Imperial College London

Department of Mechanical Engineering

ME4 Individual Project

Designing a virtual robotics lab

Project Final Report

3 June 2021

Author: Aman Didwania

Supervisor: Dr. Nicholas Cinosi

Word count:

Abstract

Robotics is a field that has been gaining traction in the past decade, especially in the past few years. With demand at an all time high, more students are encouraged to study the field. One of the recent developments in the field is the use of virtual simulations – saving on hardware. This has become an inseparable part of the field and hence, is a skill students are encouraged to pick up. However, the software is used in industry has a steep learning curve that forms a barrier to simulation software. Furthermore, with the pandemic, labs are not in used – depriving students unable to use simulation software of practicals.

The purpose of this project is to create high quality simulations that can be easily accessed/manipulated by students who are not from a software background. This is achieved by creating models in ROS and simulating them in Gazebo. These models are then connected to Matlab, from where algorithms can be programmed to the models.

Models and lab tasks are designed to help a student through the learning curve. The simulations are compared to hand calculations for verification.

Table of contents

1. Introduction

1.1 Background

Robotics has a deep link with mechanical engineering, with several aspects of mechanical engineering (such as design, simulation, control, etc) being used in a variety of robotics applications (such as industrial, medical, services, etc) [1]. With a market revenue of $48 billion in 2017, it is one of the fastest growing industries [2]. One of the many reasons for this the availability of open source simulation softwares (such as robot operating system (ROS))[3]; with the ability to simulate robots in virtual reality, robots can be prototyped/tested without having to build it. In 2015, $150 million was invested in start-ups developing robots in ROS [4].

To address the increase in demand for robotics, the new module 'Introduction to Robotics' was introduced in 2020. The module covers forward and inverse kinematics of a simple mechanical manipulator, methods of control and basics of computer vision [5]. However, the course does not cover the simulation of robots on software. One of the reasons why it is not covered is because there is a large software barrier; most of the models (and other files, such as algorithms) created in ROS are mainly written in C++ or Python, both of which are not taught earlier in the course. Additionally, ROS is supported on Linux, which is not that commonly used [3]. The module did not have any practical activity this year due to the pandemic preventing lab use. In the given circumstances, students are not given the chance to apply knowledge learnt in the course.

All mechanical engineering students are taught how to use Matlab in their first year. Matlab has a toolbox (ROS toolbox) which enables the user to connect to a device running ROS in the local network. ROS runs using a publisher-subscriber scheme between topics and nodes; nodes can send (publish) and receive (subscribe) data from topics. For example, a light sensor (node) can send information about the distance of a robot from its target to the topic 'Distance to target'. The main processor (node) can read this value by subscribing to 'Distance to target' and, after executing its code, send a voltage/torque value to another topic 'Torque' that will be read by the motor. The ROS toolbox creates a global matlab node that can subscribe and publish to all topics, enabling the user to send and receive data from the device running ROS using Matlab code [6]. Using this feature, given a pre-built model in ROS, a student can write an algorithm in Matlab that receives data from the model as feedback and sends data accordingly. This way, students are able to have an

experience akin to labs and appreciate simulation software while bypassing the software barrier. Additionally, since this is not hardware dependent, it can be conducted remotely.

1.2 Objectives

The goal of this project is to create computer simulations of robots that can be used in place of labs. This is done in 3 steps (illustrated in Figure 1); models are created in ROS. They are then simulated in Gazebo. Finally, they are connected to and can be run from MATLAB.

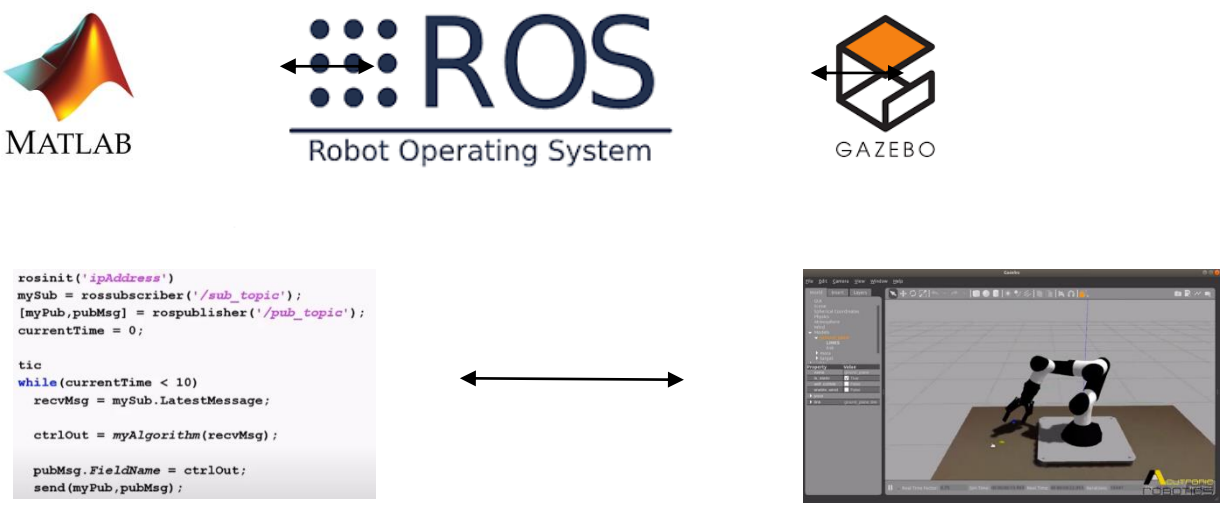

Fig. 1: Workflow; code written in Matlab is used to program model in ROS, which is simulated and visualised in Gazebo

To ensure the structure, quality and comprehensiveness of the models/labs, the following objectives have been laid out:

1. Models must cover a significant portion of the course material
2. Lab tasks should be basic initially, gradually increasing in complexity
3. It should be possible to perform all tasks without software prerequisites
4. Simulation should mimic real-world physics, accounting for gravity, wind, friction, collision-interaction, etc.
5. All labs should have visualisation of the robot moving as per code
6. Project should provide a foundation for building more advanced labs in the future

Objectives 1 and 2 are set so that labs follow a gradual learning curve while still present a challenge befitting of an undergraduate level. Objective 3 is set to make the labs student-friendly by removing the need to learn new software. Objective 4 and 5 are set to ensure the quality of the labs and help students gain a deeper appreciation of simulations. Objective 6 is set so that

2. Literature Review

2.1 Purpose and Briefing

When designing virtual labs, there are two aspects to consider – the platform on which the lab is created and the content of the lab itself. As mentioned in the objectives section, labs should be fairly challenging at an undergraduate level. While covering material from the module 'Introduction to Robotics' is a decent baseline, since the course is new and still being developed, a literature review was conducted to find out what labs were conducted other universities. Courses taught in other universities that related to robotics and taught basic concepts (such as kinematics) were studied to determine the amount of content suitable for an undergraduate level. Additionally, software usage would also provide a reference of the platforms available on which labs can be designed (Although ROS was chosen for this project, it is possible to sun simulations in Matlab or other simulators. This is discussed in more detail in the Choice of Software section).

To obtain a broader spectrum of results, universities in different countries were selected. The following universities were picked:

- Carnegie Mellon (USA)
- University of California Berkeley (USA)
- Massachusetts Institute of Technology (USA)
- Tokyo Institute of Technology (Japan)
- Indian Institute of Technology Bombay (India)
- National University of Singapore (Singapore)

2.2 Findings of literature review

Table 1 has information about the courses provided in other universities.

Table 1: Information about robotics courses in other universities [7]-[12]

University	Course Name	Topics Covered	Lab/Practical Activities
Carnegie Mellon University	Robot Kinematics and Dynamics	Kinematics, dynamics and path planning of industrial machinery and structures with linkages	Students are to write a path-planning algorithm to a mechanical arm to perform a task. While the mechanical manipulator is the same, tasks change on a yearly basis. For example, in 2016, the task was to make the robot play Jenga. In 2017, the task was to move food from a bowl to a person's mouth. Projects are written in ROS and communicate with existing hardware.

University of California Berkeley	Introduction to Robotics	Kinematics, dynamics, control, robot vision and path planning of a mechanical manipulator. Path planning includes systems with constraints and obstacle avoidance	Students have labs weekly that explain the structure of ROS. Labs (in chronological order) include moving a joint (forward kinematics), moving to a location (inverse kinematics), pick and place operation, viewing an image and performing analysis (such as thresholding, edge-detection, clustering) and building occupancy grid based on vision (for obstacle avoidance). Students are then made to combine concepts learnt in labs to a project of their choosing. Examples include using a robot to play chess and making a robot follow a target. Course uses hardware which communicates with ROS to demonstrate concepts.
Massachusetts Institute of Technology	Introduction to Robotics	Control, kinematics and dynamics of a robot	There are 3 lab activities: writing code to control (using control theory) a simple DC motor using PWM, reading sensor information and combine the previous 2 to perform a searching activity. In 2005, this was to search for mines and return.
Tokyo Institute of Technology	Robot Kinematics	Kinematics and design procedure of a mechanical manipulator. Path planning algorithm.	None; learning is conducted via lectures and tutorials covering theory.
Indian Institute of Technology Bombay	Robotics	Construction of robots as per application. Open-loop, closed-loop and servo control, kinematics, path planning and computer vision of manipulators. Developments in sensor technology	None; learning is conducted via lectures and tutorials. There are demonstrations of using code to obtain sensor information and perform tasks.
National University of Singapore	Robotic System Design	Types of sensors and acutators, construction of robots as per application. Kinematics, control and path planning of manipulators. Simulation of robots in ROS.	Labs occur weekly, covering moving a joint, moving locations, acquiring and processing sensor data. As a final project, students are to simulate a mobile robot in a simulation software.

