University of Technology, Sydney

Faculty of Engineering and Information Technology

DEVELOPMENT OF IMPROVED CONTROL SOFTWARE FOR THE JEXO EXOSKELETON ROBOT

by

Luke Eyles

Student Number: 12883423

Project Number: SPR-22-09582

Major: Mechanical and Mechatronic Engineering

Academic advisor: Dr Marc Carmichael

A 12 Credit Point Project submitted in partial fulfillment of the requirement for the Degree of Bachelor of Engineering

7 November 2022

Statement of originality

I declare that I, Luke Eyles, am the sole author of this report. Any fragments of text or programming code from other sources have been used only with proper acknowledgement. Any theories, results, and designs of others have been appropriately referenced and all sources of assistance have been acknowledged.

Luke Eyles

L.Eyles

7 November 2022

Abstract

Development of New Control Software for the JEXO Exoskeleton Robot

Luke Eyles | SPR-22-09582

Supervisor: Dr Marc Carmichael

Major: Mechanical and Mechatronic Engineering

The UTS Robotics Institute developed an upper limb exoskeleton robot, JEXO. This robot was built to be a research platform to test control algorithms related to human-robot interaction. However, the robot's control software became outdated, having been authored in 2014 and not receiving any updates since 2017. It also lacked consistent documentation, making it time-consuming for newcomers to the project to develop a complete understanding of the software. There was also no way to test new software without access to the physical robot, compounding this problem.

In this project, the JEXO codebase was updated using modern ROS features and best practice conventions to improve maintainability, robustness, and ease of use when implementing new control systems. The theory behind the control systems implemented in the existing code was analysed, and alternative control strategies were researched in a literature review.

A simulation was developed to enable programming and testing control algorithms without requiring access to the physical exoskeleton. The Gazebo simulator was used together with integration from ROS Control. The architecture of ROS Control decouples controllers from the hardware, which enables the same controllers to be used for both the real robot and the simulation.

Detailed documentation has been created for the new codebase. Together with the improved modularity, this allows future controllers to be developed more easily as it reduces the time to learn and understand the control software. Overall, the work of this project has improved the functionality of JEXO as a research platform.

Acknowledgement

I would like to thank my supervisor Dr Marc Carmichael, for his advice and guidance throughout this project.

Table of Contents

List of Figures

List of Tables

Nomenclature

q	Column vector of robot joint positions
$\dot{q}$	Column vector of robot joint velocities
x	Column vector of task space position
$\dot{x}$	Column vector of task space velocity
J	Kinematic Jacobian
J_A	Augmented kinematic Jacobian
$J^\dagger$	Moore-Penrose pseudoinverse of the Jacobian
ϕ	Swivel angle

List of Abbreviations

DOF: Degree of freedom

JEXO: Jaco EXOskeleton

RI: Robotics Institute

UTS: University of Technology Sydney

ROS: Robot Operating System

1 Introduction

The Robotics Institute (RI) at UTS has a custom robotic exoskeleton arm called JEXO (Figure 1). This exoskeleton serves as a research platform for human-robot interaction control algorithms. This research could further exoskeleton rehabilitation for patients who have lost strength or range of motion of their upper arm, or be utilised in industrial applications to enhance normal human strength and reduce risk of injury.

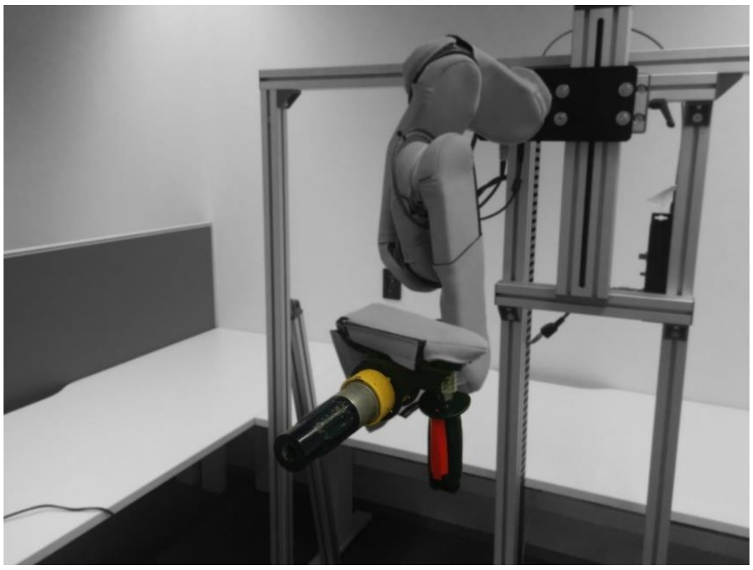

Figure 1: JEXO exoskeleton arm

The control software for JEXO was programmed in C++ using the Robot Operating System (ROS), which is an open source set of libraries and tools for robotic applications. The exoskeleton control software can be broken into two main parts. The first is the admittance controller, which takes input from the force/torque sensor on the handle of the robot and commands the velocity of the end effector of the robot in the task space. This required movement is then fed into the joint controller, which computes how to move the joints of the robot to execute the desired task.

1.1 Project Scope

The primary aims of this project are to implement a simulation of the exoskeleton so that development can be done outside the lab, to improve the codebase to facilitate later research projects, and to investigate alternative control solutions for potential future implementation.

With the original exoskeleton control software, there is no capability for testing without access to the physical robot. Implementing a simulation expedites future development by enabling immediate testing of any changes on the simulation, reducing time spent in the lab working out errors. This also enables people to work on the exoskeleton simultaneously, improving its value as a research platform.

The codebase can be improved by implementing best practice principles of documentation, modularity, and reusability. Current documentation is sparse, so building a well-documented codebase will reduce the time for newcomers to understand and add to the code. The software is also outdated, being built using Robot Operating System (ROS) Indigo, which was released in 2014. The codebase should be updated to ROS Noetic, the most recent version of ROS, to ensure compatibility with recent ROS packages and enable use of current ROS features.

The current admittance controller on the JEXO exoskeleton arm only uses the damping parameter, so the end effector velocity is simply proportional to the input force. This means that there is an inherent trade-off between stability and force required. This project investigates alterative admittance control methods, and recommends one to implement.

JEXO is a redundant robot, having five joints but only being controlled in the task space along three axes. The current redundant joint controller uses task space augmentation to choose the solution, adding one constraint task to solve for the redundancy in the shoulder and another constraint task to choose the optimal swivel angle of the elbow. However, there are many other methods of redundant robot control that can be used to optimise for different parameters, such as singularity avoidance, minimum joint velocity, avoidance of joint limits, and collision avoidance. This project investigates other potential redundant joint controllers that could be implemented on JEXO.

1.2 Context and Impact

The societal impact that a rehabilitative exoskeleton arm would have is to increase access to physical therapy for those who require it, due to injuries or other conditions such as a stroke. According to the ABS (2012), 1.8% of Australians have suffered from a stroke, with 35% of those having an impairment that lasts for six months or longer because of this. There is therefore a large population of people who would benefit from using a rehabilitative exoskeleton arm.

Strokes can cause muscle weakness, fatigue, muscle tightness, and muscle contractures in the arms. Physical therapy is needed to assist with recovery, helping to achieve functional levels before the stroke and preventing deterioration (López-Liria et al. 2016). One of the most effective types of physical therapy for the upper limbs is task-oriented training, where the patient repeatedly trains performing everyday tasks (Van Peppen et al. 2004). A rehabilitative arm exoskeleton can assist with this type of training. Training is most effective when there is a high level of transparency, meaning that the task should be clear to the patient and the robot should not force the patient to move in a particular way (Nathanael et al. 2014).

2 Literature Review

A literature review was performed on current methods of exoskeleton control. The review was split into two components, admittance control strategies in the task space, and redundant robot control methods to map desired task space velocities to joint space velocities.

2.1 Task Space Control

The most common approach to exoskeleton control in the task space is admittance control. In this system, the exoskeleton is treated as a mechanical admittance, mapping a force input to a motion output (Figure 2). The exoskeleton is modelled as a mass-damper system, which can be described by:

$$m\ddot{x} + c\dot{x} = F$$

Stiffness in the environment can cause instability, so the parameters of the controller need to be tuned to ensure the exoskeleton is stable and accurate but does not require a large amount of effort to use.

Dimeas and Aspragathos (2016) designed a controller where the admittance control gains were dynamically changed during operation to maintain stability while being highly responsive. They did this by monitoring for high frequency force oscillations and increasing the virtual inertia when more stability is needed, as they found that it was better to increase the inertia parameter than the damping parameter to increase stability while minimising the effort required from the operator.

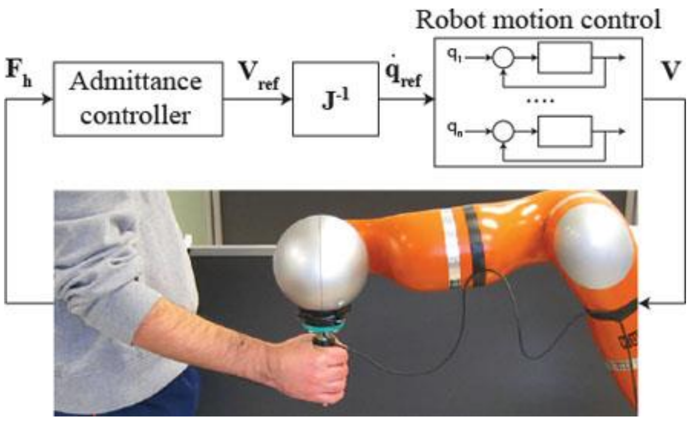

Figure 2: Admittance controller diagram (Source: Dimeas and Aspragathos 2016)

The EXO-UL7 7 DOF upper limb exoskeleton developed by UCSC used PID admittance control and applied the relation: $\dot{x_d}(s) = (M_a s + B_a + D_a/s)f_d(s)$ (Yu et. al. 2011). Their exoskeleton has 3 force/torque sensors at the upper arm, lower arm, and hand, which are combined using a weighted sum to the single command signal f_d. To determine the control parameters, they modelled the human impedance system and developed tuning rules for the admittance controller based on the human impedance.

Another exoskeleton is the 5 DOF MGA Exoskeleton, which is a rehabilitative exoskeleton for which Carignan et al. (2007) developed a modular control system. The control system takes in force and torque input from a sensor at the hand, and force input from two sensors at the elbow. The modularity of the system means it can be configured to change between impedance control and admittance control for different joint groups. The system was modelled as a pure damper when in admittance control mode, with linear damping set to 250 N/m/s during a path following task.

2.2 Redundant Robot Control

The admittance controller for the robot gives a desired velocity in the task space. To control the joints of the robot, this task space velocity needs to be mapped to a joint space velocity. The equation for the end effector velocity given the velocity of the joints is $\dot{x}(q) = J\dot{q}$, where $\dot{x}$ is a column vector of the end effector velocity in the task space, J is the Jacobian matrix, and $\dot{q}$ is a column vector of joint velocities. For a given task space velocity on a redundant robot, there are infinite solutions for $\dot{q}$, so the challenge is in choosing the best solution.

There are many different methods of solving this problem that allow for optimising different factors, such as singularity avoidance, joint limit avoidance, obstacle avoidance, reducing the joint torques, and reducing the potential energy of the robot's posture. These methods can be divided into two main approaches to solving the inverse kinematics for a redundant robot, task space augmentation and gradient projection.

2.2.1 Task Space Augmentation

One method of controlling redundant robots is task space augmentation, which is the current method used on the JEXO. Task space augmentation is where one or more constraint tasks are added to the task space, making the number of task space parameters equal to the number of joints. The added constraint makes the number of possible solutions finite so that the joint velocities can be solved for.

According to Siciliano (1989), the augmented task space can be characterised as:

$$x_A = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f(q) \\ f_y(q) \end{bmatrix}$$

With x being the original task, y being the constraint task, f(q) being the forward kinematic function, and $f_y(q)$ being the constraint task function. Differentiating this gives the augmented Jacobian:

$$\dot{x}_A = J_A(q)\dot{q}$$

Using task space augmentation, it is possible to encounter problems where it is not possible to perform both the original task and the constraint task (Chiacchio et al. 1991), causing an algorithmic singularity. To solve this, a task priority strategy utilising the null space of the Jacobian can be developed so that the constraint task does not interfere with the original task. Chiacchio considers case studies where this strategy can be used to make the body of the robot follow a path, make the robot maintain a dextrous configuration, or avoid an obstacle.

The EXO-UL7 at UCSC uses this approach to resolve the redundancy of their 7-DOF exoskeleton arm. They defined one constraint task as $x_a = h(q)$, with the gradient of the optimisation function $\dot{h}(q)$ chosen in the null space of the robot (Yu et al. 2011).

