Samuel Marriage
Constantine College
University of York
May 2026

Bio-inspired Control of a Legged Robot (TQbot)

3rd Year Dissertation Project for BEng in Electronic and Computer Engineering

Final Report

Project Supervisors: Mark Post, Alister Burr

Department of Electronic Engineering
University of York
Heslington
York
YO10 5DD

Abstract

Wheeled platforms dominate mobile robotics due to their simplicity, but struggle in complex environments where legged platforms offer significantly greater manoeuvrability. Most commercial quadrupeds use rigid bodies however, which restricts their operational degrees-of-freedom (DOFs) that quadrupeds with bio-inspired spines utilise for greater speed, stability, and manoeuvrability. TQbot, a quadruped developed at the University of York, addresses this limitation through a cable-driven tensegrity spine providing all three rotational DOFs. Prior work by Lian [1] established TQbot's mechanical design, kinematic model, and central pattern generators (CPGs) capable of producing coordinated joint trajectories for several gaits. This project addresses gaps left in Lian's work by developing the motor control software required for TQbot to perform a gantry-assisted standing posture. Of six objectives defined at the start of the project, three were fully met, two were partially met, and one was not met. The resulting system delivers a three-layer ROS2 control architecture, safety architecture, numerical kinematics layer, and Python control shell that together enable concurrent commanding of all sixteen motors across TQbot's six RS485 buses. A tensioned spine with concurrent leg movement was validated at Demo-Day, and a gantry-assisted standing posture was validated in later lab work. Ultimately, this work provides an extensible software foundation for future free-standing postures and locomotion gaits to be developed.

Acknowledgements

Firstly, I express my gratitude to both of my project supervisors, Mark Post and Alister Burr, for their guidance and feedback throughout this project. Their input has shaped this dissertation, and without their help many issues faced would have taken many more painstaking weeks of going in circles. I also thank Yunlong Lian, whose thesis work provided the baseline for this project that I was fortunate enough to contribute to. Without his initial work, none of mine would've been possible. I would also like to thank the staff of both the Institute for Safe Autonomy and the school of Physics, Engineering, and Technology at the University of York for providing me with places to work, as well as any other staff who have aided my work along the way.

To my friends, both at university and at home, thank you for providing me with a space where I am comfortable enough to be myself. The laughter and joy you have all brought me is invaluable, you have given me years of my life I will never forget. Thank you all for your constant words of encouragement, and for being there for me as an outlet when times gets stressful; I will forever be grateful for all of you.

Lastly, I would like to thank my family for their unwavering support throughout my studies, your love and care has kept me grounded throughout every step of my academic journey. The reminders of how proud you all are of me has kept me going on the hardest of days, and whether through the weekly phone calls and check-ins, or the in person visits, you have all served as a constant reassurance that no matter how far from home I may be, I will never be on my own. Thank you all for everything.

Table of Contents

Abstract **2**

Acknowledgements **3**

1. Introduction **6**

1.1 Background and Motivation 6

1.2 Project Objectives 6

1.3 Contributions Overview 7

2. Background Literature **8**

2.1 Introduction 8

2.2 Robot Locomotion Concepts: Wheels vs Legs 8

2.2.1 Wheeled Platforms' Limitations 8

2.2.2 Legged Platforms 8

2.2.3 Hybrid Platforms 8

2.3 Quadruped Robot Platforms 9

2.3.1 Current Leading Quadrupeds 9

2.3.2 Limitations in Current Quadrupeds 10

2.4 Bio-Inspired Tensegrity Spine Design 10

2.4.1 Spine Designs: Rigid, Passive, and Active 10

2.4.2 Tensegrity 10

2.5 Actuator Technology and Control 12

2.5.1 Actuator Technology 12

2.5.2 Actuator Control 12

2.6 Kinematics and Posture Control 12

2.7 Central Pattern Generators (CPGs) 13

2.8 Robot Operating System 2 (ROS2) 14

3. Inherited Work **16**

3.1 Inherited Hardware 16

3.1.1 Overview 16

3.1.2 Main Body, Tensegrity Spine, and Front/Rear Body Sections 17

3.1.3 Legs 19

3.2 Inherited Software 20

3.3 Gaps and Limitations 20

3.3.1 Hardware 20

3.3.2 Software 20

4. Contributions **21**

4.1 Motor Communication and Control 21

4.2 Safety Infrastructure 22

4.3 Multi-Bus and Multi-Motor Validation 22
4.4 Leg Connection 23
4.5 Kinematics Layer 23
4.6 Gantry Assisted Standing Posture and PD Gains Tuning 24
4.7 Control Shell 24
4.8 Validation Tests 25
5. Tests and Results 26
5.1 Motor Stack Validation 26
5.2 Kinematics Validation 26
5.3 Tensioned Spine with Leg Movement Demonstration 27
5.4 PD Gain Tuning Results 27
5.5 Gantry Assisted Standing Posture Demonstration 28
6. Planning and Outcomes 30
6.1 Initial Plan 30
6.2 Outcomes Against Objectives 31
6.3 Limitations and Reflections 32
7. Statement of Ethics 33
8. Conclusions 34
9. Future Work 35
10. References 36
Appendix 38
Output of 'help()' Function 38
Motor Stack Test Outputs 41
Kinematics Test Outputs 42

1. Introduction

1.1 Background and Motivation

Currently, wheeled locomotion is the dominant platform in robotics due to the simplicity and reliability it provides. However, traversal of more complex terrains is becoming an increasingly crucial focus when deploying robots into real world applications; a task wheeled robots often struggle with [2] (p. 1). Quadruped robots offer a solution to this, and have attracted significant commercial interest, notably in Boston Dynamics' 'Spot' [4] and Unitree's 'A1' [8]. Whilst these quadrupeds offer increased success in traversal of complex real world terrains when compared to their wheeled counterparts, they employ rigid bodies which hinder potential manoeuvrability, speed, and stability. TQbot, developed by Lian at the University of York [1], addresses this by taking inspiration from millions of years of animalistic evolution and biological structures; implementing a bio-inspired tensegrity spine coordinating with a total of sixteen Unitree A1 motors across fifteen degrees of freedom (DOFs).