2.3 Evaluation and applicability to project

While there are differences in the topics covered, labs and practical activity in all universities, the topics in common are robot kinematics, dynamics and path planning. While practical activities are varied, all courses (where there is a practical component) have the following similarities:

- 1-2 concepts are taught per lab (like moving a joint, moving to a location (University of California Berkeley) or controlling a motor, reading sensor data (Massachusetts Institute of Technology))
- An activity involving moving a robot from position A to position B that usually combines a vast majority of the concepts taught in previous labs (like making a robot move food from a bowl to the mouth, which combines vision, kinematics and path planning (Carnegie Mellon University))

From this, it can be said that as a baseline for course on Robotics at an undergraduate level, it should cover at least kinematics, dynamics, and path planning. It is noteworthy that courses which involve the use of ROS have prerequisite courses that cover knowledge of Python, Linux and other software skills.

3. Labs

This section will go over the creation of the project, design choices made and their justification.

3.1 Choice of Software

The first decision that needs to be made is the software platform on which the project will be run. As per the literature review, most universities use ROS for practical activities. It is possible to model a robot completely in Simulink. CAD models from Solidworks can be converted into xml files and imported into Simulink. The imported model is programmable and can be visualised by Simscape 3D animation. However, this would require writing down the code for all the equations of motion for the robot. Furthermore, unless there is hardware in the loop, simulation results will not reflect the effects of friction, material deflection, system constraints, and other real-world factors that normally would affect the robot unless they are all coded into the simulation [1] [13]. While it is possible to code all the environment factors, it is not feasible within the time-frame of this project.

As per objectives set, simulations must accurately represent how the robot moves. Hence, an open-source 3D robotics simulator called Gazebo was used. The advantage of using

Gazebo lies in the many physics libraries pre-coded in it, allowing for it to simulate complicated robots in complex environments realistically [14] [15]. Although Gazebo can be used without using ROS as an intermediary, there is a lot of community support and resources available for using ROS with Gazebo and lack of support for other forms of usage. As per the literature review, a lot of universities use ROS to conduct practical activities. CAD models from Solidworks can be imported into ROS. ROS can also be used with a range of hardware including but not limited to Arduino and MPLAB – two of the most commonly used hardware in education [16]-[18] – allowing for the integration of hardware in labs when the pandemic settles. Hence, ROS was chosen as the platform to build the robot in.

Due to the objective of the project having no software prerequisites, Matlab, which is taught in 1st and 2nd year, was connected to the project using the ROS toolbox, leading to the workflow as shown in Figure 1.

3.2 Base Model

When designing the base model, there were 3 main considerations to be made:

- Complexity of the model – the simpler the better
- Can the model be created within the project time frame
- Can the model be used to teach kinematics, dynamics and path planning

In order to demonstrate the difference between forward and inverse kinematics, the robot should have at least 2 independent joints.

Fig. 2: Scara robot with 3 revolute joints and a prismatic joint [19]

Figure 2 shows a industrial scara robot with 3 revolute joints and 1 prismatic joint. Note that the prismatic joint has complex geometry as the rotational motion of a motor has to be converted into translational motion via a transmission system. Revolute joints in

comparison, have simpler geometry as they can be modelled by 2 cuboids attached to each other at the end and driven directly by a motor.

Hence, the simplest model fulfilling all requirements is an RR manipulator (a robot with 2 revolute joints). Figure 3 shows the robot used [20]. Table 2 shows the specifications of the robot. (to save time, a prebuilt robot – with pre-calibrated specifications – was used)

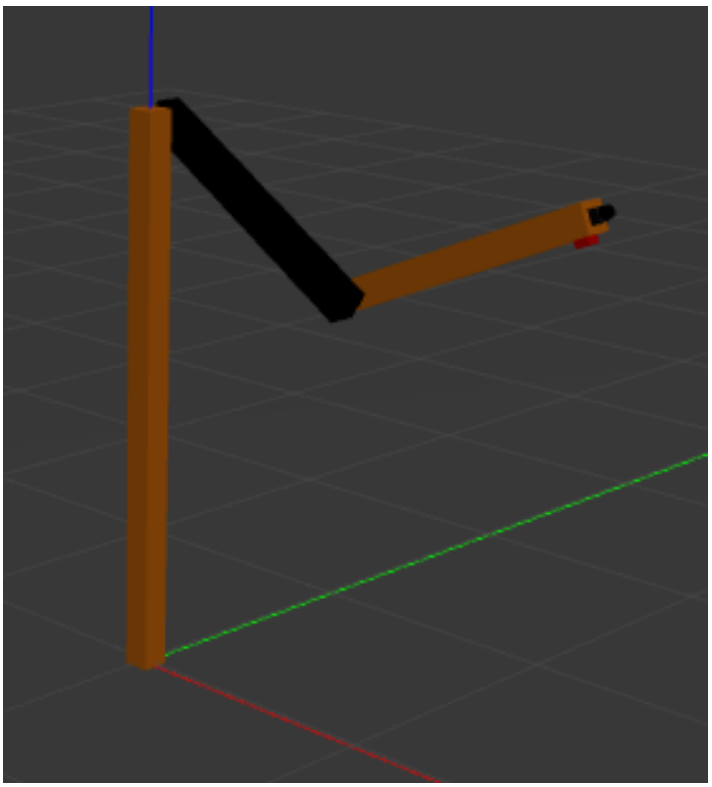

Fig. 3: RR manipulator used [20]

Table 2: Specifications of RR manipulator links

Property	Link 1 (on ground)	Link 2 (intermediary link)	Link 3
Mass (kg)	1	0.5	0.5
Height (m)	2	1	1
Width and Depth (m)	0.05	0.05	0.05
Coefficient of Friction	0.2	0.2	0.2
Damping	0.7	0.7	0.7

The model spawns with the 3 links aligned with the blue axis as shown in Figure 5.

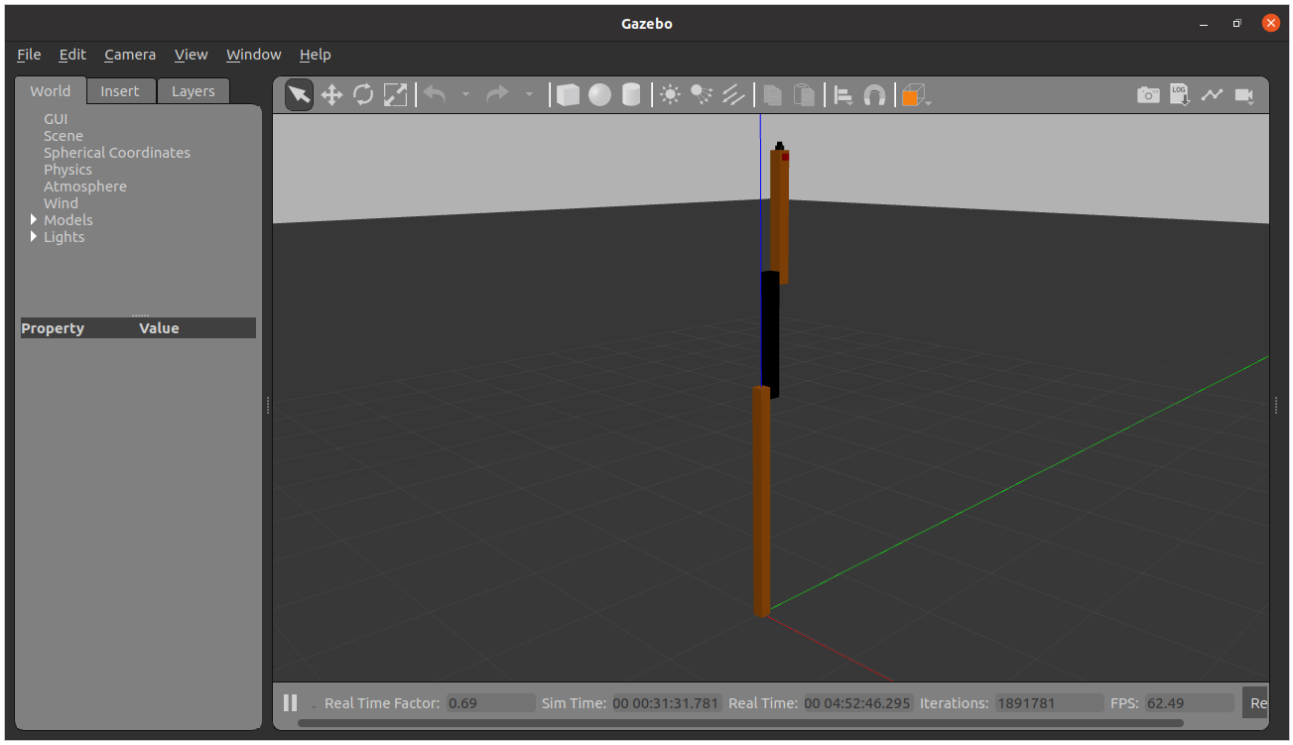

Fig. 5: Initialisation of Model

Then, the effort controllers (discussed further in next paragraph) are loaded in and initialised, setting the position θ_0 of both joints to 0. The joints can then be commanded to move to any position number. Unique positions are from 0 to 6.28 (2π). Any number that exceeds 6.28 (or precedes 0) will cause the joint to complete a number of complete rotations. If the commanded position is greater than the current position, the model will move clockwise and vice versa. For example, if the commanded position is 7, the robot will complete a rotation and end at the position 0.72.