2.2.2 Gradient Projection

Another control strategy is the gradient projection method (Siciliano 1989). In this method, an initial solution for $\dot{q}$ is found by using the pseudoinverse of the Jacobian:

$$\dot{q} = J^{\dagger}(q)\dot{x}$$

Then a term can be added to this solution that only moves the robot in the null space, and does not change the movement of the end effector:

$$\dot{q} = J^{\dagger}(q)\dot{x} + k[I - J^{\dagger}(q)J(q)]\dot{q_0}$$

By changing the value of $\dot{q_0}$, the pose of the robot can be adjusted in the null space. A cost function can then be defined as $h(q)$, and by setting $\dot{q_0} = \dot{h}(q)^T$, the cost function can be maximised or minimised by setting k positive for maximisation and negative for minimisation. For example, one cost function used by Zghal, Dubey and Euler (1990) optimises for minimal joint velocity by using the sum of squares of the joint velocities in the cost function.

Another possible criterion is singularity avoidance. The gradient of the cost function for the manipulability can be formulated numerically by iterating over each joint, and testing the manipulability when the joint is moved forward or backward. The manipulability gradient can also be computed symbolically using the DH parameters of the robot, using a recursive method described by Park et al. (1999).

Dubey and Luh (1987) use the gradient projection method to optimise flexibility of a redundant robot. They describe the manipulator velocity ratio, which is the ratio of the end effector velocity norm to the joint velocity norm. The cost function is then defined as the square of the manipulator velocity ratio along the minor axis of the manipulability ellipsoid, which has the effect of heavily penalising large joint velocities along the direction of poorest manipulability. The result of this optimisation on the motion of a three-link planar robot is shown below.

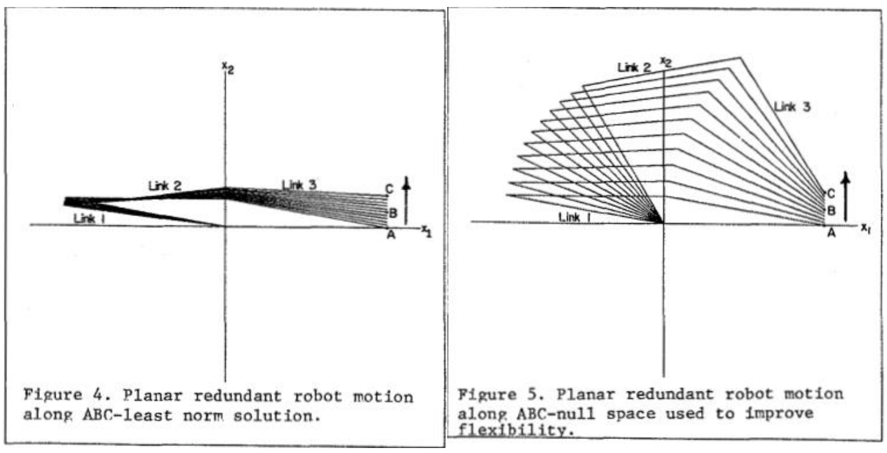

Figure 4. Planar redundant robot motion along ABC-least norm solution.

Figure 5. Planar redundant robot motion along ABC-null space used to improve flexibility.

Figure 3: Simulation of redundant robot control for higher flexibility (Source: Dubey & Luh 1987)

Liegeios (1977) formulated a cost function to achieve avoidance of joint limits, which pushes the system to the centre point of each joint range and away from the limits:

$$h(q) = \frac{1}{n}\Sigma_{i=1}^{n}\left(\frac{q_i - q_{i,mid}}{q_{i,max} - q_{i,min}}\right)^2$$

Obstacle avoidance can also be achieved. Khatib (1985) describes constructing an artificial potential field, which exerts a repulsive force on the manipulator parts as a function of the shortest distance from the part to the obstacle. The cost function can be set to minimise this force.

Another paper by Huo and Baron (2008) considers two performance criteria, singularity avoidance and avoidance of joint limits. They propose a combined performance criterion that considers both of these tasks, and use gradient projection with the negative gradient of this performance criterion to optimise the performance of a welding robot.

Tatlicioglu et al. (2008) develop a control strategy similar to the gradient projection method for a quaternion-based controller. They define measures that can be used to optimise for the sub task objectives of singularity avoidance, joint limit avoidance, bounding the impact forces of the end effector, and bounding the potential energy of the robot. Instead of a gradient-based approach, it is also possible to use these measures in a least-squares algorithm (Tatlicioglu et al. 2009).

The step size k of the null space term must be chosen large enough that the optimisation is efficient, but not so large that the minimum cost point is skipped over (Siciliano et al. 2016). This can be done using line search methods, such as Armijo’s rule.

2.2.3 Enforcing joint limits

A method of control for robots with hard joint constraints is Saturation in the Null Space (Flacco et al. 2015). This algorithm is iterative, and starts by solving for the joint states to satisfy a given task. It then checks if the constraint is exceeded and finds the joint that would most exceed its position, velocity, or acceleration constraints. It then scales the desired task down until the solution is within the joint’s constraints, and this process is repeated until all joints are within constraints.

2.2.4 Swivel Angle Control

The redundancy in the human arm is termed the “swivel angle”, which is the angle of the elbow about the axis connecting the shoulder to the wrist.

One method of controlling the swivel angle was proposed by Kim et al (2011). This paper hypothesised that a human orients their arm such that their palm can be most efficiently retracted to the head. Efficiency of movement in a certain direction can be quantified using the manipulability ellipsoid, shown in the figure below.

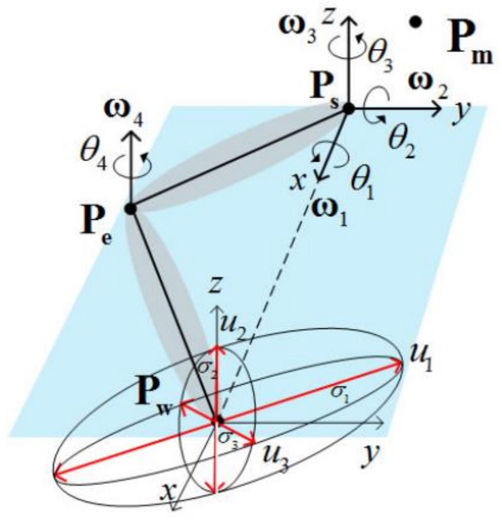

Figure 4: Manipulability ellipsoid (Kim et al. 2011)

To maximise efficiency of movement of the hand to the mouth, the projection of the longest axis of the manipulability ellipsoid (σ_1) onto the line connecting the mouth to the hand is maximised. This means that the points of the mouth, shoulder, elbow, and wrist all lie on the same plane. To test this, motion of subjects performing tasks was captured and compared with the predicted swivel angle. An average of 5 degrees of error was found using this method. Further testing found that using this method reduced the energy exchange between the robot and user by 11.22%, with users exerting less force to perform the same task (Kim, Miller et al. 2012). An extension of this method was proposed to estimate the dynamic swivel angle based on a muscle model of the human arm (Kim, Roldan et al. 2012).

Another method proposed by Kang et al (2005) solved for the swivel angle by minimising the total work done by joint torques. This paper found that joint trajectories matched subject arm movements with an r^2 generally > 0.98. However, in analysis of this method by Kim et al (2012), it was found that this method did not accurately predict the position of the human arm at low velocities. Due to gravity, the elbow would be located at the lowest possible position when static, which is not accurate to the human arm.

Another paper by Wang et al (2019) examined participant arm motion while performing reaching and grasping tasks. They found that the variation in swivel angle of the subjects was small, with a standard deviation of 5 degrees. This paper proposes a simple approach to resolve the redundancy by setting the swivel angle to the mean value measured, 155 degrees.

3 Original JEXO Design

3.1 JEXO Hardware Design

JEXO is a 5-DOF robot, with all joints being revolute (RRRRR). The shoulder is comprised of four joints, which makes the shoulder assembly redundant, as four joints control the rotation along three axes. This redundancy was built in to manage the singularities associated with 3-joint gimbal mechanisms, which occur when all three joint axes lie on the same plane (Carmichael & Liu n.d). The axes of rotation of each shoulder joint intersect at a common point corresponding with the location of the shoulder joint of the user.

The shoulder mechanism was then optimised to maximise the upper limb reachable workspace, while avoiding singularities and collisions with the user. This optimisation defined the bend angles of each link and achieved a range of motion of 97.7% that of the human arm.

JEXO was built using the actuators and joint controllers from the first generation Kinova Jaco, connected with custom-made aluminium links. The specifications of the actuators are detailed in Appendix 2.

The Kinova Jaco uses USB 2.0 to communicate with the main control system, and CAN bus to communicate from the main controller to the joint controllers. The main control system operates at a frequency of 100 Hz, and the CAN bus can transfer data at 1 Mb/s. When operating at the maximum rate of 100 Hz, this gives a theoretical maximum of 2 kB per joint.

Specification	**Value**
Control system frequency	100 Hz
CPU	270 MHz
CAN bus transfer speed	1 Mb/s

The Net F/T system is used to detect force at the end effector, which consists of a Nano25 force/torque sensor and a Net Box. The force/torque sensor is mounted using a quick release camera mount to allow for easy attachment and detachment. M3x7mm bolts are used to mount the sensor to the quick release plate, and M3x3.5mm bolts are used to mount the handle to the sensor. The bolt holes are through holes, so the bolt depth must not be exceeded as this can push the two plates of the sensor apart and damage the strain gauges (ATI Industrial Automation 2010).

Table 3.1: Nano25 sensing range and resolution (Source: ATI Industrial Automation 2010)

	Sensing Range				**Resolution**			
	Fx,Fy	**Fz**	**Tx,Ty**	**Tz**	**Fx,Fy**	**Fz**	**Tx,Ty**	**Tz**
SI-125-3	125 N	500 N	3 Nm	3 Nm	1/48 N	1/16 N	1/1320 Nm	1/2640 Nm
SI-250-6	250 N	1000 N	6 Nm	3.4 Nm	1/24 N	1/8 N	1/660 Nm	1/1320 Nm

The force/torque sensor is communicated with via Net Box, which allows network connection and configuration of the force/torque sensor.

3.2 JEXO Control System

The control system for the arm consists of 3 controllers, as shown in Figure 5. This project focuses on the first two controllers, the admittance controller and the redundant joint controller.

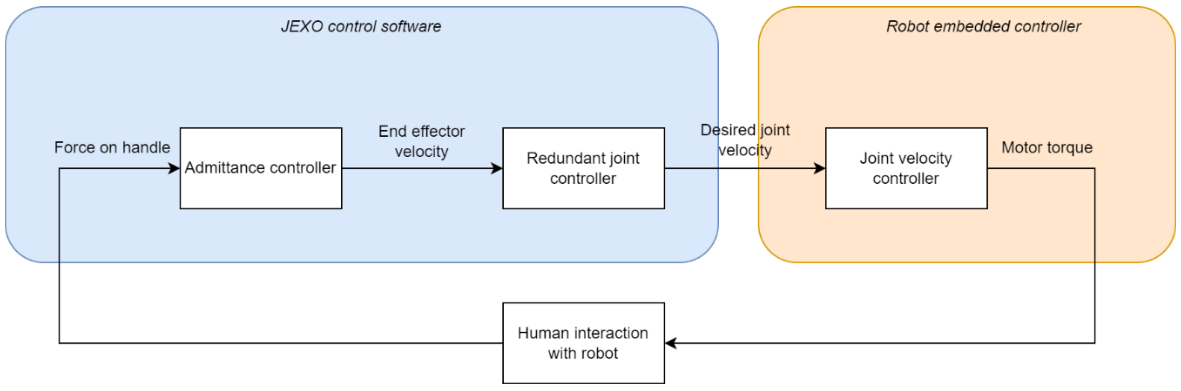

Figure 5: JEXO control system overview

The admittance controller takes the force the user applies on the robot's handle as input and determines a cartesian velocity to drive the robot's end effector to respond to this force.

The redundant joint controller takes the desired end effector velocity produced by the admittance controller, and determines the joint velocities required to drive the end effector at this velocity. JEXO has three degrees of freedom in the task space, which are linear motion in the x, y, and z directions. There are two degrees of redundancy, one in the shoulder which uses four joints to control rotation on three axes, and the swivel angle of the elbow about the line connecting the shoulder to wrist. The redundant joint controller therefore must solve for both the redundancy in the shoulder and the swivel velocity to achieve the linear velocity specified by the admittance controller, while maintaining a high degree of manipulability and avoiding collisions with itself or the user.

Once the desired joint velocity has been computed, the Jaco API is used to communicate the command to the robot. The embedded joint controllers use PID control to control the output motor torque.

3.2.1 Admittance Controller

The admittance controller models the end effector as a mechanical damper, where the velocity of the end effector is proportional to the forces exerted on it. This is given by the formula: $c\dot{x} = \Sigma F$. A force torque sensor is used to measure force exerted by the user on the end effector of the robot. The force measured is first transformed to the base frame of the robot, and the desired linear velocity of the end effector is computed as:

$$\dot{\mathbf{x}} = (\mathbf{f_h} + \mathbf{f_e})/c$$

Where $\mathbf{f_h}$ is the force exerted by the hand, $\mathbf{f_e}$ is the external force, and c is the damping coefficient set to 100.