1.2 Project Objectives

Initially, TQbot existed as a partially assembled quadruped platform with validated spine control on a gantry test rig [1] (pp. 166–174), a central pattern generator (CPG) capable of generating coordinated joint angle trajectories for various gaits in simulation (pp. 140–166), and mathematics for full kinematics solutions (pp. 86-88). However, TQbot lacked the control software to utilise the forward and inverse kinematics solutions to hold a standing position, or to translate simulated CPG gait trajectories into motor commands. Lian left behind foundational standalone Python modules, though they lacked a real-time control loop, support for sixteen motors sharing six RS485 buses, Robot Operating System (ROS) integration, and an operational safety layer.

This project aimed to bridge these gaps in Lian's work, allowing for TQbot's sixteen motors to be commanded to desired positions across fifteen DOFs, producing a stable standing posture and basic trot gait. Six objectives were defined in the initial report to work towards this aim:

(1) Establish communication with a single motor outside of TQbot
(2) Implement hybrid PD control on a single motor using Lian's control method [1] (Equation 3.1, p.75)
(3) Implement Lian's kinematics model using his Denavit-Hartenberg parameters [1] (Table 3.4, p. 81)
(4) Extend communication and control to all sixteen motors onboard TQbot

(5) Achieve a stable standing posture

(6) Achieve a basic trot gait utilising Lian's simulated CPG outputs [1] (Tables 4.1-4.3, pp. 121-123)

The first three tasks planned to use a Raspberry Pi 4B with ROS2 Jazzy and single motor outside of TQbot, before porting over to the Jetson Orin Nano onboard TQbot when working with all sixteen motors.

1.3 Contributions Overview

My final contributions to TQbot are outlined below:

- A three layer ROS2 multi-motor control architecture consisting of per-motor state, per-bus communication, and ROS2 messaging.
- Validation of multi-motor control architecture by moving multiple motors across multiple buses simultaneously.
- Numerical reimplementation of Lian's kinematics solutions, validated by round trip testing in software.
- A safety infrastructure including emergency stops, motor limits, and motor state validation.
- A Python control shell reducing ROS2's operational complexity into user-friendly prompts.
- Deployment of hybrid PD control on TQbot, allowing for position holding within a measured error margin.
- Validation of hybrid PD control by holding the spine at a neutral flat position whilst commanding the leg joints to various targets.
- Establishment of a gantry assisted standing posture.

2. Background Literature

2.1 Introduction

This literature review is structured to progress from general locomotion to more technical specifics of this project, it is intended to inform the reader about this project's concepts and justify decisions made.

2.2 Robot Locomotion Concepts: Wheels vs Legs

2.2.1 Wheeled Platforms' Limitations

Deciding how a robot should move through space is a fundamental choice to make in mobile robotic design. While many systems opt for wheeled solutions due to their simplicity, they can often perform poorly in unstructured environments. Zhong et al. [2] note that due to legged robots' multiple limbs, multiple DOFs, and discrete motion trajectory, they are more manoeuvrable and adaptable in these environments than their wheeled counterparts (p. 1). This is essential for applications where terrain cannot be prepared in advance, such as disaster response, industrial inspection, and planetary exploration [1] (pp. 2-6).

2.2.2 Legged Platforms

Quadrupeds offer a balance between stability and complexity, Zhong et al. [2] observe that quadrupeds achieve greater load capacity than bipeds, and a larger range of leg motion, less mechanism redundancy, and lower complexity than multi-legged robots (p. 1). Despite these advantages of quadrupeds, they also face control challenges. In a trot gait, the current target gait for TQbot, diagonal pairs of legs move simultaneously, leaving two legs that remain grounded [1] (p. 124). This requires dynamic balance control, as two support points cannot always ensure TQbot's centre of gravity is contained.

2.2.3 Hybrid Platforms

Hybrid approaches combining wheels and legs have also been explored. Notably, Nakajima and Ietomi [3] developed the RT-mover, a leg-wheel hybrid robot capable of switching between wheel mode for smooth terrain and leg mode for obstacles. This is accomplished by lifting the wheels to step over obstacles when they come into contact with convex terrain. While this demonstrates the effectiveness of combining locomotion methods, the RT-Mover uses five axes and requires adaptive gait, resulting in a greater system complexity than its non-hybrid counterparts.

2.3 Quadruped Robot Platforms

2.3.1 Current Leading Quadrupeds

Significant progress is consistently being made in quadruped robotics, with Boston Dynamics producing some of the most well-known and advanced quadrupeds over the past two decades. Boston Dynamics' most recent quadruped project is 'Spot' [4], a widely successful commercially available quadruped robot designed for industrial inspections and data collection.



Figure 2.1, Boston Dynamics' quadruped - 'Spot'. [4]

At a smaller scale, bio-inspired approaches have produced designs backed by years of evolution. Quinn et al. [5] developed the Whegs I hexapod robot, inspired by abstracted locomotion principles from studies of cockroaches. Whegs I uses tri-spoked appendages called 'whegs' that combine the speed of wheels and the mobility of legs, using a single drive motor and passive compliant axles that allow automatic gait adaptation from a tripod walking pattern to a climbing motion when it encounters obstacles. Morrey et al. [6] extended the concept of whegs, developing a miniaturised system capable of travelling over ten body lengths per second at 9cm in length, and climbing obstacles higher than the length of their legs. Abstracting biological principles in this manner can yield effective robots using current technology, hence abstraction being reflected in TQbot's tensegrity spine. Unitree Robotics [7] has also made quadrupeds significantly more accessible to academics, providing capable hardware and open SDKs, both of which TQbot utilises, for a substantially lower cost than Boston Dynamics. The Unitree A1 [8] notably uses the A1 actuator, also found in TQbot [1] (p. 74).

2.3.2 Limitations in Current Quadrupeds

A common limitation present across the platforms reviewed above is the absence of a multi DOF spine. Most quadruped robots use a rigid body to connect the front and rear leg pairs. Spines are typically limited to a single DOF in the pitch plane [1] (Table 2.1, p. 23), constraining potential for spine-leg coordination that biological quadrupeds use for additional speed, stability, and manoeuvrability.