The joints are modelled as if they were being driven directly by a DC motor. This is done by using an effort controller at the joints. An effort controller produces a torque output at the joints. It takes in the desired output (usually position, velocity or acceleration), asserts a torque output required to achieve desired results. For a DC motor, the equation of motion is given by equation 1, where J is the moment of inertia of the motor, b is the motor viscous friction constant, θ is the position and T is the torque.

$$J\ddot{\theta} + b\dot{\theta} = T \tag{1}$$

Consider the upper joint in Figure 5 (henceforth, referred to as joint 2), with the lower joint (henceforth, referred to as joint 1) assumed stationary. The effect of gravity will change the equation of motion from equation 1 to equation 2.

$$J\ddot{\theta} + b\dot{\theta} = T + mg\frac{l}{2}sin(\theta + \theta_1) \quad (2)$$

Where l is the link length and θ_1 is the position of joint 1. To further complicate things, θ_1 will be partially dependent on θ (as the position of link 3 changes the load on joint 1). Unfortunately $sin(\theta)$ cannot be converted into the Laplace domain [24]. Given the calculational complexity, it will be omitted from calculation – giving a poor approximation of the system. However, this will later be compared to the results of the simulation (in the Analysis of Simulation and Labs section).

Converting equation 1 into the Laplace Domain, we get:

$$s(Js + b)\theta(s) = T(s) \quad (3)$$

Giving a transfer function of:

$$\frac{\theta}{T} = \frac{1}{s(Js + b)} \quad (4)$$

Let us define the green axis on Fig. 3 as the z-axis. The moment of inertia at the joint about this axis is given by equation 5.

$$J_{zz} = \frac{1}{12}mL^2 + mr^2 \quad (5)$$

Where J_{zz} is the moment of inertia, m is the mass of the link, L is the link length, and r is the distance between the joint and the centre of mass. Using this formula, the inertia values for the joint 1 (between link 1 and link 2) and joint 2 (between link 2 and link 3) can be calculated and amount to $0.167\,Nm^2$. The damping constant is given by the specifications (0.7). The effort controller is uses proportional, integral and derivative (PID) control with negative feedback to give the desired position/velocity. Figure 4 shows the block diagram of the system. Equation 6 gives the closed-loop transfer function.

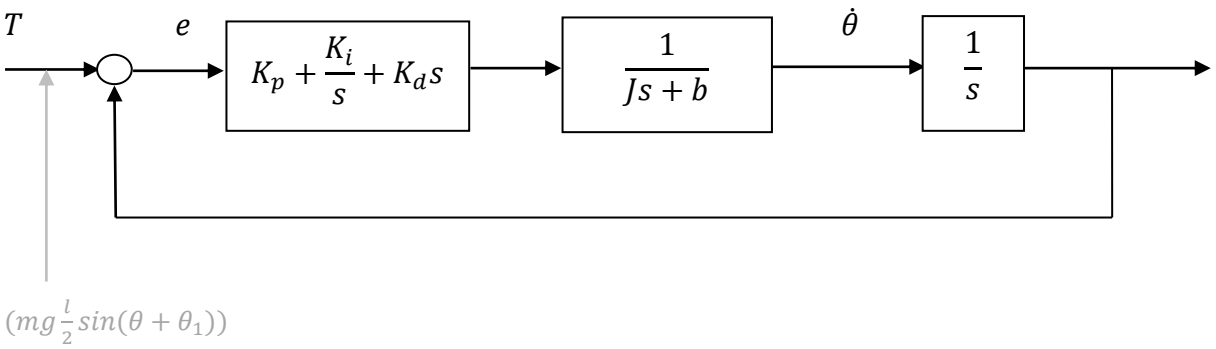

Fig. 4: Block diagram of a joint in the simulation

Note that the gravity term is in grey as it will affect the simulation.

$$\frac{\theta}{T} = \frac{s(Js+b)}{s(Js+b)+K_p+\frac{K_i}{s}+K_d s} = \frac{0.166s^3+0.7s^2}{0.166s^3+(0.7+K_d)s^2+K_p s+K_i} \quad (6)$$

Note that for velocity, transfer function is given by equation 7.

$$\frac{\theta}{T} = \frac{Js+b}{Js+b+K_p+\frac{K_i}{s}+K_d s} = \frac{0.166s^2+0.7s}{(0.166+K_d)s^2+(0.7+K_p)s+K_i} \quad (7)$$

K_p, K_i, and K_d are parameters of the effort controller and can be varied to give response as per requirement.

3.3 Node Graph

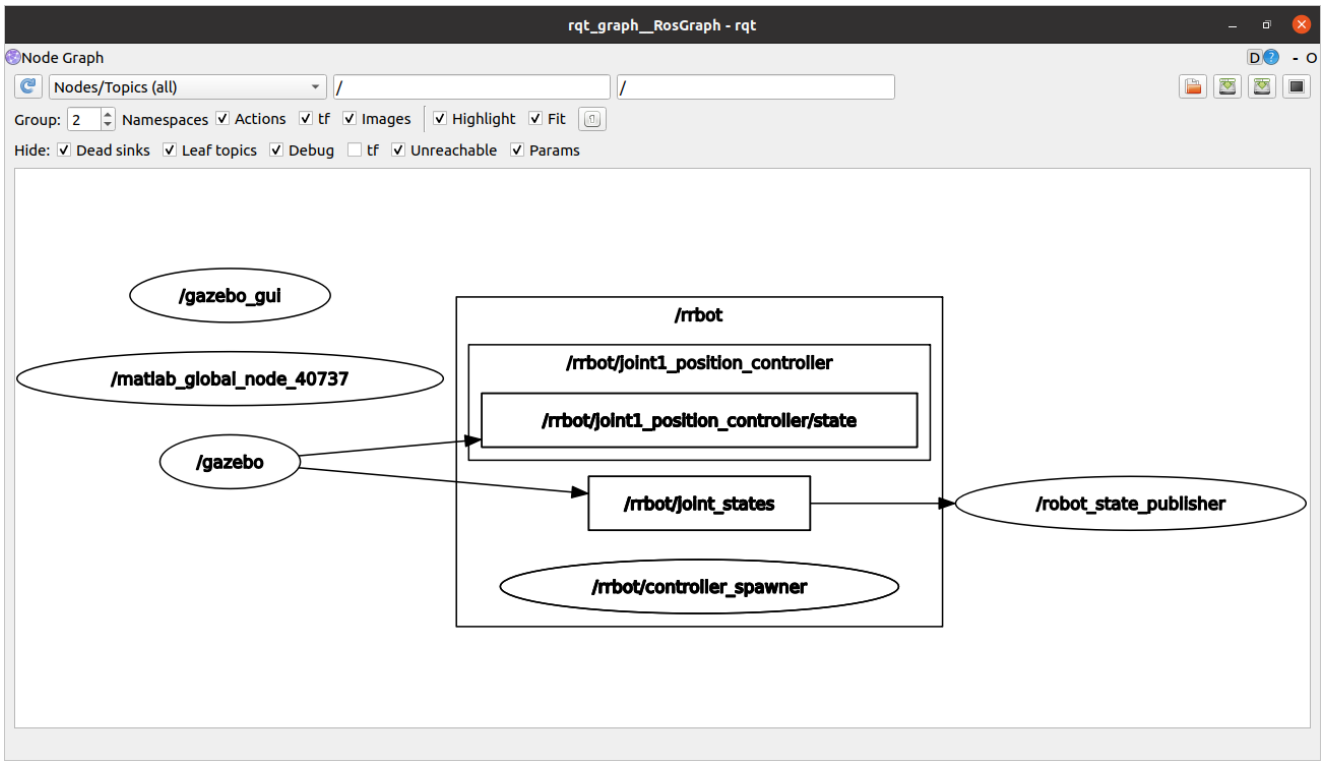

Fig. 6: Node Graph of the model

Figure 6 shows the node graph – a graphical representation of how the nodes interact with the topics – of the model [3]. The names in the circles are nodes and the names in the rectangles are topics. The arrows show how data is being transferred; for example, /gazebo is writing information about the joints' positions to /rrbot/joint_states and /robot_state_publisher is receiving information from /rrbot/joint_states.

3.3.1 Topics

/rrbot – This is the main topic – the model. It has subtopics within it for specific properties about the model

/rrbot/joint_states – This topic contains information about the kinematic and dynamic properties of joints 1 and 2 (eg. commanded position/velocity, actual position/velocity, torque at joint, etc)

/rrbot/joint1_position_controller – This topic contains information about the effort controllers loaded into the model – the PID parameters (PID constants and min/max integral windup), commanded position, actual position, and error.

/rrbot/joint1_position_controller/state – This topic is a subtopic of the /rrbot/joint1_position_controller topic.

3.3.2 Nodes

/robot_state_publisher – This node reads individual joint information (eg. position/velocity/torque at the joint) from the /rrbot/joint_states topic. It then uses the urdf (universal robot description file) of the model to transform individual joint information into information in the base frame (by applying forward kinematics) [22].

/gazebo – This is the node in which the simulation takes place. The model (along with controllers) is spawned and fixed to the ground at the origin. It is subject to environment variables such as gravity, wind, magnetic field, etc. and foreign objects spawned (discussed in more detail in lab 3 and 4 sections). It sends the values recorded at the joints to the /rrbot/joint_states and /rrbot/joint1_position_controller/state topic.

/gazebo_gui – This node is responsible for the graphical user interface (provides visual feedback for what happens in the virtual ‘world’ in the simulation).

/matlab_global_node_40737 – This is the way matlab interacts with the model. It is a global node and can subscribe to or publish data from any node. The most basic application of this would be to publish a command position/velocity to /rrbot/joint_states and read actual position/velocity.

/rrbot/controller_spawner – This node spawns the controllers into the model. Note that normally it is possible to model a robot with controller coded in its description[]. However, in order to change the type of controller or controller parameters, the robot description file would have to be rewritten. Hence, it is more convenient to have separate files for each controller, and a node that spawns (/deletes/replaces) them in the model.

3.4 Lab 1

To keep things simple at the start, the first lab would have 4 objectives:

1. To send data from Matlab to ROS
2. To receive data from ROS in Matlab
3. To perform a simple algorithm that increments the position of joints
4. To make the end-effector move from position A to position B

The lab would start by having a computer running the model in ROS/Gazebo in another machine (or virtual machine [3]). The first step is to connect Matlab to ROS. This can be done by the rosinit('IP_dddress') command (where 'IP_address' is replaced by the IP address of the computer running ROS). After running the command, it is possible to verify if the connection was successful by trying to see the topics/nodes. This can be done by running the command 'rostopic list' or 'rosnode list'. On doing so, the topics and nodes discussed in the Node Graph section (along with their subtopics, eg. /rrbot/joint_states/position) should appear [6].