3.2.2 Redundancy Resolution

The constraint task currently used on JEXO is a kinematic constraint, where the velocity of the second joint is set to be proportional to the velocity of the third joint. This constraint is expressed as:

$$q_2 = aq_3 + b$$
$$\dot{q_2} = a\dot{q_3}$$

Parameters a and b were determined in the optimisation process together with the joint angles. The best solution found $a = -0.914$ and $b = 5.5^o$.

To ensure this constraint is satisfied, the constraint error is computed as: $e = aq_3 + b - q_2$. In the task space, a constraint velocity of $K_c * e$ is calculated to drive the constraint error down. The constraint gain K_c is most commonly set to 10.

3.2.3 Swivel Angle Control

The method used to control the swivel angle of JEXO is similar to the method proposed by Kim et al (2012). This is implemented by first calculating the swivel axis t as the unit vector pointing from the shoulder to the wrist, $t = \frac{P_S - Pw}{|P_S - P_w|}$. An orthogonal unit vector *v* to *t* is computed as $v = norm([t_y, -t_x, 0])$. The cross product of *v* and *t* is then taken to complete the swivel angle reference frame, with transformation matrix T_{SA}.

$$T_{SA} = \begin{bmatrix} t & u & v & P_s \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

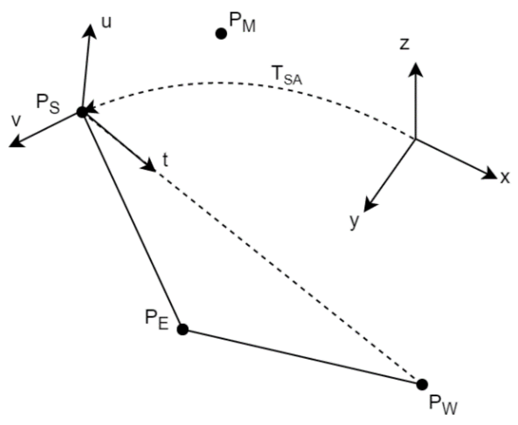

Figure 6: Swivel angle reference frame

The position of the elbow P_e is transformed into this reference frame, $P_{e,SA} = T^{-1}P_e$. The swivel angle is measured as:

$$\phi = atan2\left(-P_{,SA,y}, P_{e,SA,z}\right)$$

The position of the mouth is also transformed into this reference frame, $P_{m,SA} = T^{-1}P_m$. The optimal swivel angle is computed to be:

$$\phi_{opt} = atan2\left(P_{m,SA,z}, P_{m,SA,y}\right)$$

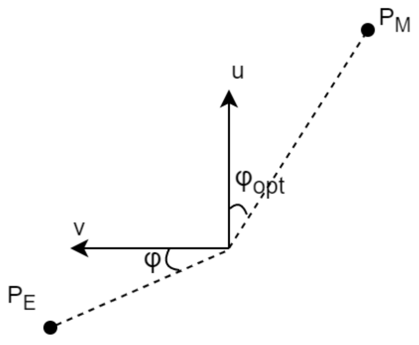

Figure 7: Swivel angle and desired swivel angle

The Jacobian of the swivel angle is calculated by multiplying the swivel axis by the rotational component of the Jacobian (Carmichael 2014):

$$\frac{d\phi}{dq} = t * J_\omega$$

The swivel velocity is calculated to be proportional to the swivel error if the error is outside a threshold.

$$\dot{\phi}_e = 0.5e_{SA} \text{ if } |e_{SA}| > 0.005$$

In addition to this, a distance sensor is used to detect the user's elbow and adapt the swivel angle to reduce collisions with the elbow.

$$\dot{\phi}_p = 0.2(d - 40)$$

The total swivel velocity is:

$$\dot{\phi} = \dot{\phi}_e + \dot{\phi}_p$$

3.2.4 Joint Velocity Calculation

As a result of the redundancy resolution, the augmented task space has 5 tasks: the original tasks of linear velocity on the x, y, and z axes, and the added tasks of the kinematic constraint velocity and swivel velocity.

$$\dot{x}_A = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ K_c * e \\ \dot{\phi} \end{bmatrix}$$

The augmented Jacobian is:

$$J_A = \begin{bmatrix} \frac{\delta x}{\delta q_1} & \frac{\delta x}{\delta q_2} & \frac{\delta x}{\delta q_3} & \frac{\delta x}{\delta q_4} & \frac{\delta x}{\delta q_5} \\ \frac{\delta y}{\delta q_1} & \frac{\delta y}{\delta q_2} & \frac{\delta y}{\delta q_3} & \frac{\delta y}{\delta q_4} & \frac{\delta y}{\delta q_5} \\ \frac{\delta z}{\delta q_1} & \frac{\delta z}{\delta q_2} & \frac{\delta z}{\delta q_3} & \frac{\delta z}{\delta q_4} & \frac{\delta z}{\delta q_5} \\ 0 & 1 & -a & 0 & 0 \\ \frac{\delta \phi}{\delta q_1} & \frac{\delta \phi}{\delta q_2} & \frac{\delta \phi}{\delta q_3} & \frac{\delta \phi}{\delta q_4} & \frac{\delta \phi}{\delta q_5} \end{bmatrix}$$

The augmented Jacobian can be used to map velocities in the augmented task space to joint velocities:

$$\dot{x}_A = J_A \dot{q}$$

Joint velocities can then be computed by inverting the augmented Jacobian:

$$\dot{q} = J_A^{-1} \dot{x}_A$$

To ensure that joint velocities don’t exceed the rated speed of the joints, the maximum joint velocity is checked. If this velocity exceeds the rated speed, the joint is set to the maximum and other joints are scaled proportionally.

3.3 Package Structure

The original implementation of the JEXO control software is in C++ using ROS Indigo, and consists of 13 ROS packages summarised in the table below.

Table 3.2: Original software package summary

Name	**Description**
daq_snowboard	Driver for the Arduino compatible platform Snowboard.
daq_dr2600	Driver for the USB-2600 series data acquisition board.
egismos_lrf	Driver and services to query the Egismos laser range finder.
icra_exp	Experiments related to the laser range finder.
jexo_control	Nodes to compute reference velocity for the joints.
jexo_core	Driver to communicate with the robot and URDF, SRDF, and mesh files to describe the robot model.
jexo_msgs	Message definitions.
jexo_wrist	Node to publish joint states of the wrist assembly.
joint_velocity_estimator	Nodes to smooth joint velocities by fitting a polynomial.
load_broadcaster	Nodes to publish fixed wrench and wrench from a simulated spring on the end effector.
netft	Driver for the force/torque sensor.
path_node	Data processing for touch sensor.
touch_node	Data processing for touch sensor.

The main packages focused on for redesign are the jexo_control, jexo_core, and netft packages.

4 Implementation

This section outlines the decisions made and details behind the implementation of the exoskeleton control software. Implementation involved deciding how to structure the ROS packages, integrating ROS Control with the control system, using Gazebo for the simulation, modelling the physical parameters of the robot, and modelling the robot in MATLAB.

Programming the control system was the largest part of this project, and involved creating Robot Hardware Interfaces for the real robot, the force/torque sensor, and the simulated force/torque sensor. Using Gazebo meant that Gazebo provided the hardware interface for the simulated robot. Controllers were created to link with these hardware interfaces. The joint controllers link to the exoskeleton hardware interfaces, and the task controllers link to the force/torque sensor interfaces.

4.1 Package Structure

The ROS packages were organised according to best practice principles: that each package should have a single purpose, a package should use the minimum number of dependencies, and a package should be able to be rewritten without breaking other packages (Robotics Back-End 2019). Packages are prefixed with "jexo" to avoid any naming conflicts with existing packages. Table 4.1 outlines the package structure and the purpose of each package.

All packages are contained within a "jexo" metapackage, which also serves as the git repository. This means that to install the JEXO packages, the steps are simply to clone the repository in the "src" directory of the catkin_workspace, install dependencies via "`rosdep install jexo`", and then build.

Table 4.1: Summary of ROS packages implemented

Package	**Summary**
ft_simulator	Hardware interface to simulate a force/torque sensor. Provides a joystick-controlled hand and simulates input to the force/torque sensor to test admittance control without requiring access to the physical robot.
jexo_bringup	Launch files to initialise the robot and switch controller groups.
jexo_core	Hardware interface for JEXO.
jexo_description	URDF, SRDF, and mesh files for the robot model.
jexo_gazebo	Gazebo configuration and launch files.

jexo_joint_control	ROS controllers that use the JEXO hardware interface to send joint commands and receive joint states.
jexo_kinematics	Computations for forward kinematics, kinematic Jacobian, swivel angle, kinematic constraint, and augmented Jacobian.
jexo_msgs (unchanged)	Message definitions.
jexo_task_control	ROS controllers that use a force/torque hardware interface to publish task space commands.
joint_velocity_estimator (unchanged)	Nodes to smooth joint velocities by fitting a polynomial.
netft	Hardware interface for the force/torque sensor.
ps4_teleop	Definitions to map joystick inputs to the joy ROS message.

4.2 Kinematics

The jexo_kinematics package is designed to assist with the calculations related to the exoskeleton's kinematics. It uses the Orocos Kinematics and Dynamics Library (KDL) to read the robot's description and calculate the forward kinematics and Jacobian, and uses the Eigen library for linear algebra computations.

Originally the jexo_kinematics package consisted of two executables, one to publish the forward kinematics and one to publish the Jacobian. However, this created a large potential latency in the use of the forward kinematics and Jacobian, as it would use the last available joint_states message to compute these variables, and when the forward kinematics or Jacobian were used, the control loop would also use the last available message. This means that if the publishers were running at 60Hz, the Jacobian and forward kinematics could potentially be up to 30ms behind the current joint state data.

Because of this, the package was changed to instead be a library that provides the KinematicModel and KinematicConstraintSolver classes. Methods in these classes can be called from inside the control loop, so that the controller can use the most recent data possible.

The KinematicModel class provides KDL wrappers to compute the forward kinematics and Jacobian directly from a joint_states message.

4.2.1 Optimal Swivel Angle Calculation

The KinematicModel class also provides a method to compute the swivel error. The optimal swivel angle is computed via the method proposed by Kim et al (2011), which maximises movement of the hand to the mouth. This was done by first defining the swivel angle reference frame, starting with the unit vector n pointing from the wrist to the shoulder:

$$n = \frac{P_w - P_s}{|P_w - P_s|}$$

The vector u was chosen to point downwards, orthogonal to n, as it is intuitive to have zero swivel angle when the elbow is at its lowest position. This was computed by setting a downward vector $a = [0 \quad 0 \quad -1]$. The vector u is:

$$u = \frac{a - a \cdot n * n}{|a - a \cdot n * n|}$$

The final vector completing the swivel reference frame is calculated with the cross product, $v = n \times u$.

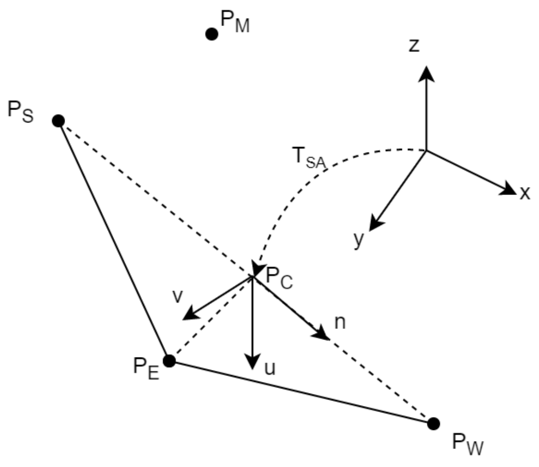

Figure 8: Swivel angle reference frame

The current swivel angle ϕ is calculated by transforming the position of the elbow P_E into the swivel angle reference frame, which is the same as the original code.

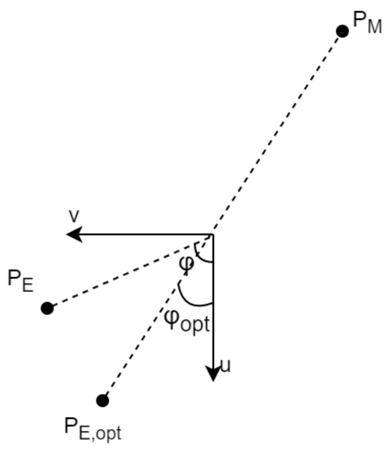

Figure 9: Current and optimal swivel angle in the swivel angle reference frame

The optimal swivel angle ϕ_{opt} is calculated by defining a vector f pointing from the wrist to the mouth. The optimal swivel angle can then be calculated by:

$$f' = f - f \cdot n * n$$
$$\phi_{opt} = -atan2(n \cdot (f' \times u), f' \cdot u)$$

Finally, the swivel error is given by:

$$\phi_e = \phi_{opt} - \phi$$

4.2.2 Joint Velocity Calculation

The KinematicConstraintSolver class provides a method to calculate the joint velocity given the current joint states and the desired task space velocity, using the same task space augmentation method as the previous control system, outlined in Section 3.2.4.