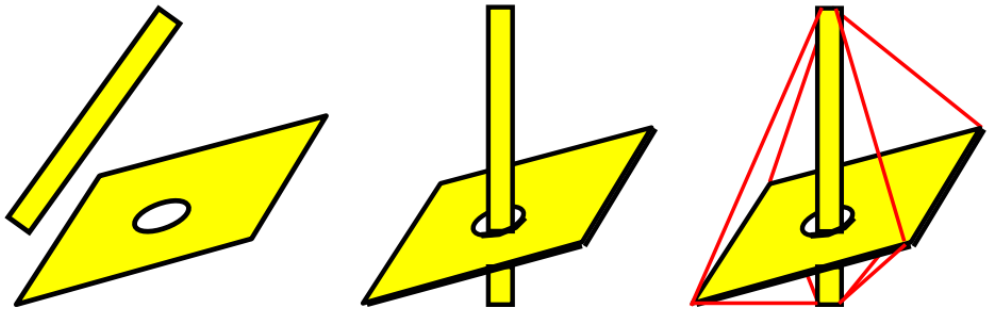
2.4 Bio-Inspired Tensegrity Spine Design

2.4.1 Spine Designs: Rigid, Passive, and Active

Lian [1] (pp. 17-22) surveys existing quadruped robot spine designs and categorises them as rigid, passive, or active. A rigid spine [1] (pp. 17-18) is defined to have no movable joints between the shoulder and the pelvic girdle of the body. It serves as a platform to connect the leg pairs and bear load, with minimal contributions to movement. Rigid spines are low cost, have a simple structure, and are easy to control. Passive spines [1] (pp. 18-20) typically use elastic elements at the waist to connect the body segments. This offers several DOFs, but is not actively controllable and will cause additional disturbance to a quadruped robot that will increase the control difficulty. Active spines [1] (pp. 21-22) provide controllable movement that passive and rigid spines lack, but most are limited to a single rotational DOF. Table 2.1 in Lian's thesis (p.23) reviews fifteen robotic spine implementations, and only the Laika robot provides all three rotational DOFs (roll, pitch, and yaw). TQbot, like Laika, uses a tensegrity structure for its spine [1] (p. 51-59), and is designed to also provide all three rotational DOFs.

2.4.2 Tensegrity

Skelton and de Oliveira [13] provide the foundational engineering analysis of tensegrity systems. They note that tensegrity, a contraction of the words 'tension' and 'integrity' coined by Buckminster Fuller, describes structures composed of rigid compressive parts suspended within tensile parts which stabilise the structure, whereby no rigid components directly rely on each other for support.



(a) Not a Tensegrity Configuration (b) Tensegrity Configuration (c) Tensegrity System

Figure 2.2, Visualisation of a tensegrity system using 2 rigid components. [13]

In robotics, tensegrity structures are a growing area of interest. Sabelhaus et al. developed the SUPERball [14] in 2015, a spherical planetary exploration robot at NASA. SUPERball allows for locomotion by shortening cables to shift centre of mass, and tensegrity design naturally allows for impact absorption when traversing other-planetary terrains. Sabelhaus et al. further developed Laika [15] in 2018, a quadruped at a much smaller scale whose entire spinal section is a tensegrity structure. Laika is capable of lifting each foot individually through spinal cable tensioning without any motors existing directly in the legs.

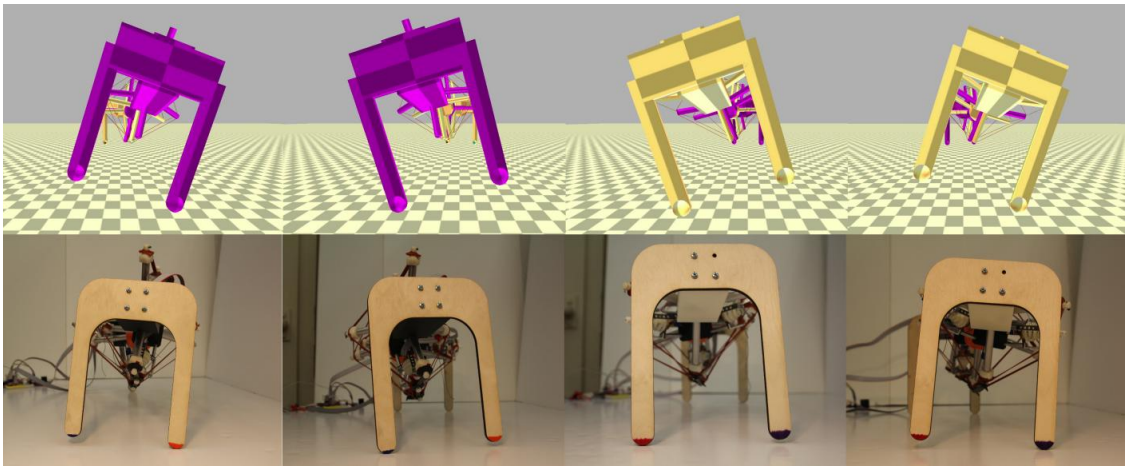


Figure 2.3, Laika lifting each of its feet individually in simulation (top) and hardware (bottom). The feet lifted are: Front Right, Front Left, Back Right, Back Left; When viewed from left to right. [15] (Figure 10)

TQbot's current spine configuration uses a tensegrity vertebra combined with springs and fishing lines, actuated by four motors controlling all three rotational DOFs through cable length adjustment. This can be seen in Figure 3.2.

2.5 Actuator Technology and Control

2.5.1 Actuator Technology

TQbot houses sixteen Unitree A1 actuators across fifteen DOFs [1] (pp. 73-75). The A1 actuator is a permanent magnet synchronous motor (PMSM) with a 9.1:1 reduction ratio, achieving a maximum torque of 33.5 Nm and maximum angular velocity of 21 rad/s [1] (p. 74).

2.5.2 Actuator Control

The hybrid PD control method used is noted in Equation 2.1:

$$\tau = \tau_f + k_p \cdot (q_{des} - q) + k_d \cdot (dq_{des} - dq) \tag{2.1}$$

Where τ is the output torque, τ_f is the feedforward torque, q_{des} and dq_{des} are the desired position and velocity, q and dq are the current position and velocity, and k_p and k_d are the stiffness and damping coefficients respectively. [1] (Equation 3.1, p. 75).

Position control is achieved with high k_p and k_d values while setting τ_f to zero. Conversely, torque control relies on only τ_f. The control loop supports frequencies up to 1000 Hz.