To meet the first objective, a publisher must be set up. Figure 7 shows a piece of code that accomplishes this task.

```
IP = '10.149.7.74';% IP address of computer running ROS
rosinit(IP)
joint1 = rospublisher('/rrbot/joint1_position_controller/command');
msg1 = rosmessage(joint1);
msg1.Data = 0.0;
send(joint1,msg1);
```

Fig. 7: Code for sending data to ROS from Matlab

The rospublisher command creates a publisher that can write a message to the topic /rrbot/joint1_position_controller/command. The next thing to do is to define a message. This is done by the rosmessage command. A message has 2 parameters: Data and Message type. Matlab automatically updates the message type based on the publisher/subscriber of the message. Finally, the send command sends the data to ROS (publishes msg1 to joint1). The successful transmission of information can be verified by the gazebo GUI, which should show the joint moving to position coded (0 in this example).

Figure 8 shows a piece of follow up code that receives information from ROS.

```
joint1_position = rossubscriber('/rrbot/joint_state/position[0]');
current_position1 = joint1_position.LatestMessage;
display(current_position1.Data)
```

Fig. 8: Code for receiving data from ROS

Using the rossubscriber command, a subscriber is setup to the topic /rrbot/joint_states/position[0]. The topic /rrbot/joint_states/position has 2 elements for the 2 joints. The indexation [0] at the end of the position is to select the position of joint 1. Note again that the message will have 2 parameters and hence, to verify, the Data parameter of the message can be displayed.

Moving on, the goal of objective 3 is to synchronize the communication frequency between Matlab and ROS. Let us consider a simple Matlab script that increments the position of a joint (as shown in Figure 9):

```
for i = 1:5
    msg1.Data = current_position1.Data+2;
    send(joint1,msg1)
    current_position1 = joint1_position.LatestMessage;
end
```

Fig. 9: Example code

This code – which should increment the position by 10 (2 five times) – will fail as the loop will ask for the position data before ROS has registered the data published by the send command, causing the loop to stop. To fix this, the rate at which data is being sent and received must be controlled. This can be done by the rateControl command – which controls the loop frequency in Matlab. This helps illustrate the reality of the simulation and helps students appreciate the problems associated with sampling, data transfer and processing – something that would occur normally when tested with hardware.

Finally, to satisfy objective 4, the following task (as shown in Figure 10) has been laid out:

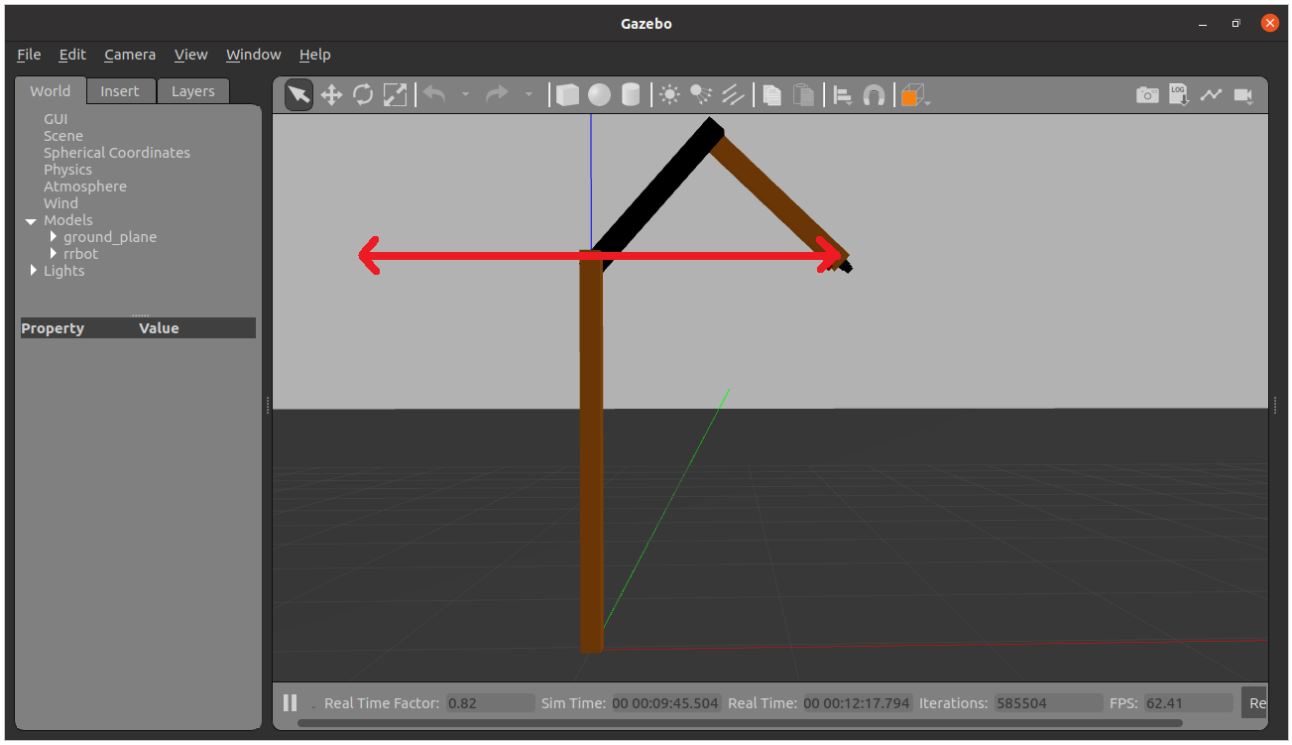

Fig. 10:

Assuming the red axis to be the y-axis, blue axis to be the x-axis and green axis to be the z axis, students must move the end effector between position (2, -1.4, 0.05) and position (2, 1.4, 0.05) (Note that 0.05 is the link offset). To find the joint positions that will result in these end-effector positions, students will have to calculate the forward kinematics of the manipulator and then derive the inverse kinematics [5] (this is done in the Analysis of Simulation and Labs section).

3.5 Lab 2

While position control provides control over the initial and final position of the end-effector, it cannot control its trajectory, which is controlled by velocity control. Building on Lab 1, Lab 2 is about velocity control. The effort controllers that output torque to give desired position output are replaced with controllers that output torque to give desired velocity. Since Lab 1 covered communication, Lab 2 can focus on performing a slightly more complicated task.

The task is as follows: (assuming same co-ordinate system as in Figure 10) students must write code in Matlab that causes the end-effector to move from along the red, from position (2, -1.4, 0.05) to position (2, 1.4, 0.05) in a line (velocity in the x and z directions must be zero). To do this, they will have to calculate the Jacobian of the manipulator from the inverse kinematics (calculated in Lab 1), set the velocities of y and z to zero, calculate the

inverse Jacobian and solve the matrix equation to get joint velocities [5] (hand calculation carried out in the Analysis of Simulation and Labs section).

3.6 Lab 3 / 4

Lab 3 (and 4) introduces foreign objects into the simulation. Pre-built models [23] were taken due to time constraints. A bookshelf and a coke can are spawned 1 m away (in the y direction) from the manipulator but aligned in the z axis (green axis) with the end effector as shown in Figure 11. The height of the bookshelf is 1.2 m and the height of the coke can is approximately 0.1 m.

