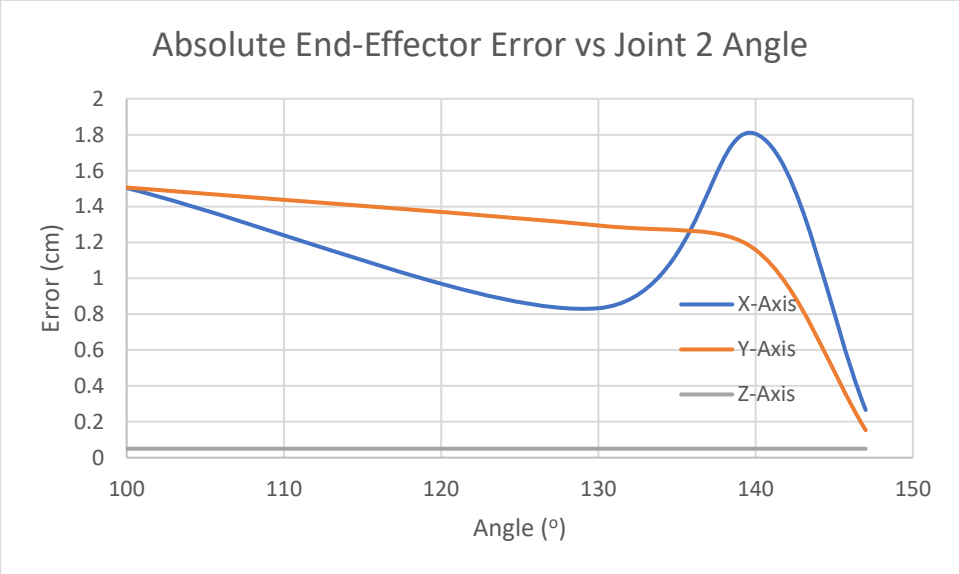
8.9 Appendix I

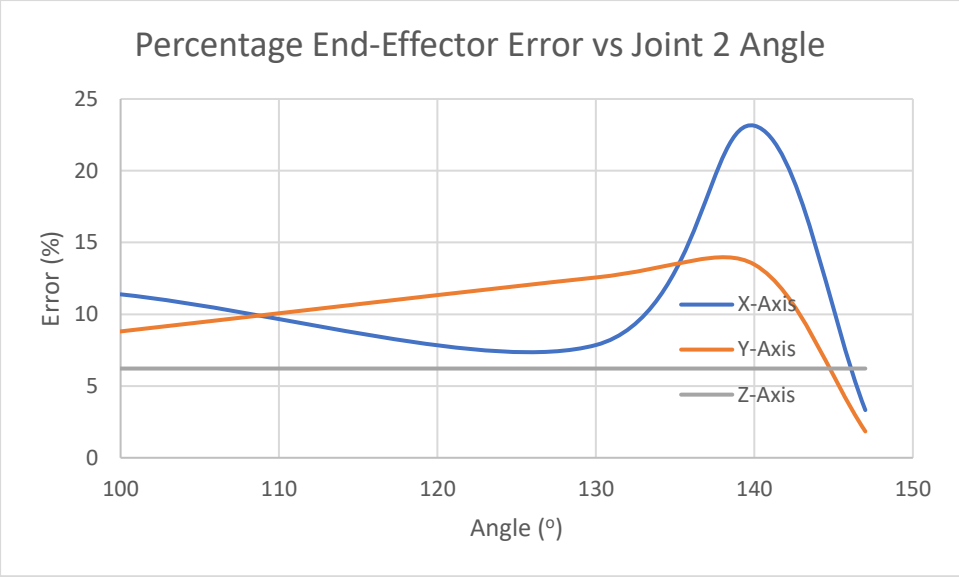
Error in end effector position per axis per robot joint for the physical and virtual robot


Percentage End-Effector Error vs Joint 0 Angle
Error (%)
25
20
15
10
5
0
-5
0
50
100
150
200
Angle (°)
X-Axis
Y-Axis
Z-Axis


Absolute End-Effector Error vs Joint 1 Angle
Error (cm)
0.6
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
120
Angle (°)
X-Axis
Y-Axis
Z-Axis


Percentage End-Effector Error vs Joint 1 Angle
Error (%)
9
8
7
6
5
4
3
2
1
0
0
20
40
60
80
100
120
Angle (°)
X-Axis
Y-Axis
Z-Axis


Absolute End-Effector Error vs Joint 2 Angle
Error (cm)
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
100
110
120
130
140
150
Angle (°)
X-Axis
Y-Axis
Z-Axis


Percentage End-Effector Error vs Joint 2 Angle
25
20
15
10
5
0
Error (%)
100
110
120
130
140
150
Angle (°)
X-Axis
Y-Axis
Z-Axis