This hybrid approach is well suited to quadruped locomotion, as stiff position control is needed during standing to resist TQbot's weight, while more compliant control is needed when walking. The ability to alternate between modes through parameter adjustment is a significant advantage.

2.6 Kinematics and Posture Control

To make a quadruped stand, calculating the joint angles required to place a foot at a desired location is foundational. This is known as the inverse kinematics problem. TQbot's legs each have three DOFs, and Lian [1] established the Denavit-Hartenberg parameters for the kinematic chain, noted in Table 2.1:

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	-90°	l_1	θ_2
3	l_2	0	0	θ_3
4	l_3	0	0	0

Table 2.1, Denavit-Hartenberg parameters for TQbot's front left leg [1] (Table 3.4, p. 81).

The desired foot positions are typically directly beneath their corresponding hip joints, as this maximises the support area for the centre of mass whilst minimising torque on the joints. The PD

control parameters must be tuned to provide sufficient stiffness to support TQbot's weight while avoiding oscillation or overheating.

2.7 Central Pattern Generators (CPGs)

Central pattern generators (CPGs) are self-organising biological neural circuits found in vertebrate and invertebrate animals that produce rhythmic patterns without requiring rhythmic input [9] (p. 642). In robotics, CPG models are used to generate oscillating signals that drive joint actuators in such a way that produces locomotion gaits.

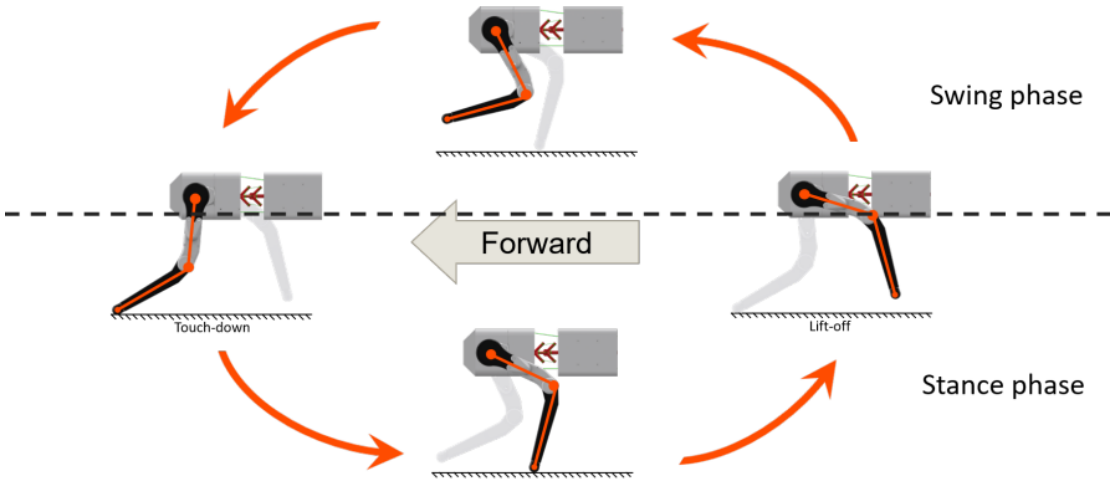


Figure 2.4, A quadruped locomotion gait produced via oscillating CPG signals of the joints. [1] (Figure 4.1, p. 98)

Ijspeert [9] provides a detailed review of CPG applications in robotics, identifying key advantages as being: modulation through a few control parameters, smooth rhythmic motion even when these parameters are drastically changed, and suitability for sensory feedback integration (p. 646).

Lian [1] designed a CPG model for TQbot (pp. 94-134). Each oscillator produces an angle for a joint, described by Lian's Equation 4.11 (p. 99) as follows:

$$\theta_i = x_i + f(r_i)\cos(\phi_i) \tag{2.2}$$

Where x_i is the current offset of the oscillator, $f(r_i)$ is a function to shape oscillator amplitude (defined in equation 4.12 [1] (p. 100)), and ϕ_i is the phase.