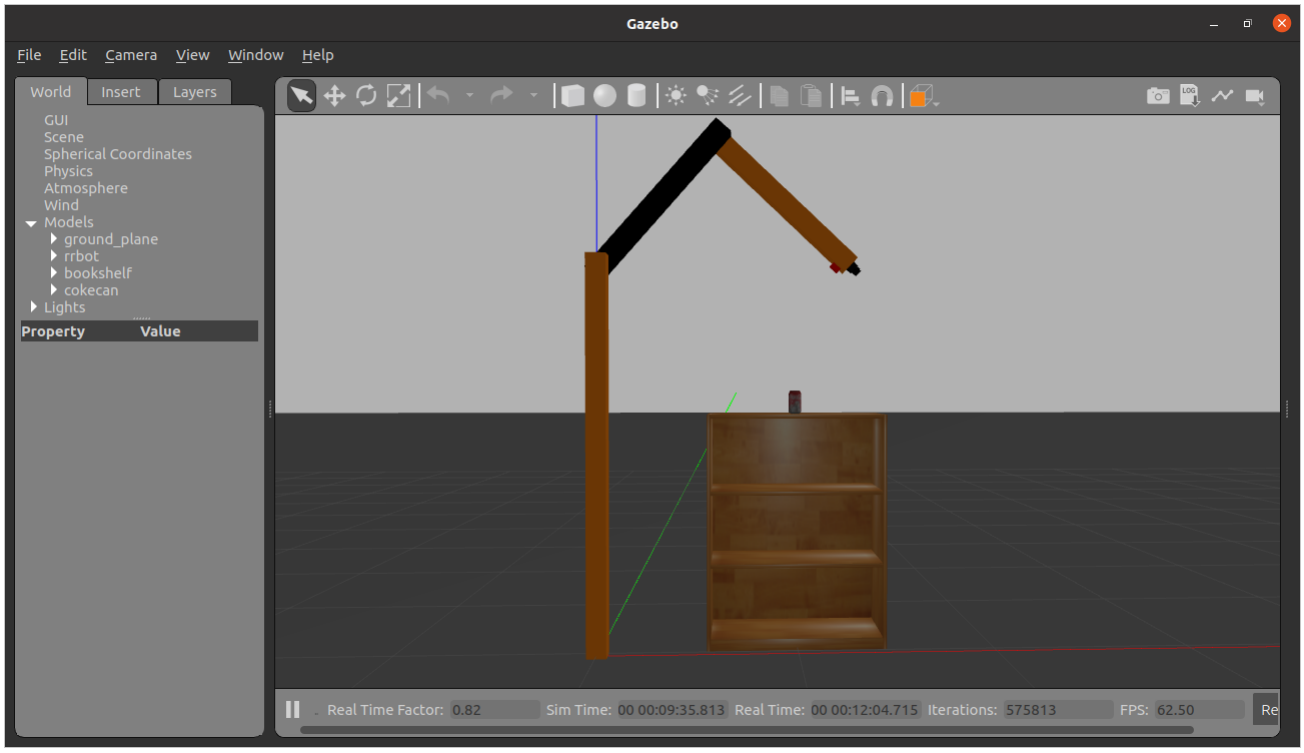

Fig. 11: RR manipulator with bookshelf and coke can

The bookshelf is fixed to the ground and is rigid. The coke can rests on top of the bookshelf, but is not attached to it, and can be pushed off. The goal of the lab is to use position control (Lab 3) and velocity control (Lab 4) to knock the coke can off the bookshelf. As the bookshelf is fixed, if the manipulator tries to push it, it will be stopped and hence, the manipulator must avoid it. When using position control, students will have to choose via points close to each other to ensure the manipulator does not run into the bookshelf as there is less control over the trajectory. When using velocity control, it should be possible (in theory) to knock over the coke can using 2 straight-line trajectories (as done previously in Lab 2) and 1 via point; using Figure 11 as an example, the first step would be getting the end-effector to a position parallel to the coke can – from position (2, 1.4, 0.05) to position (1.25, 1.4, 0.05) and then performing another straight-line trajectory to knock the can off.

4. Analysis of Simulation and Labs

This section will be broken into 2 subsections – response analysis and lab validation. Response analysis will go over the response of the model to a command. Lab tasks will go over the calculations and steps required to accomplish labs 1-4.

4.1 Response Analysis

From equation 6 and 7, we get an approximation of the transfer function of the position vs torque and velocity vs torque. The PID parameters were set as follows:

$$K_p = 100 \qquad K_i = 0.01 \qquad K_d = 10$$

Since equation 6 and 7 do not take account of the dynamics of the systems, the simulation was run by keeping the value of joint 1 at 0 (for both position and velocity) and setting the command value of joint 2 at 1 rad (or rads^{-1} for velocity).

Figure 12 shows the step response of the position-torque function. Figure 13 shows its simulation counterpart. Figure 14 shows the position of joint 1 over time. Figure 15 shows the velocity-torque transfer function and Figure 16 its simulation counterpart. Since it is hard to distinguish between the actual and commanded velocities in Figure 16, Figure 17 shows a magnified portion of Figure 16. Figure 18 shows the velocity of joint 1 over time. The step response was chosen as commanded position (or velocity) is sent as a step function.

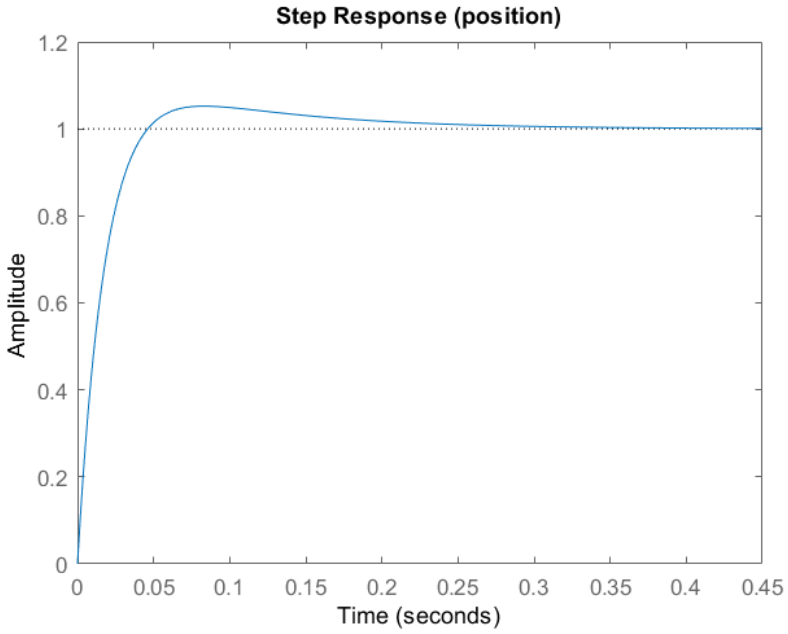

Fig. 12: Step response of position by torque as calculated by Matlab using equation 6

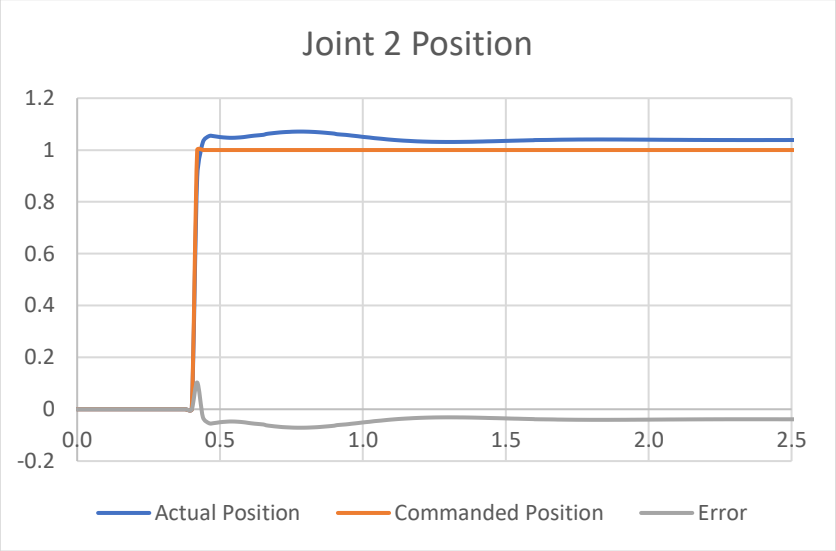

Fig. 13: Simulation position response

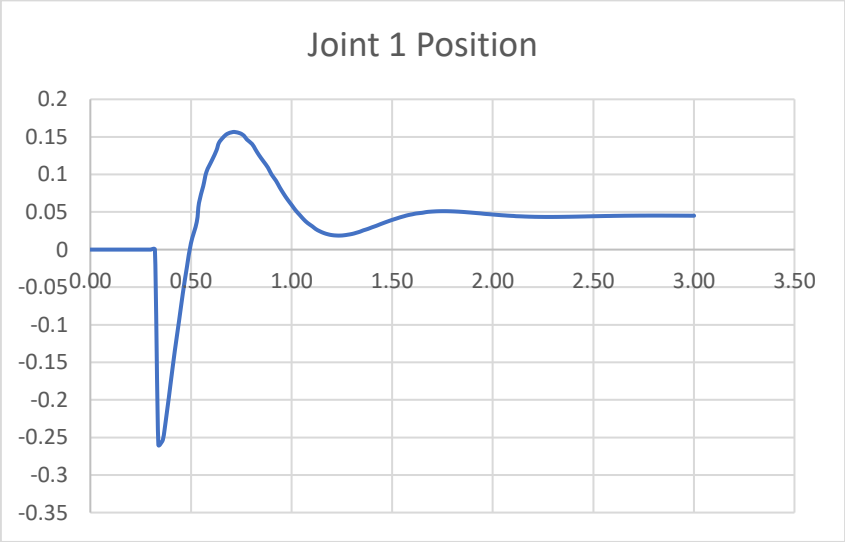

Fig. 14: Joint 1 position over time as a result of the movement at joint 2

When comparing the results plotted, it is worth noting that even with the approximation ignoring gravity and other dynamic effects, the simulated response matches very closely with the step response plotted by Matlab, Having a similar shape, rise time and settling time. The differences are that there is a steady state error and the simulation response has more complex frequencies in the signal. This can be attributed to the effects of gravity and variation in the position of joint 1 over time. For example, when joint 2 rises, it causes a jerk that pushes joint 1 away from its initial position. This in turn, causes the torque due to gravity on joint 2 (which is dependent on the position values of joint 1 and joint 2) to fluctuate, giving a more complicated response.

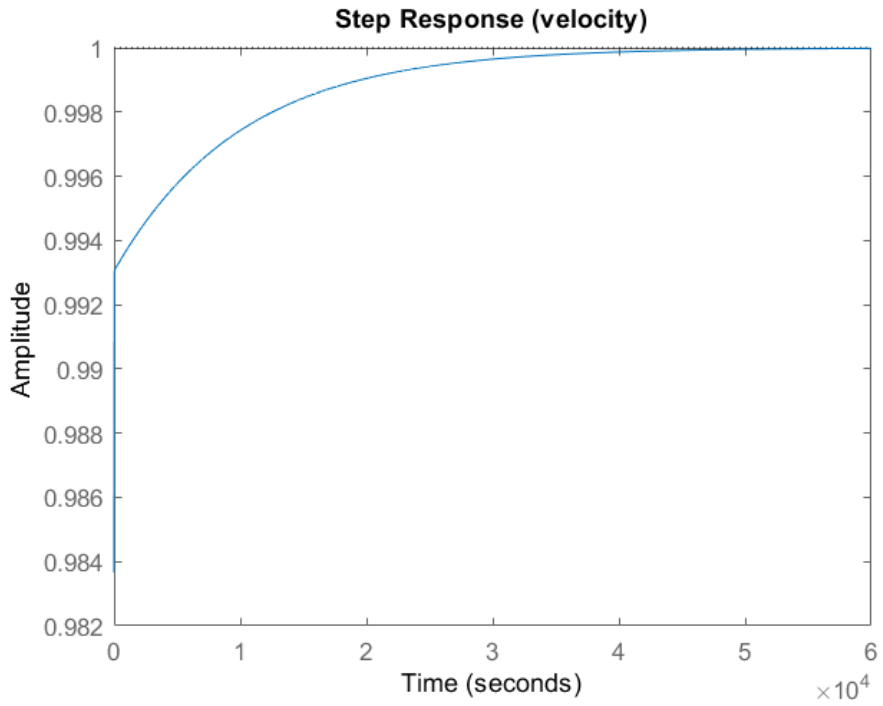

Fig. 15: Step response of velocity by torque as calculated by Matlab using equation 7