4.3 ROS Control

ROS Control is a set of ROS packages designed to decouple controllers from the hardware of the robot. This decoupling from the hardware was key in implementing the robot simulation, as it enabled the same controllers to be used for both the real robot and simulation. The below flow diagram provides an overview of the architecture and flow of data in ROS Control.

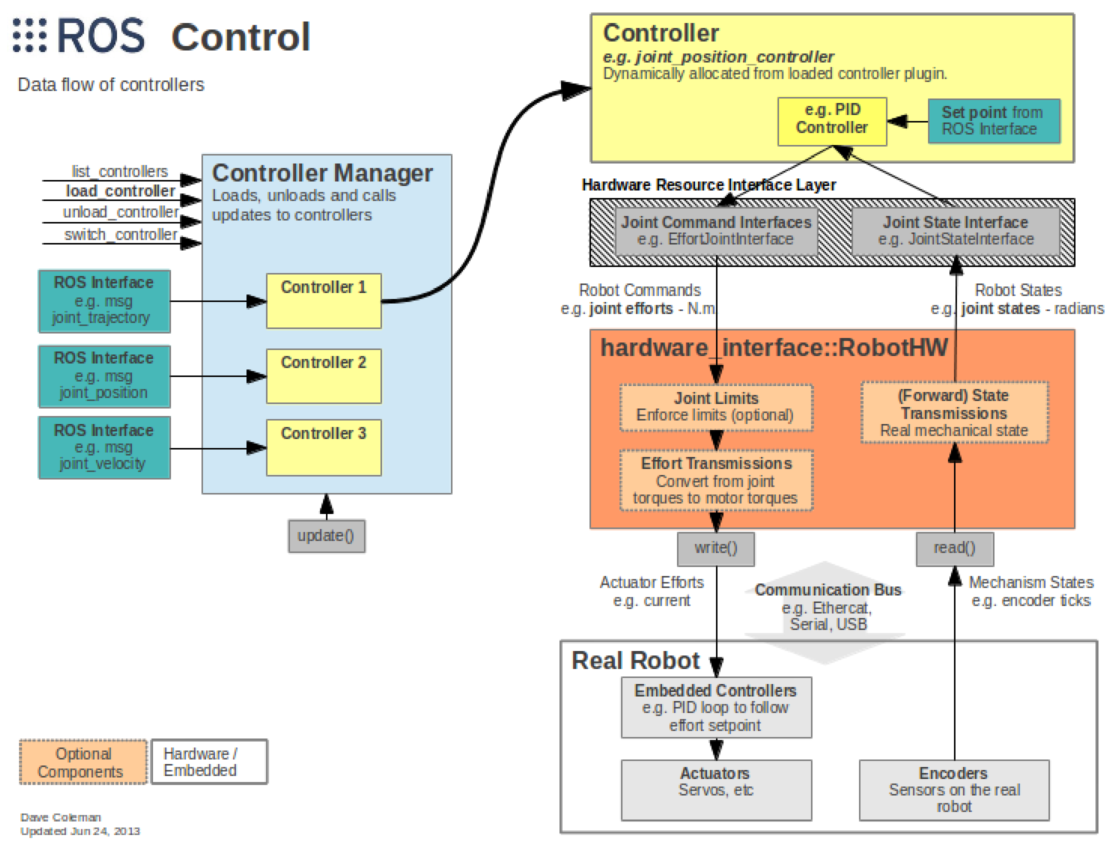
ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
Effort Transmissions Convert from joint torques to motor torques
(Forward) State Transmissions Real mechanical state
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 24, 2013

Figure 10: ROS Control Flow Diagram (Source: ROS 2014)

At a high level, a hardware interface is created for the robot that provides the controllers access to standardised interfaces. For example, the “JointStateInterface” will give access to the robot’s current joint states, and the “EffortJointInterface” will allow the controller to send effort commands.

The controllers run in a real time loop, using the hardware interface provided to respond to inputs and control the robot. The controller can be specified using standardised interfaces, meaning controllers can be made generic to any robot that provides the same interface.

Controllers are run through the controller manager. The controller manager can also load, unload, and switch controllers, meaning different controllers can be used without needing to stop and start a separate program.

4.3.1 Advantages

ROS Control was used for this project because it provides three main advantages. Firstly, ROS controllers run in a real time loop, which reduces latency and increases the responsiveness of the robot.

Secondly, use of ROS control improves modularity and reusability through the set of standardised interfaces. The capability for a controller to be used on any robot that provides a compatible hardware interface means that the same controllers can easily be used on both the real hardware and in simulation.

Thirdly, implementing the hardware interface enables switching controllers during runtime. For example, the robot could first be jogged to a comfortable starting position, then the operator could grasp the handle and switch to the admittance controller when they are ready.

4.4 JEXO Hardware Interface

The JEXO Hardware Interface is located in the jexo_core package. Originally, the robot control software used the “jaco” node to receive joint commands through messages sent to the /ref_vel topic, and interface with the API to forward these commands to the robot and query the robot’s joint states. This was switched this to a ROS Control Robot Hardware Interface for the advantages outlined in the section above.

The robot hardware interface implemented for JEXO provides three standardised interfaces, the JointStateInterface which provides data from the robot’s joints, the VelocityJointInterface which is used to command joint velocity, and the PositionJointInterface which enables commanding joint position. The Kinova Jaco API is used to communicate with the robot’s embedded controllers.

The read() function uses the GetAngularPosition API call to retrieve the joint positions. The joint velocities are then calculated by dividing the change in position by the time between position readings. The write() function uses the SendBasicTrajectory API call to command the joint velocity or position. The hardware interface determines whether to command joint velocity or position through the doSwitch() function, which checks which controllers are loaded and sets an internal variable based on whether they are position or velocity controllers.

In the main loop, the hardware interface is created, and a ControllerManager is initialised with the hardware interface to provide controllers with access to the hardware interface. A separate AsyncSpinner thread is created to ensure callbacks are handled. The node then enters a loop that reads the joint positions, updates the controller manager, and commands joint velocities.

The main limiting factor in the loop rate of the main loop is the data rate in communicating with the robot. Kinova Support verified that the Gen 1 and Gen 2 Jaco arms use the same

API, so a separate hardware interface using the newest API version was implemented and tested to determine if there were any speed improvements between the two versions. Two tests were run with each hardware interface, the first calling the getAngularPosition method in the read() function and the second calling both the getAngularPosition and getAngularForce methods. The main loop was run for 1 minute continuously in each case, and the average loop frequency was calculated after this time. Unfortunately, there was no significant difference in speed between new and old versions of API, with the new version being marginally slower in both cases. The API version used was therefore left unchanged in this project.

With no difference between the two API versions, this means that the CAN bus data rate is the most likely factor limiting the loop rate. Improving this would require modifying the firmware of the robot to optimise the data communicated, as each packet contains data pertaining to the Jaco which is unnecessary for JEXO, such as finger position.

Table 4.2: Hardware Interface loop rate API comparison

API Version	**Mean loop rate with effort**	**Mean loop rate without effort**
6.1.0 (Gen 2)	140.7 Hz	188.0 Hz
5.0.1 (Gen 1)	140.8 Hz	188.3 Hz

4.5 Joint Controllers

Joint controllers are located in the jexo_joint_control package. The joint controllers take commands from the ROS interface as input, and communicate with the JEXO hardware interface to receive joint data and execute joint commands. The joint controllers implement the controller interfaces corresponding to the standardised interfaces in the JEXO hardware interface.

Controllers make use of two features of the Realtime Tools ROS Package, RealtimeBuffer and RealtimePublisher. RealtimeBuffer is used in callbacks and operates similar to a mutex, with the key difference being that writing to the buffer is done in non-real time, and reading from the buffer is done in real time. This means that the update() loop of the controller can function in real time, whereas if a mutex were used, the update() loop would be blocked when the callback was writing to the variable. RealtimePublisher is a realtime safe wrapper for the standard ROS Publisher, and allows publishing ROS messages from within the update loop.

4.5.1 Joint Velocity Controller

The JointVelocityController implements the VelocityJointInterface to give direct control of the velocity of each joint of the robot. The controller subscribes to the /joy topic so that a joystick can be used to control the robot's joints. In the update loop, it forwards the joystick commands for each joint to the Controller Manager, which then communicates with the hardware interface to move the robot.

4.5.2 Cartesian Controller

The CartesianController also implements the VelocityJointInterface. It subscribes to the /jexo/task_cmd topic to get commands in the cartesian task space, and /jexo/joint_states to receive the most recent joint information. In the update loop, it calculates the velocity to command each joint of the robot, using the methods provided in the jexo_kinematics package.

4.5.3 Joint Position Controller

The JointPosition controller implements the PositionJointInterface. It subscribes to the topic /jexo/joint_cmd, and the command line can be used to publish to this topic. In the update loop, it forwards these position commands to the Controller Manager.

4.5.4 Joint State Controller

The JointStateController is a fork of the Joint State Controller provided by ROS Control. The original controller published joint states to the /joint_states topic, which could not be changed as the remap node argument does not work on ROS Controllers, since they are not run as an individual node but through the Controller Manager. The controller was modified to take the topic name as a controller parameter, so that the controller can be configured to publish to /joint_states_raw, which is then smoothed by the joint_velocity_estimator, which publishes the smoothed joint states to /joint_states.

4.6 Force Torque Sensor Hardware Interface

The NetFTInterface is located in the netft package and provides the ForceTorqueSensorInterface for the NetFT force/torque sensor. When the interface is running, it communicates with the NetFT sensor over ethernet to query the force and torque data. This data is used to update the Controller Manager, which makes the force and torque data available to controllers that implement the ForceTorqueSensorInterface. As with the original netft node, the NetFTInterface also provides a “bias” service, which can be called to reset the force/torque readings to zero.

4.7 Force Torque Sensor Simulator

As the robot is velocity controlled, the position of the robot tends to drift slowly over time. In reality, this error is corrected by the human operator exerting a small opposing force to the direction it is drifting in. This force is fed through the admittance controller and then the joint controller to correct the position of the exoskeleton.

To make the simulation behave closer to reality and to enable testing of admittance controllers, the force torque sensor simulator was created with the ft_simulator package. It works by simulating a human hand, modelling it as a spring-damper system to compute the force exerted on the end effector. This is given by the equation $F = ky + c\dot{y}$, where y is the distance to the end effector and $\dot{y}$ is the difference between the velocity of the hand and the velocity of the end effector.

The velocity of the virtual hand is operated by a joystick. The simulator uses the forward kinematics of the robot to calculate the distance between the position of the hand and the position of the robot’s end effector, then calculates a restoring force to pull the end effector back to the position of the hand. This force is expressed in the reference frame of the end effector so that it can be easily substituted by force from a force torque sensor on the real robot’s end effector. The hand is visualised in RViz using a red marker and the force is visualised using a green arrow, shown in Figure 11.

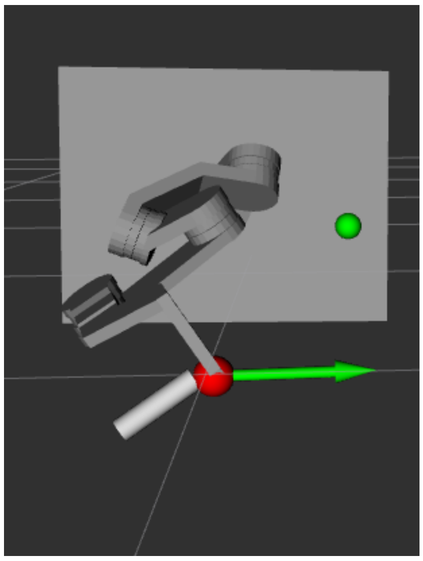

Figure 11: Admittance control using the force/torque sensor simulator

The force torque sensor simulator was implemented using ROS Control, with the simulator providing a ForceTorqueSensorInterface. The task space controllers then implement this interface.

4.8 Task Controllers

The task space controllers are located in the jexo_task_control package, and implement the ForceTorqueSensorInterface. The task space controllers publish commands in the (x, y, z) task space, which are input to the CartesianController to convert to joint space commands.

4.8.1 Admittance Controller

The AdmittanceController commands the task space velocity based on the force measured by the force/torque sensor at the end effector. The task space velocity is computed by modelling the exoskeleton as a mechanical admittance, using the same method outlined in Section 3.2.1. An inertia parameter is passed to the controller, but unfortunately inertia could not be implemented due to time constraints. Once calculated, the task space velocity is transformed from the sensor frame to the global frame, and published to /jexo/task_cmd.

4.8.2 Joystick Controller

The JoystickController subscribes to /joy, and uses joystick input to directly command the task space velocity, publishing the commands to /jexo/task_cmd. Although it is implemented with the ForceTorqueSensorInterface, it does not use the force/torque readings, but was implemented this way so that the Controller Manager could be used to switch between admittance control and joystick control.

4.9 JEXO Model

JEXO is described using the Unified Robot Description Format (URDF), which is an XML format for modelling robots. The URDF is generated with Xacro, which is an XML macro language that makes the URDF easier to understand, modify, and reduces repetition. The key elements of the URDF are <link> and <joint>. The Gazebo simulator requires an additional <transmission> element, which tells Gazebo how this joint will be controlled.