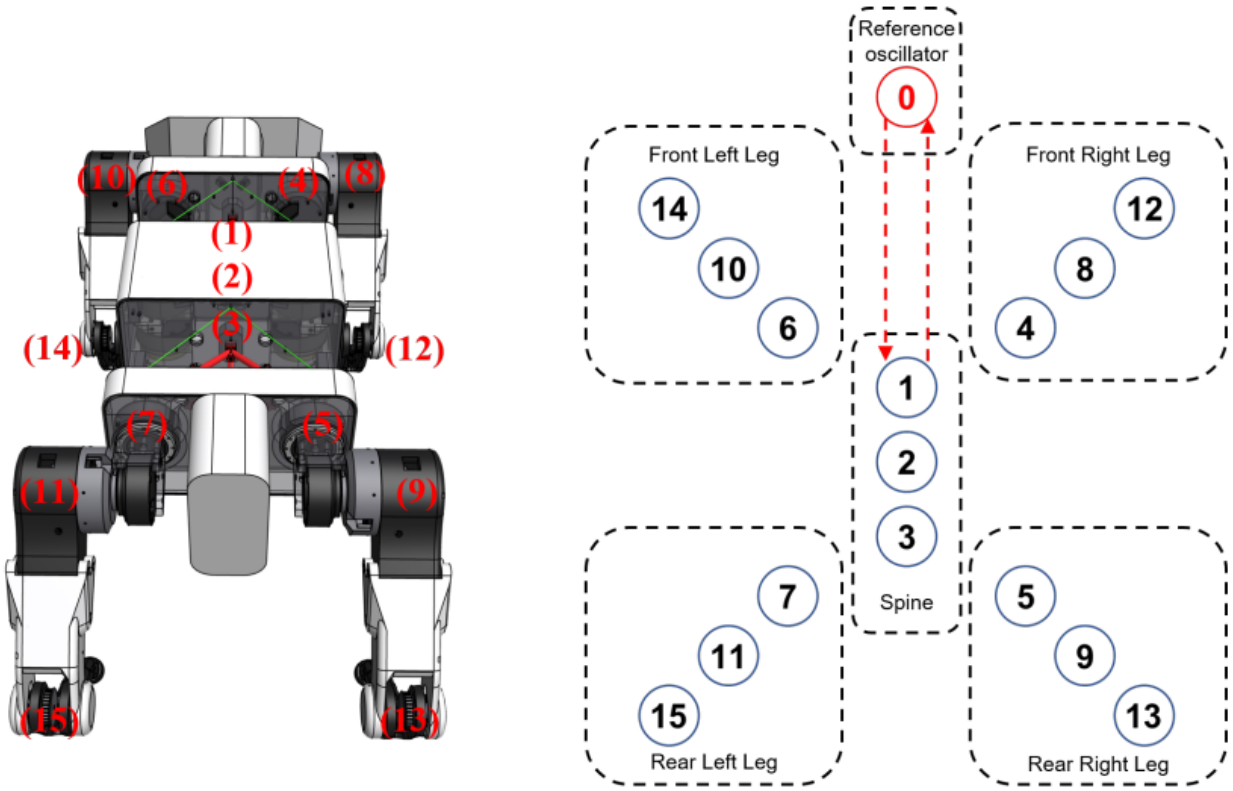


Figure 2.5, TQbot CPG joint correspondence diagram. The left shows TQbot model, and the right shows a CPG map, numerical indicators show where they relate to TQbot's body. [1] (Figure 4.5, p. 105)

Limb coordination is achieved by coupling oscillators through phase lag parameters. Different sets of parameters produce distinct gaits, such as walking, trotting, pacing, bounding, and spot trotting, shown in Tables 4.1-4.3 [1] (pp. 121-123).

While this project does not involve direct CPG development, understanding of general CPG concepts as well as the CPG Lian [1] developed is crucial, as the motor control layer must receive the values Lian's CPG produces and send them to the actuators at the correct frequency.

2.8 Robot Operating System 2 (ROS2)

ROS2 provides a communication layer between the operating system and user applications, enabling inter-process communication and data exchange between modules. Macenski et al. [10] detail the design of ROS2, stating that it is based on the Data Distribution Service (DDS), a standard for reliable real-time data distribution using a publish/subscribe paradigm. This architecture enables systems to be broken down into modular subsystems that can be developed and tested independently. This modularity is important for TQbot's development, as it means the motor control software can be tested independently of Lian's CPG work [1]. ROS2 Jazzy [11] has been used for initial motor development, and ROS2 Humble [12] has been used for TQbot development.

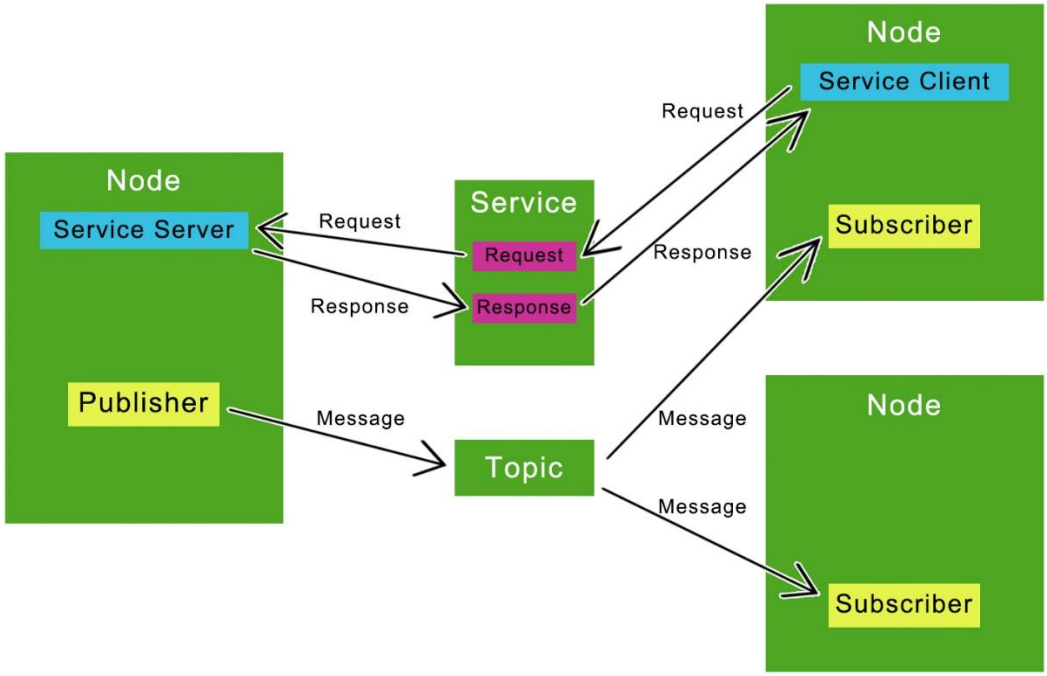


Figure 2.6, ROS2 architecture diagram demonstrating flow of communication.

3. Inherited Work

3.1 Inherited Hardware

3.1.1 Overview

TQbot's entire final shell was left behind from Lian's prior work [1] (pp. 46-77). The entire quadruped weighs approximately 30kg and is comprised of a main body, two tensegrity spinal sections, and front/rear body sections, as well as four legs; two designed for the right and two for the left. In total, TQbot is designed to house an onboard Jetson Orin Nano, two USB expansion boards, a bulk convertor, six USB-to-RS485 convertors, sixteen Unitree motors operating across fifteen DOFs, two IMUs, and three battery modules.