8.10 Appendix J

IK Results for the model of the physical robotic arm

Destination (x,y,z)	Start position (x,y,z)	End position (x,y,z)	IK step count	Success
12.05, -2.5, -0.2	10.97, 0.911, -0.329	10.7, -2.16, -0.143	200	Yes
7.24, 2.38, -0.2	10.7, -2.16, -0.143	10.566, 3.21, -0.329	800	Fell in line with destination
8.71, -2.42, -0.2	10.566, 3.21, -0.329	10.368, -2.997, -0.149	900	Fell in line with destination
6.89, 1.47, -10.78	10.368, -2.997, -0.149	6.14, 1.18, -9.024	400	Yes
-1.82, 0.04, 2.03	6.14, 1.18, -9.024	5.40, 0.054, -9.268	1000	Fell in line with destination
-5.3, 0.04, 2.03	5.40, 0.054, -9.268	-0.319, -1.811, 0.0949	600	Collision loop
-10.3, 2.04, 2.03	-0.319, -1.811, 0.0949	-0.319, -1.811, 0.0949	600	Collision loop
-2.04, 5.43, -9.45	-0.319, -1.811, 0.0949	-1.703, 4.54, -7.035	800	Yes
-15.53, 5.43, -1.68	-1.703, 4.54, -7.035	-9.06, 3.812, -1.43	250	Yes
-1.39, -2.84, -1.68	-9.06, 3.812, -1.43	-1.86, -3.02, -0.862	1200	Yes

IK results for the 3-link 8-DOF robot

Destination (x,y,z)	Start position (x,y,z)	End position (x,y,z)	IK step count	Success
2.254, -2.5, -10.74	18, -8.731, -2.91	1.66, -1.75, -8.062	2000	Collision loop
-5.17, 7.35, 3.71	1.66, -1.75, -8.062	-4.85, 6.75, -1.671	800	Collision loop
3.72, -2.89, 4.52	-4.85, 6.75, -1.671	3.17, -385, 6.673	400	Yes
-0.62, 2.08, 0.65	3.17, -385, 6.673	3.17, -385, 6.673	400	Collision loop
-0.62, 6.13, -7.06	3.17, -385, 6.673	8.02, -2.21, -0.68	2000	Overtime
1, 3.74, 2.3	9, 5.82, 5.82	5.4280, 3.5744, 1,55	1000	Local Minima
-1.35, 2.22, 6.06	5.4280, 3.5744, 1,55	2.235, -1.17, 1.93	600	Collision loop
-2.92, 7.61, -6.33	2.235, -1.17, 1.93	-1.83, -0.895, 1.44	1200	Collision loop
1.02, 5.11, 2.509	18, -8.731, -2.91	0.11, 3.30, 2.89	1500	Yes
5.05, -3.23, 1.22	0.11, 3.30, 2.89	5.04, 2.89, 1.70	1000	Collision loop

Collision loop: The robot IK solution involved rotating through another joint, so the robot continually collides with another joint and never changes position, causing the robot to never update its solution.

Fell in line with destination: final link of the robot aligning with the goal. This caused the robot to believe it needed no more rotations to reach a goal and for its movements to cease as it cannot translate itself.

Local minima: Stuck in a certain pose it believes is close to the solution pose but cannot actually reach the goal from this pose. Normally another, better solution will exist that will not be explored

Overtime: the robot took too long to find an answer.

8.11 Appendix K

Results of single movement tests

Changing Joint 0 | Initial joint angles = 0, 15, 147 for joints 0,1,2						
Angles (º)	x error absolute (cm)	y error absolute (cm)	z error absolute (cm)	x error (%)	y error (%)	z error (%)
0	0.0095	0.0479	0.04975	0.678571429	0.303164557	6.21875
10	0.017874615	0.0479	0.04835335	1.44150121	0.303164557	4.6945
30	0.005534722	0.0479	0.027489348	0.683299074	0.303164557	1.977650899
90	0.26575	0.1525	0.04975	3.321875	1.837349398	6.21875
120	0.38375	0.07365	0.20175	21.31944444	0.466139241	20.175
180	0.0095	0.0479	0.04975	-0.678571429	0.303164557	6.21875
avg	0.115318223	0.069625	0.07114045	4.461019955	0.586024478	7.58390015

Changing Joint 1 | Initial joint angles = 0, 15, 147 for joints 0,1,2						
Angles (º)	x error absolute (cm)	y error absolute (cm)	z error absolute (cm)	x error (%)	y error (%)	z error (%)
15	0.0095	0.0479	0.04975	0.678571429	0.303164557	6.21875
50	0.1599	0.3975	0.04975	2.665	2.839285714	6.21875
65	0.4585	0.32275	0.04975	6.843283582	2.645491803	6.21875
90	0.26575	0.1525	0.04975	3.321875	1.837349398	6.21875
105	0.4885	0.4895	0.04975	6.344155844	8.439655172	6.21875

Changing Joint 1 | Initial joint angles = 0, 15, 147 for joints 0,1,2						
Angles (º)	x error absolute (cm)	y error absolute (cm)	z error absolute (cm)	x error (%)	y error (%)	z error (%)
147	0.26575	0.1525	0.04975	3.321875	1.837349398	6.21875
140	1.80475	1.158025	0.04975	23.13782051	13.46540698	6.21875
130	0.833	1.294	0.04975	7.858490566	12.5631068	6.21875
100	1.50325	1.505725	0.04975	11.38825758	8.805409357	6.21875