Each link of the robot requires a corresponding link element in the URDF, to describe the physical characteristics of the link. This includes the inertial properties, the visual mesh and material, and the collision mesh. It also includes contact coefficients of friction, stiffness, and damping, which are used to compute collision behaviour in the simulation (ROS 2021).

Solidworks was used to compute the inertial properties and centre of mass for each link, which were then viewed in Gazebo to verify accuracy, shown in Figures 12 and 13. The actuators used have a significant weight, so they were modelled in the URDF as cylinders fixed at each joint. Actuator inertia was calculated as the inertia for a solid cylinder of the same size and weight. Details of these properties for each link and the actuators can be found in Appendix 3.

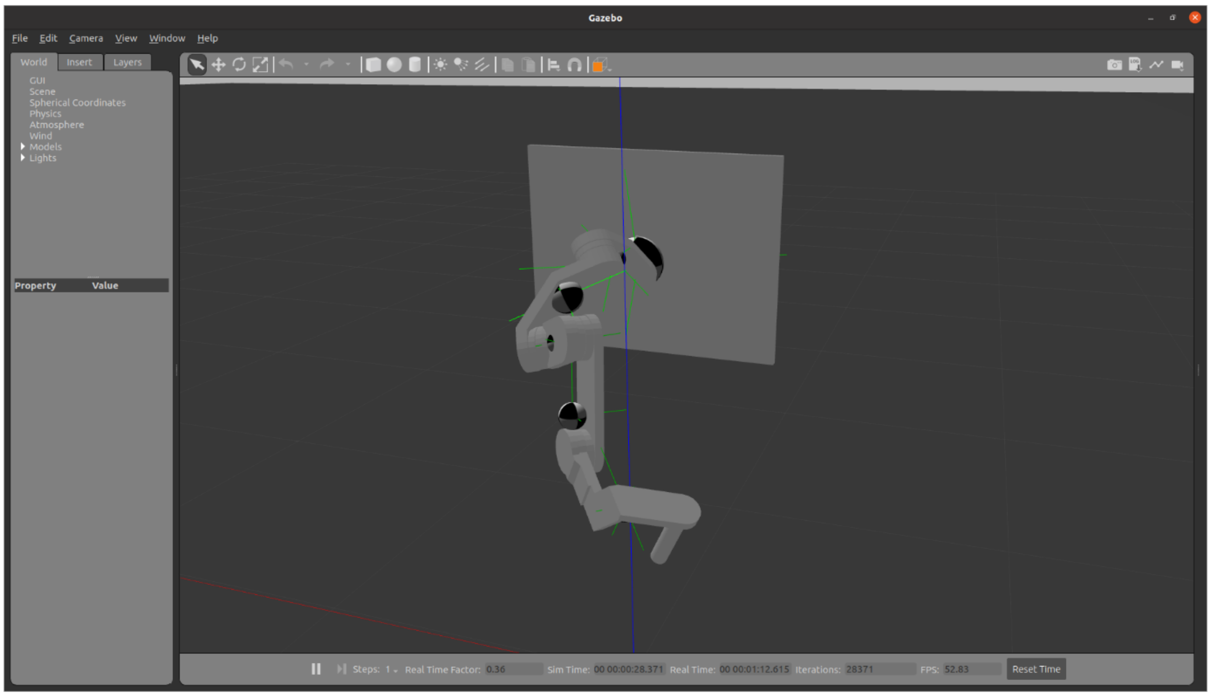

Figure 12: Link centres of mass

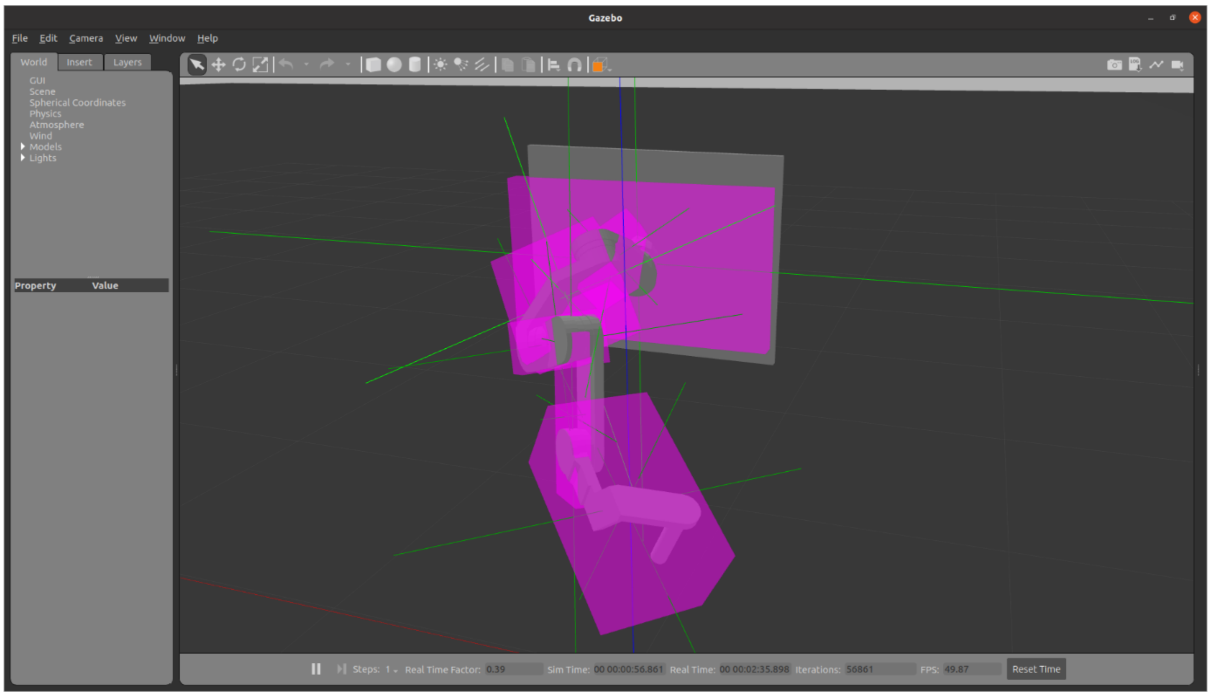
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.39 Sim Time: 00 00:00:56.861 Real Time: 00 00:02:35.898 Iterations: 56861 FPS: 49.87 Reset Time

Figure 13: Link equivalent inertia

Each joint of the robot is described in the joint tag of the URDF. Joints can be fixed, revolute, or prismatic. JEXO has a fixed joint connecting the first link to the world, revolute joints connecting links, fixed joints connecting the actuator masses to the links, and fixed joints connecting the handle and force/torque sensor to the last link. For each joint, joint limits of position, effort, and velocity are specified. Effort and velocity limits are specified according to the actuator specification sheet, at 26 Nm of maximum torque and 0.8 rad/s velocity. Dynamic properties of viscous damping and static friction can also be specified for joints, but as these properties are not given by the actuator specification sheet, they are estimated at 1.0 Nms/rad and 0.5 Nm respectively.

4.10 Gazebo Simulation

Gazebo is a robotics simulation tool with integrated physics engines, which enables testing robotics control systems without access to the physical robot. Gazebo has been used for the majority of development in this project, as the physical robot was initially not working and required a control board replacement.

The Gazebo ROS Control plugin provides a simulated hardware interface that can be connected to, instead of the real robot's hardware interface (Figure 14). This makes development simple when using ROS Control, as the same controller can be used for both the simulation and the real robot.

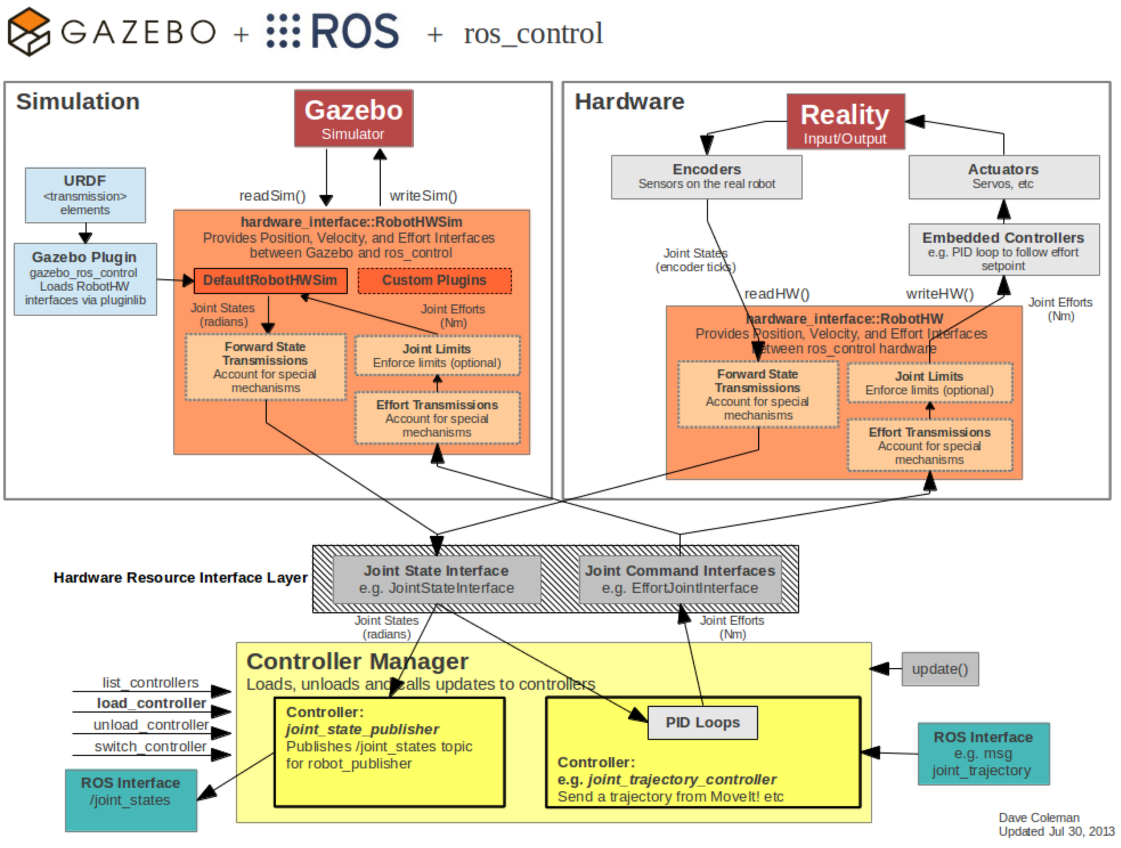
GAZEBO + ROS + ros_control
Simulation
Gazebo Simulator
URDF <transmission> elements
Gazebo Plugin gazebo_ros_control Loads RobotHW interfaces via pluginlib
readSim()
writeSim()
hardware_interface::RobotHWSim Provides Position, Velocity, and Effort Interfaces between Gazebo and ros_control
DefaultRobotHWSim
Custom Plugins
Joint States (radians)
Joint Efforts (Nm)
Forward State Transmissions Account for special mechanisms
Joint Limits Enforce limits (optional)
Effort Transmissions Account for special mechanisms
Hardware
Reality Input/Output
Encoders Sensors on the real robot
Actuators Servos, etc
Embedded Controllers e.g. PID loop to follow effort setpoint
Joint States (encoder ticks)
readHW()
writeHW()
Joint Efforts (Nm)
hardware_interface::RobotHW Provides Position, Velocity, and Effort Interfaces between ros_control hardware
Hardware Resource Interface Layer
Joint State Interface e.g. JointStateInterface
Joint Command Interfaces e.g. EffortJointInterface
Controller Manager
Loads, unloads and calls updates to controllers
list_controllers
load_controller
unload_controller
switch_controller
ROS Interface /joint_states
Controller: joint_state_publisher Publishes /joint_states topic for robot_publisher
PID Loops
Controller: e.g. joint_trajectory_controller Send a trajectory from MoveIt! etc
update()
ROS Interface e.g. msg joint_trajectory
Dave Coleman Updated Jul 30, 2013

Figure 14: Gazebo ROS Control integration diagram (Source: Open Source Robotics Foundation 2014)

To use Gazebo ROS Control, the <transmission> element must be added to the robot's URDF file to specify the transmission type and hardware interface provided by each joint. Unfortunately, a limitation of Gazebo is that it cannot provide both a Position and Velocity joint interface for the same joint. To work around this, the interface type was parameterised in Xacro so that the interface type can be specified in the launch file. This means that either a Position or Velocity joint interface can be used as required, but both interfaces cannot be provided at the same time.

When the joint controllers for the Gazebo simulation are loaded, Gazebo throws a soft error of "No p gain specified for pid". These PID gains are used by Gazebo to compute an effort to control the joints. When the PID gains are not provided, Gazebo instead directly controls the velocity or position of the joint, depending on the interface. The PID gains have not been included as the real robot has embedded joint controllers that accurately control the position and velocity of the joints, so it is not required to simulate these controllers if they can be assumed to be accurate.