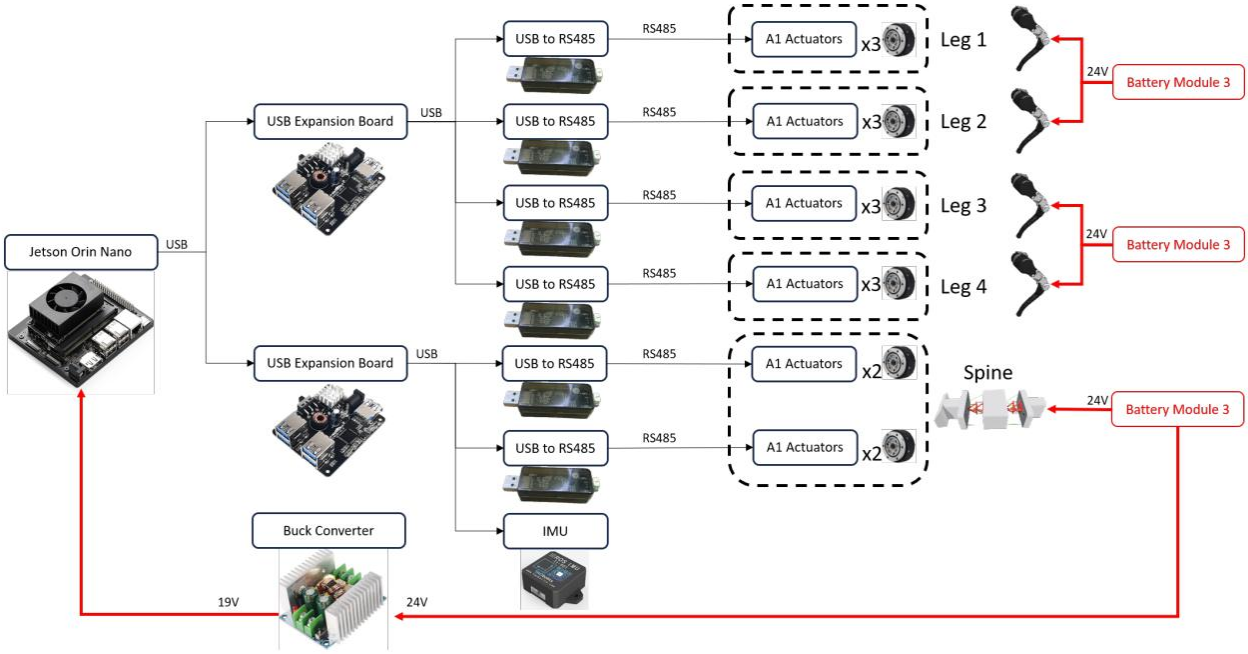


Figure 3.1, TQbot's electronic hardware architecture. Red lines represent power, and black lines represent communication. [1] (Figure 3.28, p. 74)

3.1.2 Main Body, Tensegrity Spine, and Front/Rear Body Sections

TQbot's main body holds its onboard computer, a Jetson Orin Nano, which sends motor commands through two USB expansion boards connected to six USB-to-RS485 connectors (Figure 3.1). It also houses four of the sixteen Unitree A1 motors, which drive the active component of the tensegrity spine.

TQbot's spine is categorised in two distinct passive and active sections, as shown in Figure 3.2.

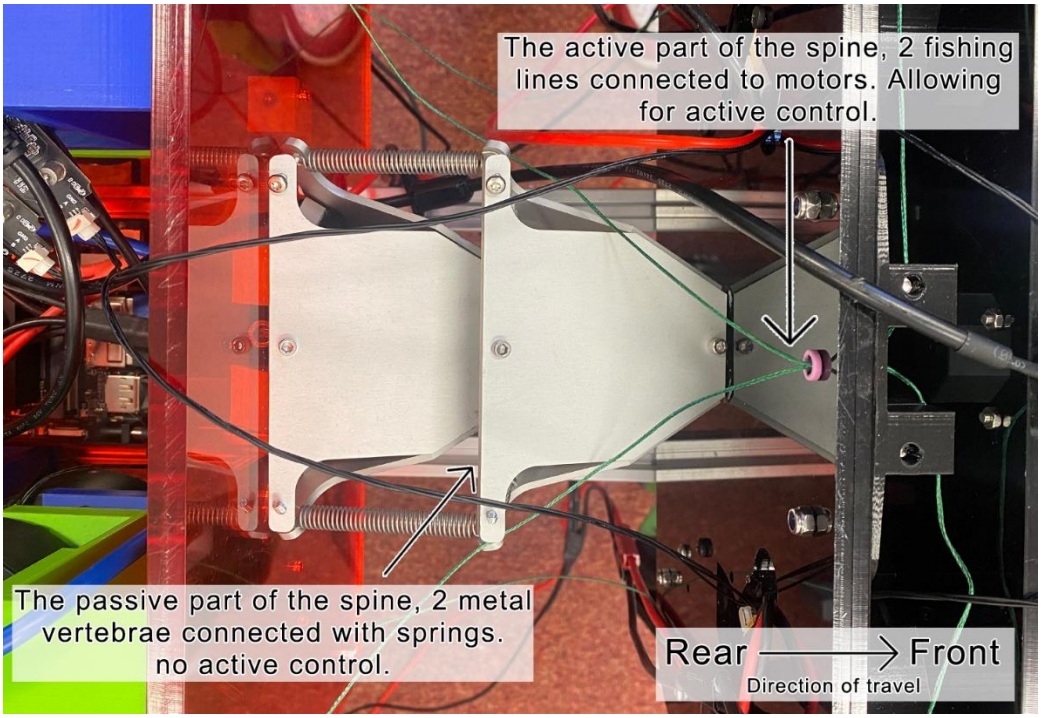


Figure 3.2, passive and active segments of TQbot's spine.

The passive metal sections are completely detached from motors and provide a rigid structure to connect the body together, whilst still maintaining flexibility through the use of springs. The active section comes in the form of fishing lines which feed directly into the four motors situated within TQbot's spine, allowing the fishing line to be pulled tauter or slacker through motor rotation, resulting in a flexible body (Figure 3.3). The fishing lines are setup identically rather than symmetrically across the front and rear sections, exiting the main body from a singular point in the rear, and two separate points in the front, shown in Figure 3.4. This allows for TQbot to achieve symmetrical rear and front roll motion [1] (Table 3.1, p. 58).