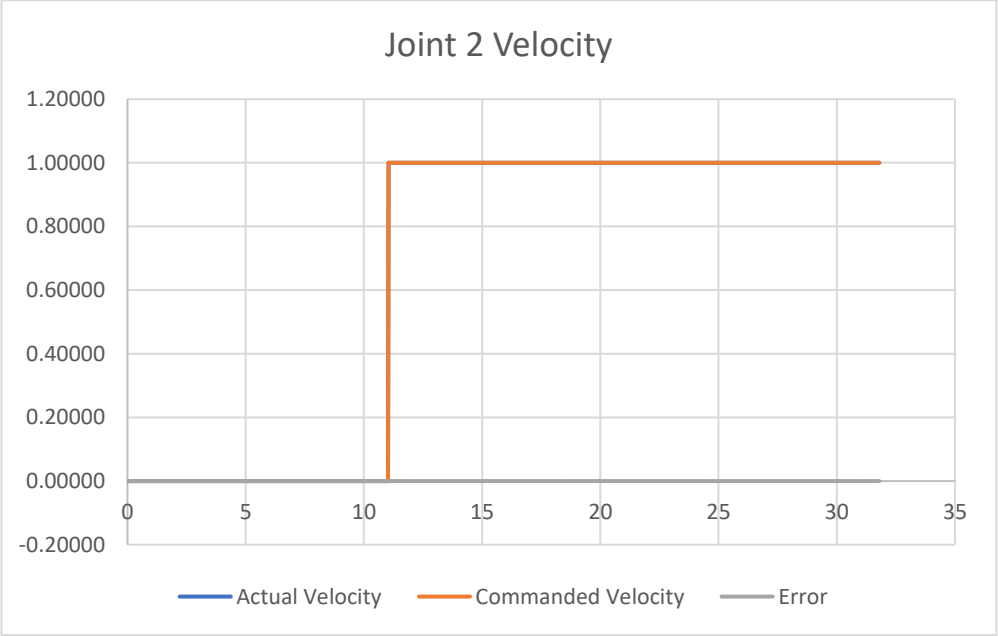

Fig. 16: Simulation velocity response

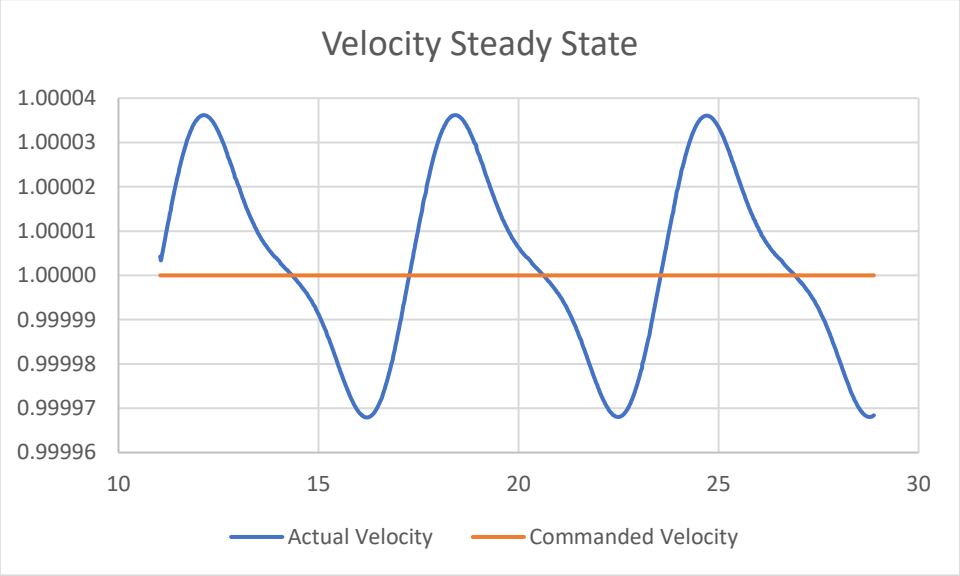

Fig. 17: Magnification of rise (left) and steady state (right)

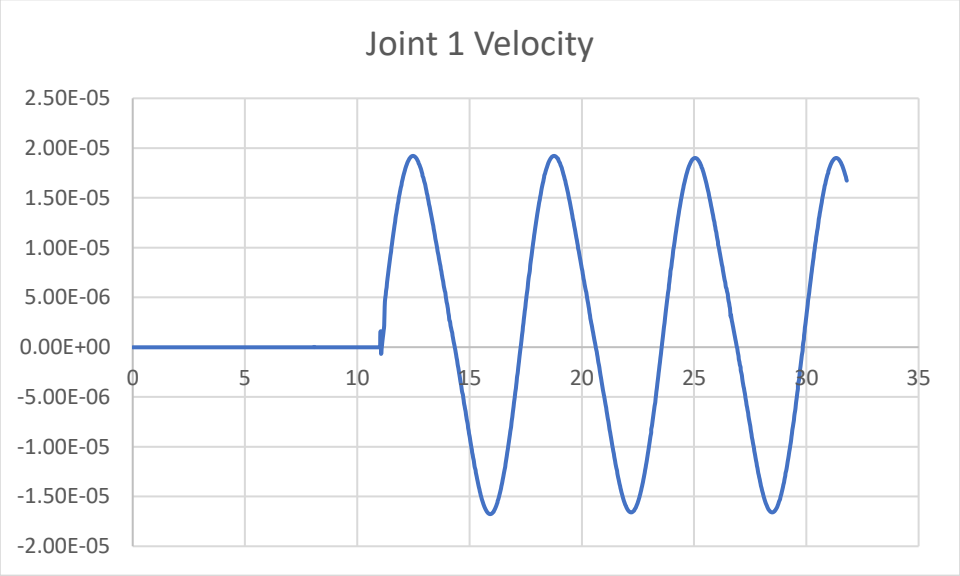

Fig. 18: Joint 1 velocity over time

As with position, the simulated response for velocity matches very closely with the Matlab plotted velocity step response. As shown in the Matlab plot, output rises to 99.3% of its final value instantly. However, settling time is over 6×10^4 seconds. While it is possible to run a long simulation, presenting it in a graph is difficult (as the change takes place over a long period of time, graphs will look steady).

Again, due to the dynamics of the robot, a complex frequency response is output. This response is reflected in joint 1's velocity response (Figure 18).

4.2 Lab Tasks

4.2.1 Lab 1

The task in Lab 1 is to move the end effector from its initial position (0, 4, 0.05) to positions (2, 1.4, 0.05) and (2, -1.4, 0.05). The first step is to calculate the robot's geometrical transforms. This can be done by writing down the Denavet-Hatenburg (DH) parameters. Figure 19 shows the position of the joints assigned to the model. The co-ordinate system is the same as in the Labs section (blue – x axis, red – y axis, green – z axis). Putting joints.

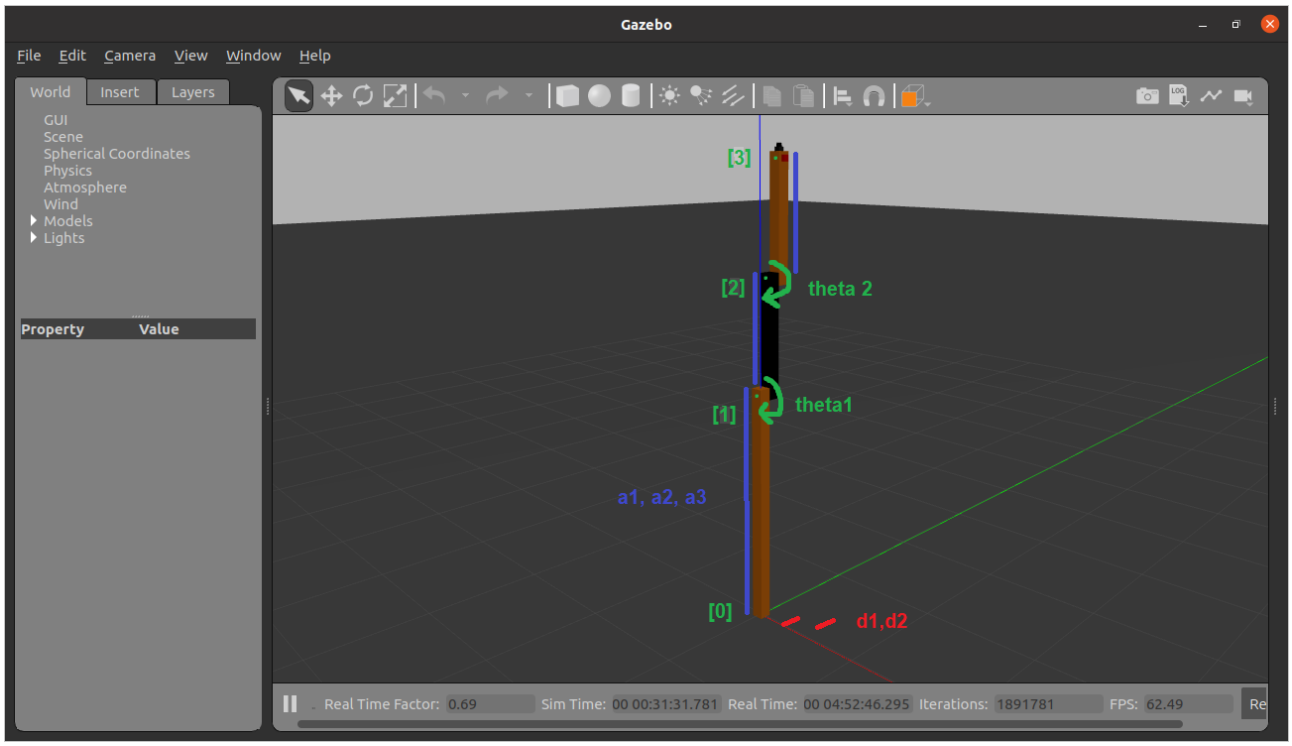

Fig. 19: Joint frames on RR manipulator

Table 3 gives the DH parameters for this configuration.