4.11 MATLAB Model

A model of the robot was created in MATLAB using Peter Corke's Robotics Toolbox. The use of MATLAB enabled testing and visualisation of control algorithms before implementation in C++. The MATLAB model is defined by the DH parameters of the robot, which are outlined in Table 4.3. Figures 15 and 16 show a comparison between the wireframe model using the DH parameters, and the model with the mesh applied. Figure 15 makes it visually apparent that the axes of rotation of all joints in the shoulder intersect at the same point, which isn't immediately obvious looking at the mesh model of the robot in Figure 16.

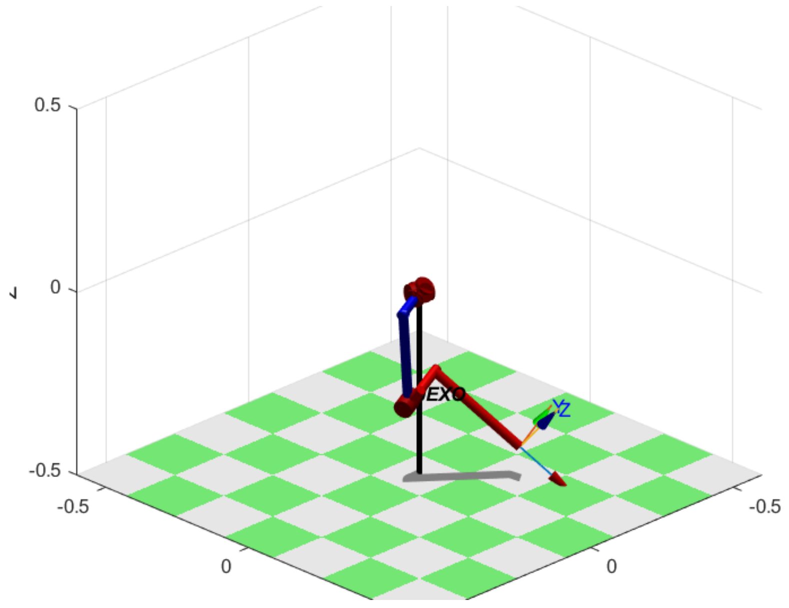

Figure 15: Wireframe MATLAB model

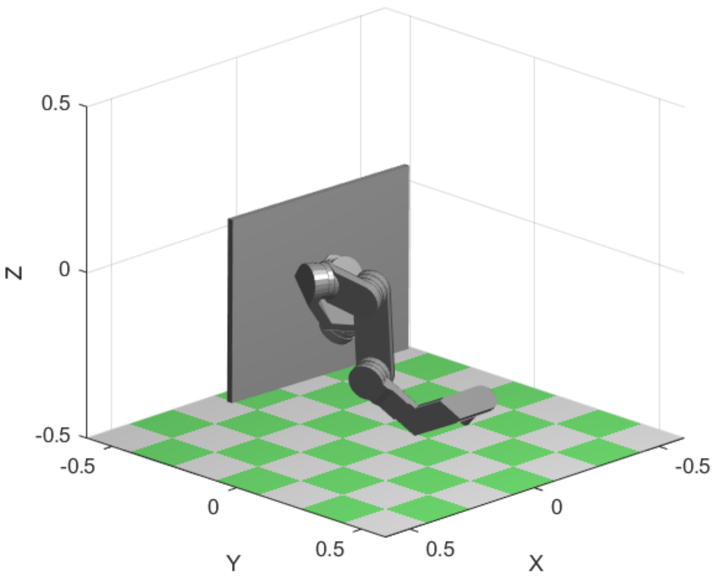

Figure 16: MATLAB model with mesh applied

Table 4.3: JEXO DH Parameters (Carmichael & Liu n.d.)

Joint	**d (m)**	**a (m)**	**alpha (rad)**
1	0	0	0.661
2	0	0	1.3055
3	0	0	0.6702
4	-0.095	0.275	3.1416
5	-0.095	0.283	-1.5708

The MATLAB model was used to verify the swivel angle calculation before implementing in C++, shown in Figure 17. MATLAB enabled visualisation of the calculation steps, which helped with understanding the reasoning behind each step and checking that the result was correct.

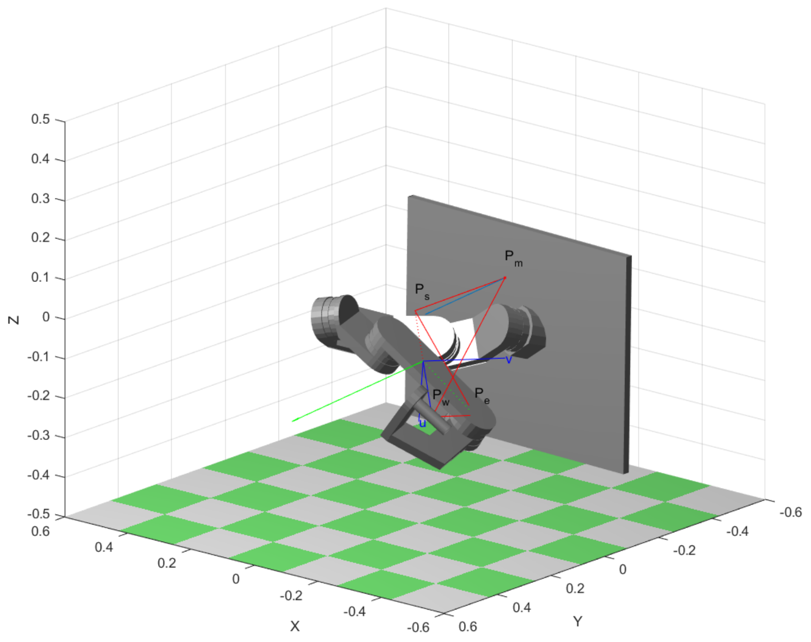

Figure 17: Swivel angle optimisation visualised in MATLAB

5 Results

5.1 Controller Architecture

Figures 18 and 19 are diagrams of the exoskeleton controller architecture, similar to the ROS Control flow diagram in Figure 10. They show the data flow within the control system, and illustrate where the simulator diverges from the real system at the Hardware Interface level.

Figure 18 shows the joint control system, which contains the joint controllers, the JEXO Hardware Interface, and the Gazebo Simulator Hardware Interface.

Figure 19 shows the task control system, which contains the task controllers, the NetFT Hardware Interface, and the Force Torque Simulator Hardware Interface.

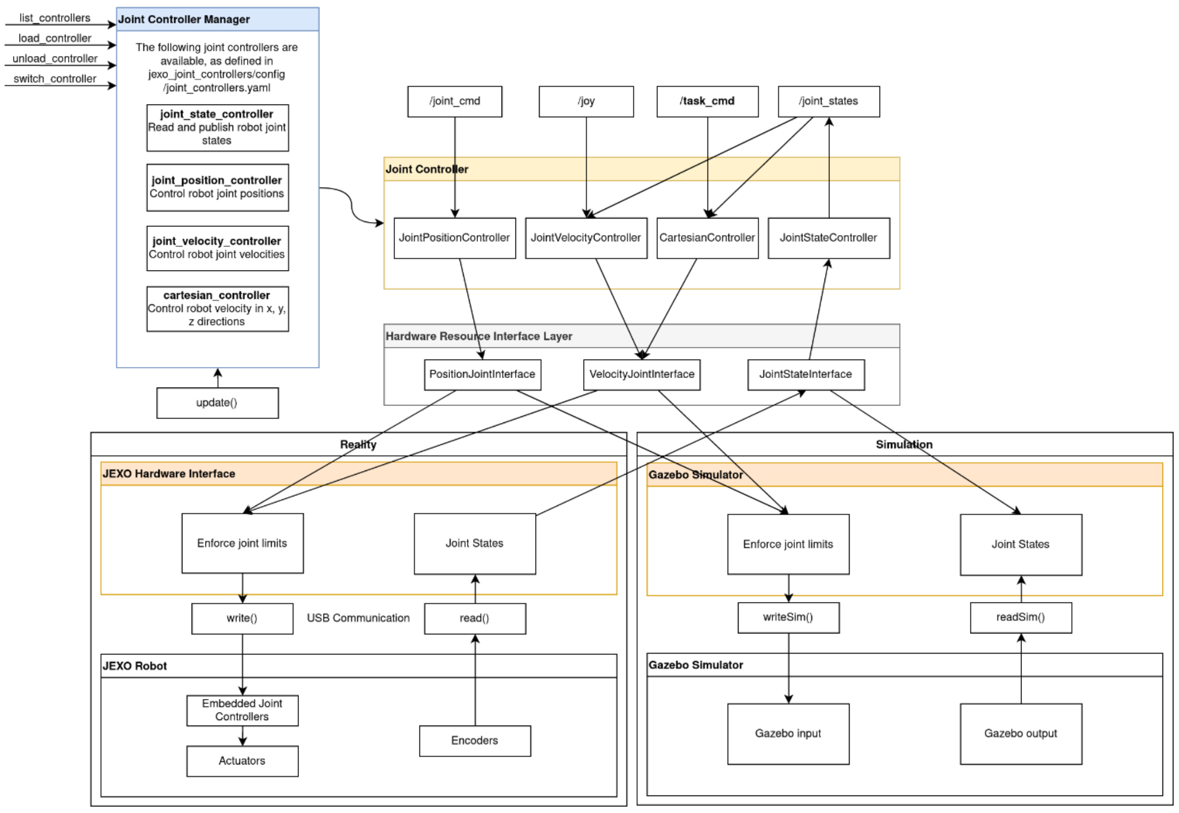

Figure 18: Joint space control system diagram

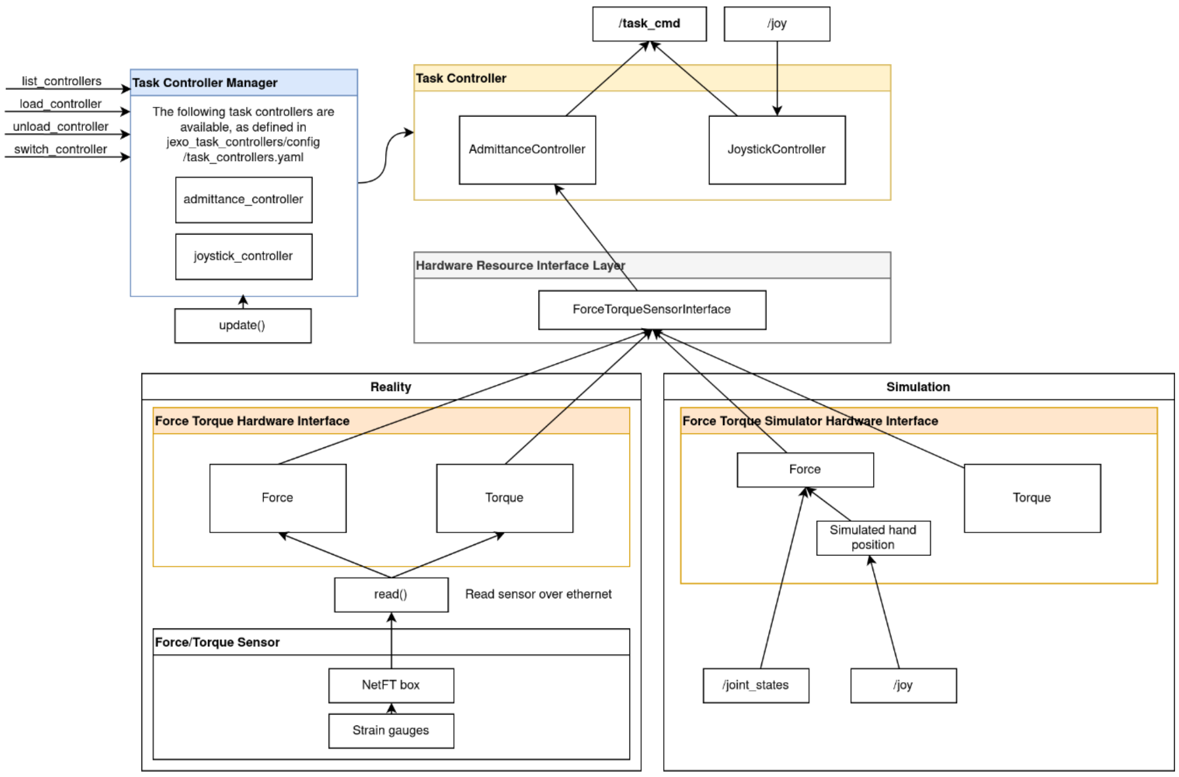

Figure 19: Task space control system diagram

5.2 Demonstration

To start JEXO, the following launch file is used:

```
roslaunch jexo_bringup load_controllers.launch sim:=<false/true>
```

The “sim” argument tells the launch file whether to start the simulation or the real robot. If “sim” is true, it will load the robot in Gazebo and start the force/torque simulator. If it is false, it will start the real robot’s hardware interface and the hardware interface for the real force/torque sensor.

The launch file will then load the available joint and task controllers and their corresponding parameters. All controllers are loaded as stopped, except the joint_states_controller, as joint states are required to visualise the robot in RViz. The launcher will also load the collision detection and joint velocity estimator nodes.