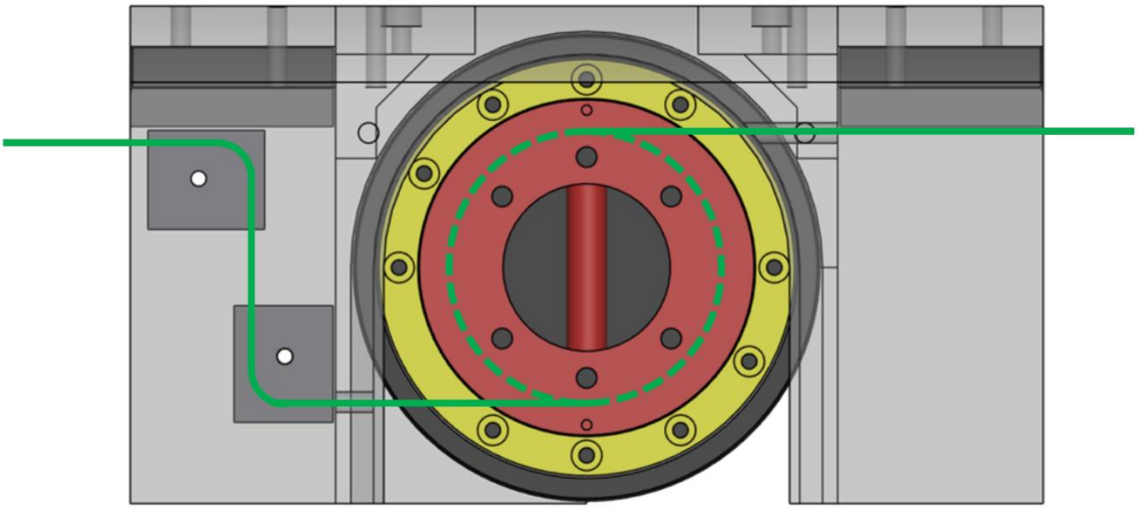


Figure 3.3, The fishing line (green) is wound on the winch (red), and encased within an external wall (yellow) to avoid the fishing line slipping. These are attached to the top of each of the four motors in TQbot's spine, allowing for rotation and thus movement of the fishing line. [1] (Figure 3.17, p. 65)

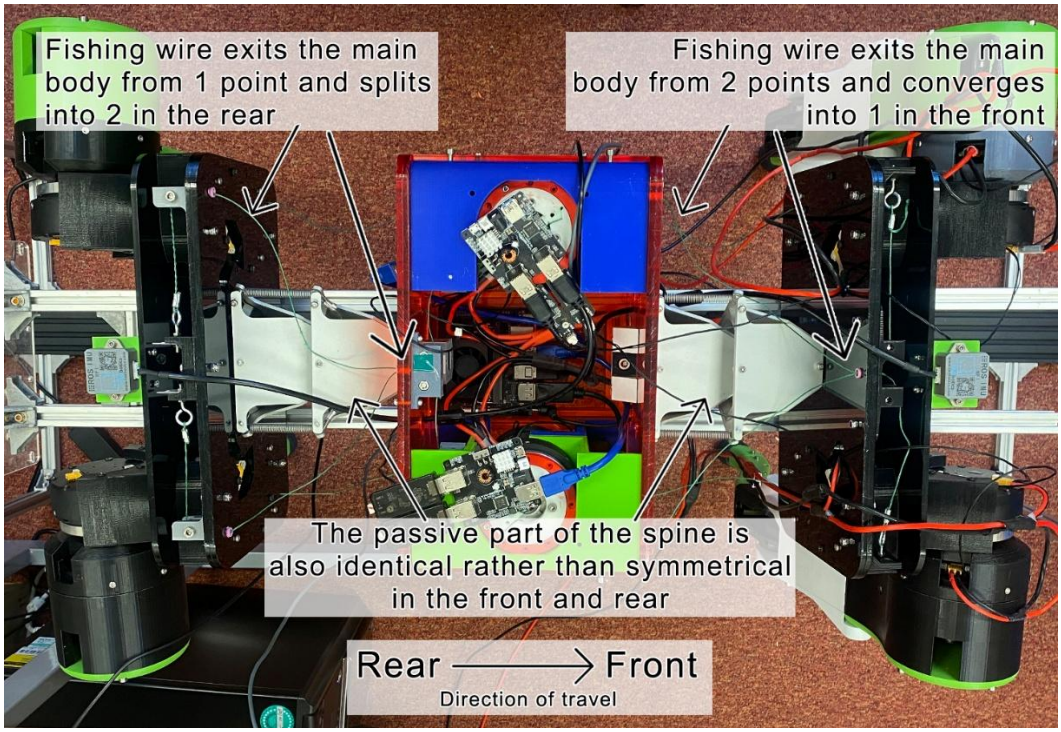


Figure 3.4, The rear spine and front spine are identical in both their passive and active components. This allows for symmetrical roll motion.

The front and rear body sections have IMUs mounted, allowing for more accurate readings of bodily rotation and velocity. Three battery modules were also completed, one to power the spinal motors and Jetson, and one each for the front and rear legs, however these were not present for my work as a bench power supply was a more reliable option for early tests and demonstration.

3.1.3 Legs

Lian left behind four legs, two left and two right. These legs use 3 Unitree A1 motors each to allow for operation of three joints: the shoulder, hip, and knee. The shoulder mounts into the front/rear body sections and rolls the leg, the hip rotates the leg in pitch, and the knee motor drives the calf via a belt drive (Figure 3.6), allowing pitch rotation of the calf independently of the hip. Each leg also contains a foot component, designed to house a pressure sensor for feedback.



Figure 3.5, An assembled leg. [1] (Figure 3.20, p. 67)

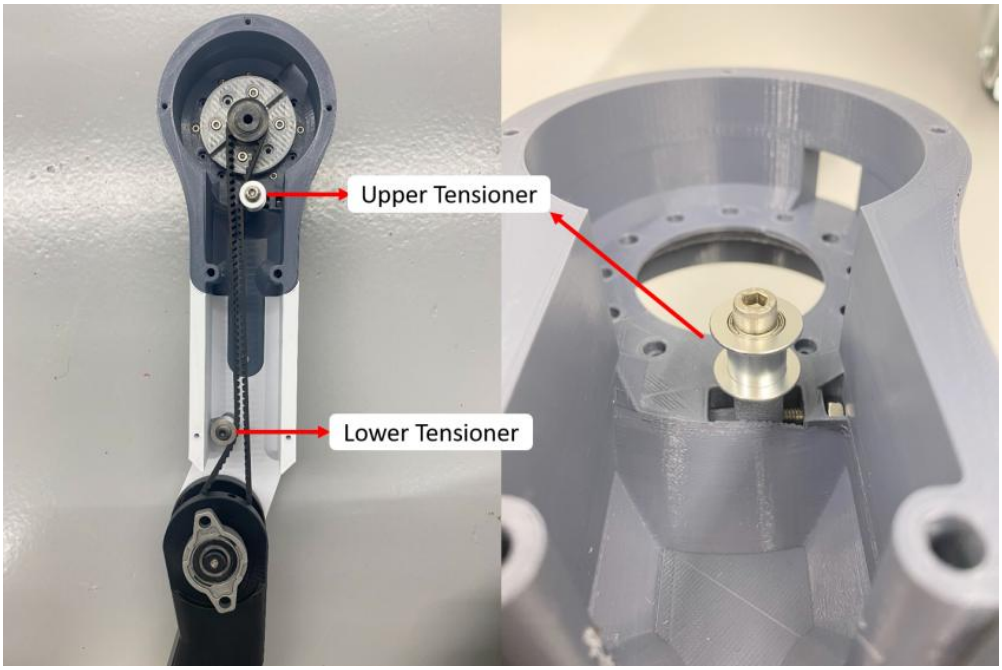


Figure 3.6, The belt drive present in each leg, allowing for exclusive pitch rotation of the knee. [1] (Figure 3.25, p. 71)