Table 3: DH Parameters of RR manipulator

Frame	Link Length (a) /m	Link twist (α)	Link offset (d) /m	Joint angle (θ)
[0]	0	0	0	0
[1]	2	0	0	θ_1
[2]	1	0	0.025	θ_2
[3]	1	0	0.025	0

Using the DH Parameters, the forward kinematics of the manipulator can be calculated using the formula:

$$ {}^{m-1}_{m}T = \begin{pmatrix} c\theta_m & -s\theta_m & 0 & a_{m-1} \\ s\theta_m c\alpha_{m-1} & c\theta_m c\alpha_{m-1} & -s\alpha_{m-1} & -s\alpha_{m-1} d_m \\ s\theta_m s\alpha_{m-1} & c\theta_m s\alpha_{m-1} & c\alpha_{m-1} & c\alpha_{m-1} d_m \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (8) $$

Using equation 8 [25], transforms for each frame are as follows:

$$ {}^{0}_{1}T = \begin{pmatrix} c\theta_1 & -s\theta_1 & 0 & 2 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{1}_{2}T = \begin{pmatrix} c\theta_2 & -s\theta_2 & 0 & 1 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0.025 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

$$ {}^{2}_{3}T = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.025 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Multiplying the 3 transforms gives us the transform:

$$ {}^{0}_{3}T = \begin{pmatrix} c(\theta_1+\theta_2) & -s(\theta_1+\theta_2) & 0 & 2+c(\theta_1)+c(\theta_1+\theta_2) \\ s(\theta_1+\theta_2) & c(\theta_1+\theta_2) & 0 & s(\theta_1)+s(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (9) $$

Note that it is possible to start at frame [1], but since the position is desired relative to the origin, frame [0] is located at the origin. To solve for the inverse kinematics, equation 9 is set equal to the demanded x and y positions, giving equations 10 and 11.

$$ c(\theta_1)+c(\theta_1+\theta_2) = (x-2) \quad (10) $$

$$ s(\theta_1)+s(\theta_1+\theta_2) = y \quad (11) $$

Note that for simplicity of calculation, $x-2$ will be considered u. Squaring and adding equation 10 and 11, we get:

$$ 2+c\theta_2 = u^2+y^2 \quad (12) $$

Rearranging:

$$ c\theta_2 = \frac{u^2+y^2-2}{2} \quad , \qquad \theta_2 = \pm acos(\frac{u^2+y^2-2}{2}) \quad (13) $$

Using trigonometric identities:

$$ \sin(a+b) = \sin a \cos b + \cos a \sin b $$

$$ \cos(a+b) = \cos a \cos b - \sin a \sin b $$

Equation 10 and 11 can be rewritten as:

$$(1 + c\theta_2)c\theta_1 - (s\theta_2)s\theta_1 = x \quad (14)$$

$$(s\theta_2)c\theta_1 + (1 + c\theta_2)s\theta_1 = y \quad (15)$$

The terms in red are constants and can be replaced by K_1 and K_2 as follows:

$$K_1 = 1 + c\theta_2$$

$$K_2 = s\theta_2$$

Using the substitutions:

$$r = +\sqrt{K_1^2 + K_2^2}$$

$$\lambda = Atan2(\frac{K_2}{K_1})$$

Equations K_1 and K_2 can be simplified to:

$$K_1 = rcos(\lambda)$$

$$K_2 = rsin(\lambda)$$

And consequently, equations 9 and 10 can be simplified to:

$$\frac{x}{r} = c\lambda c\theta_1 - s\lambda s\theta_1 = c(\lambda + \theta_1) \quad (16)$$

$$\frac{y}{r} = s\lambda c\theta_1 + c\lambda s\theta_1 = s(\lambda + \theta_1) \quad (17)$$

Dividing equation 12 by 11:

$$\frac{y}{x} = tan(\lambda + \theta_1) \quad , \quad \theta_1 = Atan2\left(\frac{y}{x}\right) - \lambda \quad (18)$$

Using equation 13 and 18, we can calculate the possible joint angles for the positions (2, -1.4, 0.05) and (2, 1.4, 0.05). They come out to be approximately $(\theta_1, \theta_2) \approx (45, 90)\ or\ (135, -90)$ for position (2, 1.4, 0.05) and $(225, 90)\ or\ (315, -90)$ for position (2, -1.4, 0.05).

4.2.2 Lab 2

Since Lab 2 is based on velocity control, the Jacobian will need to be calculated to find out appropriate joint velocities. This can be calculated via partial differentiation w.r.t. time.

Using equation 10 as an example:

$$2 + c(\theta_1 + \theta_2) + c\theta_1 = x$$

$$\frac{\partial x}{\partial t} = (-s\theta_1 - s(\theta_1 + \theta_2))\dot{\theta}_1 - (s(\theta_1 + \theta_2))\dot{\theta}_2$$

The terms in red are the elements of the Jacobian. Equation 11 is differentiated similarly. The resulting Jacobian is as follows:

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} -s\theta_1 - s(\theta_1 + \theta_2) & -s\theta_1 \\ c\theta_1 + c(\theta_1 + \theta_2) & c\theta_1 \end{pmatrix} \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{pmatrix} \tag{19}$$

Using equation 19, the inverse Jacobian can be calculated as follows:

$$J^{-1} = \frac{1}{s\theta_1\big(c(\theta_1 + \theta_2)\big) - c\theta_1\big(s(\theta_1 + \theta_2)\big)} \begin{pmatrix} c\theta_1 & s\theta_1 \\ -c\theta_1 - c(\theta_1 + \theta_2) & -s\theta_1 - s(\theta_1 + \theta_2) \end{pmatrix} \tag{20}$$

Since the task is to travel in a straight line parallel to the y axis, $v_x = 0$, and the values for joint velocities can be calculated using equation 21.

$$\begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{pmatrix} = J^{-1} \begin{pmatrix} 0 \\ k \end{pmatrix} \tag{21}$$

Where k is a constant and the values of θ_1 and θ_2 will updated by feedback.

4.2.3 Lab 3

Lab 3 uses position controllers to manoeuvre the end effector in a way that it knocks over the coke can without colliding into the bookshelf. The initial position is taken to be (2, 1.4, 0.05) as shown in Figure 20.

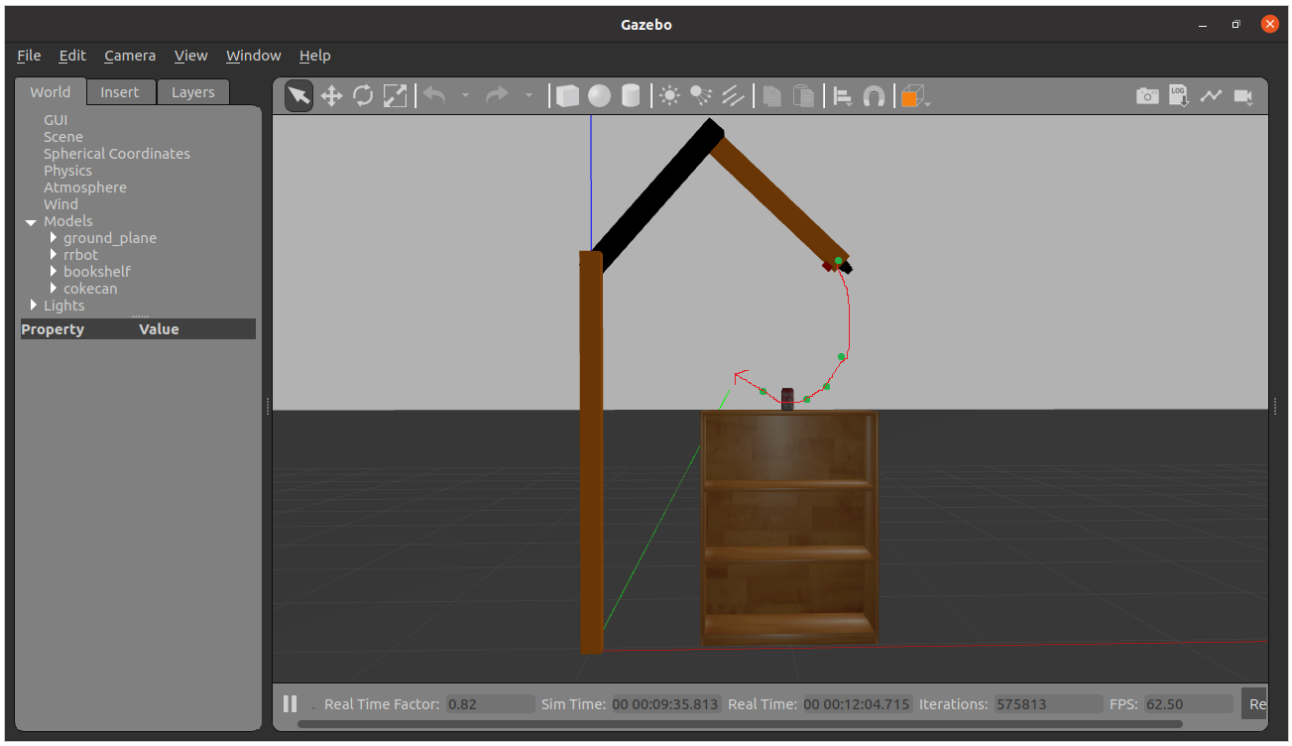

Fig. 20:

To ensure the manipulator does not collide with the bookshelf, more via points are taken as the end-effector approaches the target. A combination of points, such as (1.4, 1.4, 0.05), (1.3, 1.2, 0.05), (1.25, 1.1, 0.05) and (1.3, 0.85, 0.05) were used in the testing of the lab.