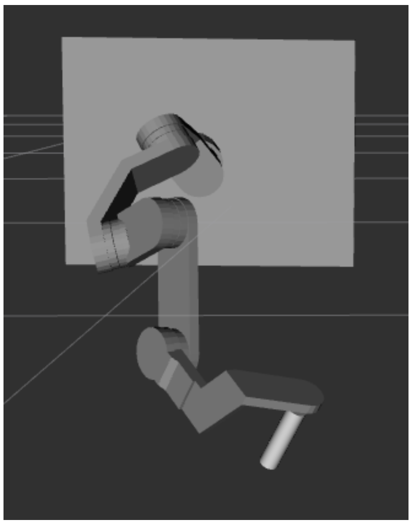

Figure 20: Initial position of the robot after launching load_controllers.launch

After loading the controllers, another launch file can be called to start a set of controllers:

```
roslaunch jexo_bringup switch_controller_group.launch controller:=<controller
configuration>
```

The controller configurations available are:

- admittance: This loads the admittance controller which implements the force/torque sensor hardware interface to publish task space commands. It also loads the cartesian controller, which subscribes to the task space commands and computes the joint velocities to send to the JEXO hardware interface.

- linear_jog: This loads a controller which publishes the joystick input as task space commands. Similar to the admittance configuration, it then controls the joint velocity using the cartesian controller.
- joint_jog: This loads the joint velocity controller, which takes input from the joystick to control the velocity of individual joints.
- joint_position: This loads the joint position controller, which allows directly setting the joint positions by publishing to the joint command topic via the "rostopic pub" command.

To demonstrate, after running the load_controllers.launch file, the "linear_jog" controller group was loaded with:

```
roslaunch jexo_bringup switch_controller_group.launch controller:=linear_jog
```

A green marker appears, showing the position of the mouth used for the swivel angle calculation, and the robot starts swivelling to the optimal angle calculated while maintaining the same end effector position.

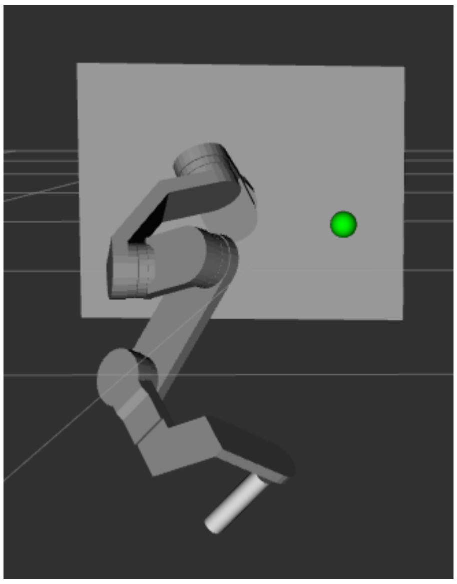

Figure 21: Position of the robot after starting the "linear_jog" controller configuration

The position of the robot is then controllable via a joystick. The figures below show the dexterity of the robot, reaching a wide range of positions while moving in a way that is natural to the human arm.

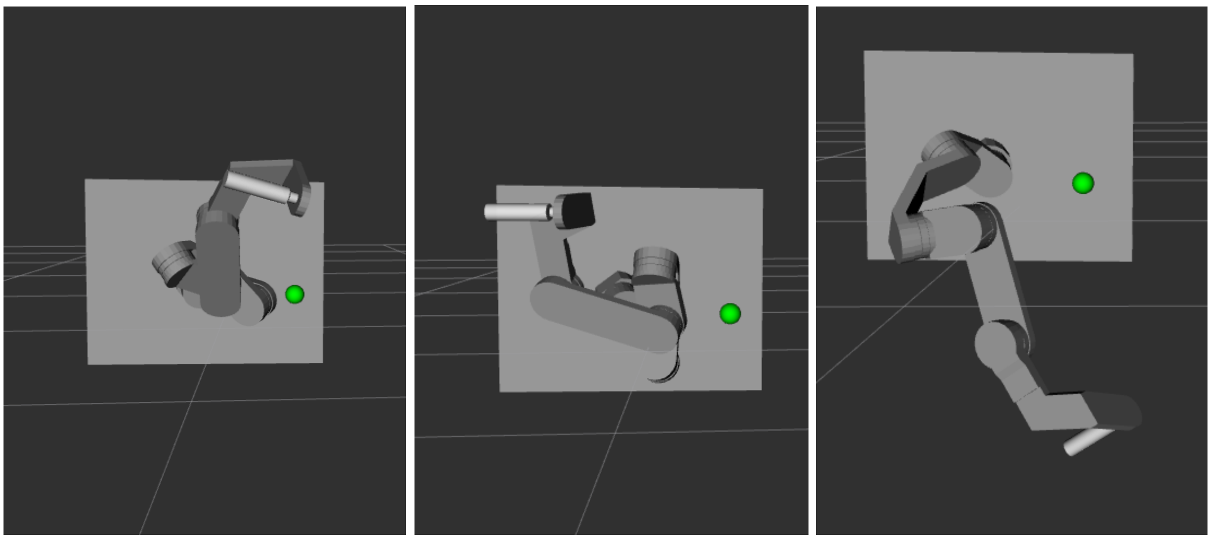

Figure 22: Positions reached in "linear_jog" mode, controlling the robot with a joystick

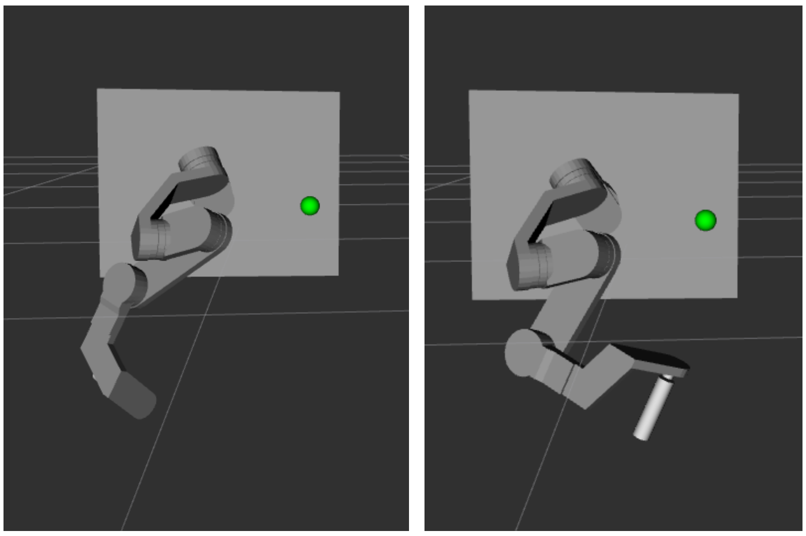

Figure 23: More positions reached in "linear_jog" mode

To switch the controller, the switch_controller_group.launch file simply needs to be called again with the desired controller configuration. In this demonstration, the "admittance" controller group was loaded with:

```
roslaunch jexo_bringup switch_controller_group.launch controller:=admittance
```

The robot is then controlled using the admittance controller, in this case using force computed from the simulated force/torque sensor. The red marker shows the location of the hand, and the green arrow shows the force exerted by the hand on the end effector.

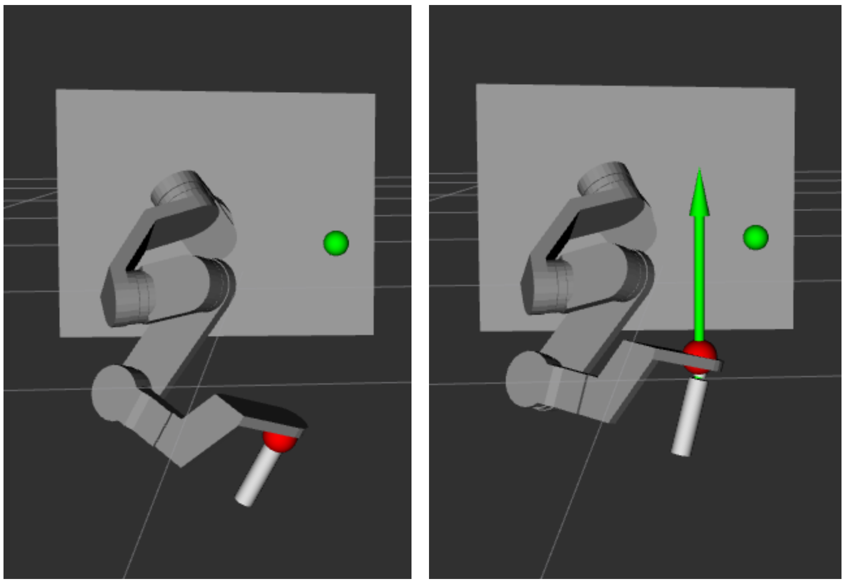

Figure 24: Robot controlled in "admittance" control mode

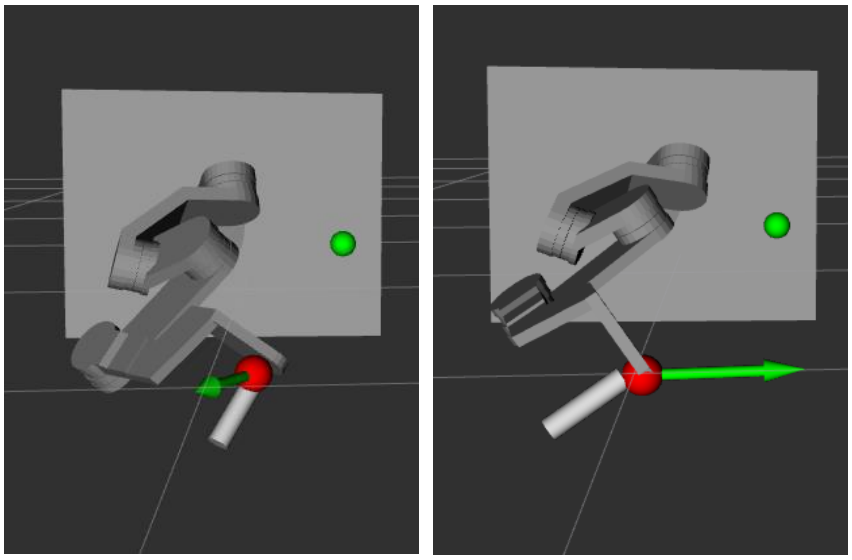

Figure 25: Robot controlled in "admittance" control mode

The switch_controller_group launch file uses the controller group feature of the controller manager (ROS 2019), which allows the user to define sets of controllers that are started and stopped as a group. When a controller group is spawned, the controllers in that group are started and controllers in other groups are stopped.

Controllers can also be switched manually via service calls to /controller_manager/switch_controller, but using the switch_controller_group launch file ensures that controllers are compatible and that there are no conflicts between controllers.

6 Conclusion

In this project, the JEXO exoskeleton control system was updated to use the ROS Control, a modular system that enables robot controllers to be separated from the hardware. Robot Hardware Interfaces were created for the real exoskeleton, the NetFT force/torque sensor, and a simulated force/torque sensor. The exoskeleton's control system was separated into two components, the task space control and the joint space control.

Two task controllers were implemented to publish task commands:

- AdmittanceController takes input from the force torque sensor and models the robot as an admittance to compute desired end effector velocity.
- JoystickController commands the velocity of the end effector directly using joystick input.

Four joint controllers were implemented to interface with the exoskeleton's joints:

- The JointVelocityController uses joystick input to directly set the joint velocity.
- The CartesianController takes task velocity commands, and uses the task space augmentation method to resolve the robot's redundancy and compute the joint velocities.
- The JointPositionController uses the exoskeleton's PositionJointInterface to command the joint positions, with input from the /joint_cmd topic.
- The JointStateController publishes the robot's joint states, which include position and velocity data.

Launch files were created to streamline the process of running the hardware interfaces, loading the controllers, and switching between different controllers.

Gazebo was used to simulate the robot, and the use of ROS Control enabled controlling the Gazebo simulation with the same controllers as the real robot. The simulation allows programming and preliminary testing of control algorithms without requiring access to the physical exoskeleton, which improves productivity when performing final testing on the real robot.

Detailed documentation was created for the codebase to improve its maintainability, improve ease of use, and reduce the time for future developers to understand the control software and add to the codebase.

Overall, this project has achieved its aims set out in Section 1.1, and has added to the functionality of JEXO as a research platform.

6.1 Future Work

JEXO has multiple wrist configurations that were not implemented in this project. Future work could implement these wrist configurations as separate Robot Hardware Interfaces, and combine them using the ROS Control Combined Robot HW class (ROS 2016). This class combines multiple hardware interfaces into a single interface, making controllers see the provided joints of all Robot HWs as belonging to a single robot (Belyaev 2021).

Future work could also be done to improve the accuracy of the simulation, by comparing exoskeleton behaviour in the simulation and reality and modifying parameters so that the simulation better matches reality. Motor parameters of damping and friction defined in the URDF were assumed as there was no data available on these parameters, so experiments could be performed on the motors to characterise the motor damping and inertia and further increase the simulation accuracy.

With respect to the admittance controller, an inertia parameter could be incorporated, and the solution proposed by Dimeas and Aspragathos (2016) of dynamically altering the inertia of the admittance controller to improve stability could be implemented. This would improve the responsiveness of the exoskeleton while mitigating any oscillations. Additionally, if there is future capability for the user to be strapped into the exoskeleton, the admittance controller could also compensate for the weight of the user’s arm due to gravity, which would improve its function as a rehabilitative device.

For the swivel angle control, the method proposed by Kim & Roldan et al. (2012) could be implemented to improve the accuracy of the swivel angle while in motion. The jexo_kinematics package also provides a method to compute the manipulability gradient of the shoulder, which was not implemented into a joint controller due to time constraints. A joint controller could be created using the manipulability gradient with the gradient projection method discussed in Section 2.2.2, and this controller could be compared against the original controller to determine whether it exhibits better singularity avoidance behaviour. If it does, the robot could potentially monitor whether it is close to a singularity, and switch controllers depending on the proximity to a singularity.

References

ATI Industrial Automation 2010, ‘ATI Industrial Automation: F/T Sensor Nano25’, *www.ati-ia.com*, viewed 5 November 2022, <https://www.ati-ia.com/products/ft/ft_models.aspx?id=Nano25>.

Australian Bureau of Statistics 2012, ‘Profiles of Disability, Australia’, *www.abs.gov.au*, viewed 5 November 2022, <https://www.abs.gov.au/ausstats/abs@.nsf/Lookup/4429.0main+features100262009>.

Belyaev, I. 2021, ‘Writing CombinedRobotHW · ros-controls/ros_control Wiki’, *GitHub*, viewed 7 November 2022, <https://github.com/ros-controls/ros_control/wiki/Writing-CombinedRobotHW>.

Carignan, C., Tang, J., Roderick, S. & Naylor, M. 2007, ‘A Configuration-Space Approach to Controlling a Rehabilitation Arm Exoskeleton’, *2007 IEEE 10th International Conference on Rehabilitation Robotics*, viewed 5 November 2022, <https://ieeexplore.ieee.org/document/4428425>.

Carmichael, M. 2014, *Admittance control*.

Carmichael, M. & Liu, D. n.d., *Biomechanical Model-based Optimal Design for a Novel 4-Joint Shoulder Exoskeleton*.

Chiacchio, P., Chiaverini, S., Sciavicco, L. & Siciliano, B. 1991, ‘Closed-Loop Inverse Kinematics Schemes for Constrained Redundant Manipulators with Task Space Augmentation and Task Priority Strategy’, *The International Journal of Robotics Research*, vol. 10, no. 4, pp. 410–25, viewed 5 November 2022, <https://journals.sagepub.com/doi/abs/10.1177/027836499101000409>.

Dimeas, F. & Aspragathos, N. 2016, ‘Online Stability in Human-Robot Cooperation with Admittance Control’, *IEEE Transactions on Haptics*, vol. 9, no. 2, pp. 267–78, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7384497>.

Dubey, R. & Luh, J. 1987, ‘Redundant robot control for higher flexibility’, *IEEE Xplore*, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1087918>.

Dubey, R.V., Euler, J.A. & Babcock, S.M. 1991, ‘Real-time implementation of an optimization scheme for seven-degree-of-freedom redundant manipulators’, *IEEE*

Transactions on Robotics and Automation, vol. 7, no. 5, pp. 579–88, viewed 5 November 2022, <https://ieeexplore.ieee.org/abstract/document/97869>.

Flacco, F., De Luca, A. & Khatib, O. 2015, 'Control of Redundant Robots Under Hard Joint Constraints: Saturation in the Null Space', *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 637–54, viewed 24 May 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7097068>.

Huo, L. & Baron, L. 2008, 'The joint-limits and singularity avoidance in robotic welding', J. Franks (ed.),*Industrial Robot: An International Journal*, vol. 35, no. 5, pp. 456–64.

Jarrasse, N., Proietti, T., Crocher, V., Robertson, J., Sahbani, A., Morel, G. & Roby-Brami, A. 2014, 'Robotic Exoskeletons: A Perspective for the Rehabilitation of Arm Coordination in Stroke Patients', *Frontiers in Human Neuroscience*, vol. 8, viewed 5 November 2022, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4249450/>.

Kang, T., He, J. & Helms Tillery, S.I. 2005, 'Determining natural arm configuration along a reaching trajectory', *Experimental Brain Research*, vol. 167, pp. 352–61, viewed 7 November 2022, <https://link.springer.com/article/10.1007/s00221-005-0039-5>.

Khatib, O. 2019, 'Real-time obstacle avoidance for manipulators and mobile robots', *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, viewed 5 November 2022, <https://ieeexplore.ieee.org/document/1087247/>.

Kim, H., Miller, L.M., Al-Refai, A., Brand, M. & Rosen, J. 2011, 'Redundancy resolution of a human arm for controlling a seven DOF wearable robotic system', *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, pp. 3471–4, viewed 5 November 2022, <https://pubmed.ncbi.nlm.nih.gov/22255087/>.

Kim, H., Miller, L.M., Li, Z., Roldan, J.R. & Rosen, J. 2012, 'Admittance control of an upper limb exoskeleton - Reduction of energy exchange', *IEEE Xplore*, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6347475>.

Kim, H., Roldan, J.R., Li, Z. & Rosen, J. 2012, 'Viscoelastic model for redundancy resolution of the human arm via the swivel angle: Applications for upper limb exoskeleton control', *IEEE Xplore*, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6347476&casa_token=msEZAfO5wf0AAAAA:uNzHetYXpBeqx8s0RHksFHq6WI83ZFfQHh177VX-E7jS06gT0bJ6mSlw0CmNZdEjYhM47x9BJA>.

Kinova n.d., *Joints Specifications*, Kinova Robotics.

Liegeois, A. 1977, ‘Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms’, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 7, no. 12, pp. 868–71, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4309644>.

López-Liria, R., Vega-Ramírez, F.A., Rocamora-Pérez, P., Aguilar-Parra, J.M. & Padilla-Góngora, D. 2016, ‘Comparison of Two Post-Stroke Rehabilitation Programs: A Follow-Up Study among Primary versus Specialized Health Care’, A. Martinuzzi (ed.),*PLOS ONE*, vol. 11, no. 11, p. e0166242, viewed 5 November 2022, <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0166242>.

Nakamura, Y., Hanafusa, H. & Yoshikawa, T. 1987, ‘Task-Priority Based Redundancy Control of Robot Manipulators’, *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, viewed 5 November 2022, <https://journals.sagepub.com/doi/abs/10.1177/027836498700600201>.

Open Source Robotics Foundation 2014, ‘Gazebo: Tutorial: ROS control’, *classic.gazebosim.org*, viewed 6 November 2022, <https://classic.gazebosim.org/tutorials?tut=ros_control>.

Park, J., Chung, W. & Youm, Y. 1999, ‘Computation of Gradient of Manipulability for Kinematically Redundant Manipulators Including Dual Manipulators System’, *Transaction on Control, Automation and Systems Engineering*, vol. 1, no. 1, pp. 8–15.

Ren, L., Mills, J.K. & Sun, D. 2007, ‘Experimental Comparison of Control Approaches on Trajectory Tracking Control of a 3-DOF Parallel Robot’, *IEEE Transactions on Control Systems Technology*, vol. 15, no. 5, pp. 982–8, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4294020>.

Robotics Back-End 2019, ‘Package Organization For a ROS Stack [Best Practices]’, *The Robotics Back-End*, viewed 7 November 2022, <https://roboticsbackend.com/package-organization-for-a-ros-stack-best-practices/>.

ROS 2015, ‘ros_control - ROS Wiki’, *Ros.org*, viewed 7 November 2022, <http://wiki.ros.org/ros_control>.

ROS 2016, ‘combined_robot_hw - ROS Wiki’, *wiki.ros.org*, viewed 7 November 2022, <http://wiki.ros.org/combined_robot_hw>.

ROS 2019, ‘controller_manager - ROS Wiki’, *wiki.ros.org*, viewed 7 November 2022, <http://wiki.ros.org/controller_manager>.

ROS 2021, ‘Adding Physical and Collision Properties to a URDF Model’, *ROS.org*, viewed 7 November 2022, <http://wiki.ros.org/urdf/Tutorials/Adding%20Physical%20and%20Collision%20Properties%20to%20a%20URDF%20Model>.

Siciliano, B. 1990, ‘Kinematic control of redundant robot manipulators: A tutorial’, *Journal of Intelligent and Robotic Systems*, vol. 3, no. 3, pp. 201–12, viewed 5 November 2022, <https://link.springer.com/content/pdf/10.1007/BF00126069.pdf>.

Siciliano, B. 2016, *Springer handbook of robotics*, Cham Springer International Publishing.

Tatlicioglu, E., Braganza, D., Burg, T.C. & Dawson, D.M. 2009, ‘Adaptive control of redundant robot manipulators with sub-task objectives’, *Robotica*, vol. 27, no. 6, pp. 873–81, viewed 5 November 2022, <https://gcris.iyte.edu.tr/bitstream/11147/2469/1/2469.pdf>.

Tatlicioglu, E., McIntyre, M.L., Dawson, D.M. & Walker, I.D. 2008, ‘Adaptive Non-Linear Tracking Control Of Kinematically Redundant Robot Manipulators’, *International Journal of Robotics and Automation*, vol. 23, no. 2, viewed 5 November 2022, <https://www.actapress.com/Abstract.aspx?paperId=33314>.

Vahrenkamp, N., Asfour, T., Metta, G., Sandini, G. & Dillmann, R. 2012, ‘Manipulability analysis’, *IEEE Xplore*, viewed 5 November 2022, <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6651576>.

Van Peppen, R.P., Kwakkel, G., Wood-Dauphinee, S., Hendriks, H.J., Van der Wees, P.J. & Dekker, J. 2004, ‘The impact of physical therapy on functional outcomes after stroke: what’s the evidence?’, *Clinical Rehabilitation*, vol. 18, no. 8, pp. 833–62, viewed 5 November 2022, <https://journals.sagepub.com/doi/abs/10.1191/0269215504cr843oa>.

Wang, C., Pen, L., Hou, Z.-G., Li, J. & Luo, L. 2019, ‘Kinematic Redundancy Analysis during Goal-Directed Motion for Trajectory Planning of an Upper-Limb Exoskeleton Robot’, *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5251–5, viewed 7 November 2022, <https://ieeexplore.ieee.org/document/8857716>.

Yoshikawa, T. 1985, ‘Manipulability of Robotic Mechanisms’, *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, viewed 5 November 2022, <https://journals.sagepub.com/doi/abs/10.1191/0269215504cr843oa>.

Yu, W., Rosen, J. & Li, X. 2011, ‘PID admittance control for an upper limb exoskeleton’, *IEEE Xplore*, viewed 5 November 2022, <https://ieeexplore.ieee.org/abstract/document/5991147>

Appendices

Appendix 1: Kinova Support email

From: "Kinova Support"

Sent: 2022-09-06 11:22:10

To: Luke.Eyles@student.uts.edu.au

Subject: RE: Fwd: Backwards compatibility of Jaco API

Hello Luke,

The GEN1 and GEN2 Jaco use the same API, however, it is possible that certain functions won't work on your original JACO if your firmware version is not up to date.

Of course, some features simply won't be available on your harware.

The GEN3 API is simply not compatible with it.

Sincerely

Éric Linglet

Customer Support

Appendix 2: Actuator Specifications

Kinova actuator specifications (Source: Kinova n.d.)

Specification	**Value**
Diameter	74.5 mm
Height	67 mm
Input/output fasteners	M4x5mm screws
Weight	0.64 kg
Absolute position accuracy	+/- 0.5 degree
Relative position accuracy	+/- 0.05 degree
Motor type	24V Brushless DC
Maximum velocity	8 RPM
Torque (continuous)	15 Nm
Torque (peak)	26 Nm

Appendix 3: Exoskeleton Component Properties

Mass, inertia, and centre of mass for exoskeleton components used in URDF

		Inertia matrix diagonals (kg m^2)			**Centre of mass (m)**		
Link	**Mass (kg)**	**Ixx**	**Iyy**	**Izz**	**x**	**y**	**z**
Link 0	16.085	0.2489	0.9602	0.7191	0	0	0.32739
Link 1	1.425	0.009816	0.01068	0.002731	0	-0.08439	0.23581
Link 2	3.725	0.04883	0.02890	0.02859	0	-0.10732	0.15822
Link 3	1.064	0.004657	0.005665	0.001668	0	-0.05292	
Link 4	1.752	0.002533	0.02548	0.02755	0.185	0	0.14015
Link 5	2.120	0.01495	0.04711	0.04430	0.19483	0.04599	0.1
Actuator	0.640	0.0004614	0.0004614	0.0004440	0	0	0

Appendix 4: GitLab Repository Link

The repository for the codebase created during this project can be found at:

https://code.research.uts.edu.au/12883423/jexo