The joint positions are calculated as per equations 13 and 18 and are sent as step commands to the joint position controllers

4.2.4 Lab 4

Lab 4 performs the same task as lab 3, but with velocity control. Starting at the same position as in lab 3, the task can be performed in 2 movements using 1 via point. The via points chosen for the simulation were (1.4, 1.25, 0.05) and (0.6, 1.25, 0.05). The joint command velocities were calculated using the inverse Jacobian (equation 20).

Resulting motion is much smoother than using position control with many via points. However, it is more calculation intensive – as the inverse Jacobian needs to continually be computed – and so, for more complex systems, a position control based algorithm might be preferred depending on the computational power available.

5. Future Prospects

The project has a lot of scope for future work. This work can be vastly broken down into 2 categories:

1. Improvements to the current model
2. Additions to the model

One of the several ways to improve the current model, is adding different types of robots (eg. a SCARA, a cartesian, etc). This will provide more variety to the labs and allow for more practice (as the kinematics of the other models will have to be calculated from scratch). Another method is to add more links/elements – like a gripper to the end-effector – to the current model. This will not only increase the complexity of the model, but also allow for more complex operations, and with more complex operations, more nuances of the robotics field can be explored. Lastly, instead of modelling the robot out of simple geometrical shapes, a CAD mesh can be imported and converted into a urdf that can be simulated in Gazebo. By doing this, real industrial robots can be simulated, making the labs more and more similar to industrial simulation.

With regards to the possible additions to the model, sensors – a part which is fundamental to the robotics field and is taught by many universities in the literature review – are not touched upon. By adding a simple camera to the end-effector, it would be possible to perform image processing techniques. Image data then be combined into many fields of study such as SLAM algorithms, Machine Learning algorithms, etc. Of course sensors are

not limited to cameras; a proximity sensor could be added to detect position and correct the steady state error that arises in the position control labs (as in the Analysis of Response section).

6. Conclusion

The goal of this project was to create a virtual laboratory – that is easily accessed by students of different backgrounds – where robots can be simulated. 6 objectives were laid out at the beginning (shown in table 4):

Table 4: Objectives and Achievements

Objective	Achievement/Comments
Models must cover a significant portion of the course material	Models fully cover the kinematics portion – the fundamental material – of the module and touch upon dynamics, control and path-planning
Lab tasks should be basic initially, gradually increasing in complexity	Labs start with forward and inverse kinematics, then proceeding to Jacobians and velocity control, and finally touching on path planning
It should be possible to perform all tasks without software prerequisites	All tasks can be performed from a separate machine running Matlab, which is taught to all students
Simulation should mimic real-world physics, accounting for gravity, wind, friction, collision-interaction, etc.	The effects of gravity and dynamics are reflected in the Response Analysis section, clearly showing the effect in contrast to the hand-calculated approximation
All labs should have visualisation of the robot moving as per code	All labs can be visualised on Gazebo GUI
Project should provide a foundation for building more advanced labs in the future	Code and software used is all open source. Report

While there remains much to be done, this project is successful in achieving all the objectives set out at the beginning.

7. Reflection

This FYP was one of the most educational experiences in my university life. In 3rd year, after building a SCARA robot for my DMT, I grew an interest in robotics and decided to specialise in it, picking my modules and FYP topic accordingly. Coming from a hardware background, most aspects of the project were very new to me and took a lot of time to pick up. Some of the most notable challenges were learning how to use ROS, connecting ROS to Matlab, and learning xml (the language which most of the robot's files are written in). Funnily, this was very motivating, as the purpose of this project was to help students interested in robotics overcome the software barrier – something that I have personally experienced for years. Due to the learning curve of this project (and the time spent learning software), it lacks the amount of content or personal contribution a FYP would normally expect. However, I am very glad that I took this project as it taught me (apart from the software knowledge gained) 2 skills:

1. How to go about a project in a subject that I am not very comfortable in or sure that I can complete with my skillset.
2. Knowing when and how to plan/replan a project and its objectives when it looks like things are going south.

I would like to sincerely thank Dr. Cinosi for his continued patience, guidance and support. Due to the timeline constraints, the project is still somewhat incomplete (as discussed in the Future Prospects and Conclusion sections) and has a lot of scope. While incomplete, I believe that this project serves as a good base and someone software-challenged looking to pick up this topic (or simply learn ROS/try and code robotics algorithms in Matlab/etc.) can use this project as a reference, saving some time on the software learning.

8. References

[1] J. J. Castaneda, A. F. Ruiz-Olaya, W. Acuna, A. Molano (2016) 'A low-cost Matlab-based educational platform for teaching robotics', IEEE Colombian Conference on Robotics and Automation, accessed 18/5/2021

[2] M. Teulieres, J. Tilley, L. Boltz, P.L.M. Dehm, S. Wagner (2019) 'Industrial Robotics' [Online] Available at: https://www.mckinsey.com/~/media/mckinsey/industries/advanced%20electronics/our%20insights/growth%20dynamics%20in%20industrial%20robotics/industrial-robotics-insights-into-the-sectors-future-growth-dynamics.ashx

[3] Open Source Robotics Foundation, 'ROS.org' [Online] Available at: http://wiki.ros.org/

[4] G. Nicholas (2020) 'Robotics in business: Everything humans need to know' [Online] Available at: https://www.zdnet.com/article/robotics-in-business-everything-humans-need-to-know/

[5] N. Cinosi (2020) 'Introduction to Robotics' lecture notes, Imperial College London Department of Mechanical Engineering

[6] MATLAB (2017) 'Getting Started with MATLAB and ROS' [online video] Available at: https://www.youtube.com/watch?v=KPlzSsWWfLE&t=1168s

[7] T. Dear (2018) 'Robot Kinematics and Dynamics' Carnegie Mellon University [Online] Available at: https://www.cs.cmu.edu/~tdear/16384.html

[8] S. Sastry (2020) 'Introduction to Robotics' University of California Berkeley [Online] Available at: https://ucb-ee106.github.io/106a-fa20site/

[9] H. Asada, J. Leanard (2005) 'Introduction to Robotics' Massachusetts Institute of Technology [Online] Available at: https://ocw.mit.edu/courses/mechanical-engineering/2-12-introduction-to-robotics-fall-2005/index.htm

[10] Y. Takeda, K. Suzumori, Y. Sugahara (2020) 'Robot Kinematics' Tokyo Institute of Technology [Online] Available at: http://www.ocw.titech.ac.jp/index.php?module=General&action=T0300&GakubuCD=2&GakkaCD=321500&KeiCD=15&KougiCD=202001889&Nendo=2020&lang=EN&vid=03

[11] A. Gupta (2020) 'Robotics' Indian Institute of Technology Bombay [Online] Available at: https://portal.iitb.ac.in/asc/Courses

[12] P. Vadakkepat, A. S. Putra (2018) 'Robotic System Design' National Institute of Singapore [Online] Available at: https://www.eng.nus.edu.sg/wp-content/uploads/sites/10/2019/03/Descriptions-of-new-modules-for-RoboticsSpecialisation.pdf

[13] C. Osorio (2017) 'How a differential equation becomes a robot' [Online] Available at: https://uk.mathworks.com/videos/series/how-a-differential-equation-becomes-a-robot-95157.html

[14] Gazebo Simulator, 'Why Gazebo?' [Online] Available at: http://gazebosim.org/

[15] R. K. Megalingam, D. Nagalia, R. K. Pasumarthi, V. Gontu, P. K. Allada (2018) 'ROS Based, Simulation and Control of a Wheeled Robot using Gamer's Steering Wheel', IEEE International Conference on Computing Communication and Automation, accessed 20/5/2021

[16] A. Kumar (2020) ‘Learn about ROS, or Robot Operating System, a popular open-source middleware used in robotics!’ [Online] Available at: https://maker.pro/arduino/tutorial/how-to-use-arduino-with-robot-operating-system-ros

[17] A. F. M. Torres, A. B. Gomez (2012) ‘So Many Educational Microcontroller Platforms, So Little Time!’, ASEE Annual Conference & Exposition, accessed 21/5/2021

[18] Create a URDF for an Industrial Robot, ‘ROS.org’ [Online] Available at: http://wiki.ros.org/Industrial/Tutorials/Create%20a%20URDF%20for%20an%20Industrial%20Robot

[19] EVERYTHING YOU NEED TO KNOW ABOUT SCARA ROBOTS, ‘DIY Robotics’ [Online] Available at: https://diy-robotics.com/blog/scara-robots/

[20] D. Coleman (2017) ‘Gazebo ROS Demos’ Github repository [Online] Available at: https://github.com/ros-simulation/gazebo_ros_demos

[21] D. Tilbury, B. Messner (2017) ‘DC Motor Speed: System Modelling’, Carnegie Mellon University [Online] Available at: https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling

[22] T. Foote (2013) ‘tf: The Transform Library’, Open Source Robotics Foundation, IEEE International Conference on Technologies for Practical Robot Applications, accessed 20/4/2021

[23] nkoenig (2019) ‘gazebo_models’ Github repository [Online] Available at: https://github.com/osrf/gazebo_models

[24] WolframAlpha, Wolfram|Alpha Wigets, ‘WolframAlpha.com’ [Online] available at: https://www.wolframalpha.com/widget/widgetPopup.jsp?p=v&id=3272010f63d3145699ca78bbe0db05a7&title=Laplace%20Transform%20Calculator&theme=blue&i0=sin(e%5E(1t))&podSelect=&showAssumptions=1&showWarnings=1

[25] A. Didwania (2021) ‘Introduction to Robotics Coursework’ Imperial College London Department of Mechanical Engineering