3.2 Inherited Software

Various standalone Python code was also inherited from Lian's prior work on TQbot. This included a motor wrapper class utilising the Unitree software development kit (SDK), a sympy kinematics solver which implemented Lian's kinematics equations [1] (pp. 86-88) for a singular leg, a posture controller for an arbitrary body height and orientation, a driver for the onboard IMUs, implementation of the CPGs, and a top level controller.

These files formed the foundation of my work, whereby I identified and patched the gaps and limitations present in these files, allowing for a full TQbot controller to be developed.

3.3 Gaps and Limitations

3.3.1 Hardware

TQbot's hardware was largely complete upon starting work, with all individual parts available. The identified gaps here were: the belt drives in the legs had not been attached, the legs were not screwed onto the body, the legs were not wired up with either power or communication, and the battery modules were not present. Though, the battery modules were not required at this stage.

3.3.2 Software

Gaps identified within inherited code informed the initial direction of this project, to bring TQbot to a state at which new functionality could be built on top of Lian's algorithms.

First, the inherited motor class opened its own RS485 serial port per motor. This conflicts with TQbot's physical bus network, where multiple motors share a singular RS485 line in each leg and across both sides of the spine. A refactor separating the per-motor class and bus-level transport was necessary for multi-motor functionality on TQbot.

Second, no safety infrastructure was present outside of the SDK checks within the motor class. These SDK checks covered the commanded position, velocity, and torque, however adding further safety guidelines such as an emergency stop and a maximum position jump to prevent erroneous user inputs were things I deemed necessary on a 30kg robot capable of producing 33.5Nm of torque per joint.

The last major gap identified was that all code existed as standalone Python modules with no real-time control loop and no inter-process communication layer, meaning every communication channel was a direct method call. ROS2 integration was therefore required as the architectural foundation for all future contributions to be built from.

4. Contributions

4.1 Motor Communication and Control

Initial motor work used a single Unitree A1 motor outside of TQbot, controlled by a Raspberry Pi 4B with ROS2 Jazzy, loaned from the school of PET. First, the Unitree SDK was compiled for the Pi's ARM64 architecture, and a single-motor ROS2 node implementing hybrid PD control was written, whereby position commanding within a measured error margin was demonstrated. From this work, a three-layer motor architecture was developed to handle communications across TQbot's six buses and sixteen motors, ready to port over to the Jetson Orin Nano running ROS2 Humble on board TQbot. The layers are as follows: 'Motor', 'MotorBus', and 'MultiMotorNode'.

The 'Motor' class is an abstraction of a single physical motor onboard TQbot, it holds the motor's ID, current state, command buffer, and safety configuration (See 4.3). The main functionality of this class is to convert user-side values into SDK rotor-side values, apply safety checks, and populate a command buffer for transmission to the physical motor. The use of a command buffer is particularly important, as this layer never communicates with the hardware directly due to there being no reference to a serial port. This decision allows for multiple motors to share a single serial port in 'MotorBus', as opposed to Lian's existing code where each motor owned its own serial port.

'MotorBus' is an abstraction of the RS485 connection, it owns a single RS485 serial port and holds a list of all 'Motor' objects sharing that port; matching TQbot's physical architecture where three leg motors share one RS485 line per leg, and two spine motors share one line on each side of the spine. The class exposes methods that command each motor in order, and receive their state data. Because the two-wire RS485 configuration used on TQbot is half-duplex [16], multiple motors cannot transmit simultaneously without collision. For this reason, a threading.Lock is established around the transmission block to allow for commands and replies of each motor on a bus to be sent in a single control tick without collision. Additionally, as each 'MotorBus' class owns its own serial port, multiple buses can run in parallel, allowing for simultaneous movement across limbs.

At the top layer, 'MultiMotorNode' wraps a 'MotorBus' class and exposes it through ROS2. It sets up per-motor target positions, per-motor state publications, a bus status topic, and services including enable, disable, emergency stop, and reset. Control runs at 300Hz on a ROS2 timer, and state is published at 100Hz on a second timer.

Each class has its own clear function; 'Motor' imports the SDK's type definitions, 'MotorBus' adds a SerialPort, and 'MultiMotorNode' adds ROS2. This means the bottom two layers are able to be

tested against a mock SerialPort with no ROS2 dependencies (See 4.8), and the top layer can be thought of exclusively in topics and services.

4.2 Safety Infrastructure

Due to TQbot weighing 30kg and having joints capable of producing 33.5Nm of torque, safety parameters were built into the motor stack to provide a line of defence against unexpected behaviours. There are four distinct safety guidelines operating across the motor stack: per-motor limits, position-jump rejection, bus emergency stops, and a brake-on-exit context manager.

'Motor' enforces per-motor limits with the SDK's bounds, as well as user-defined limits on position, velocity, and torque. Notably, a 'position jump' check rejects any movement whose target exceeds a defined delta from the most recent position, protecting against erroneous user input.

At bus level, 'MotorBus' includes an emergency stop which replaces motor commands with brake commands on every subsequent send cycle. Once set, the emergency stop can only be cleared by an explicit reset call from the user. A context manager guarantees that all motors are braked when the process exits, even if an exception such as a port disconnect on TQbot or Ctrl-C from the user occurs.

Additionally, at the node level bus state is continuously reported to the bus_status topic. This allows for the user to view diagnostics such as emergency stop state, motor temperature, and motor error conditions and plan accordingly. Motors with commands that violate input limits (SDK bounds and user-defined limits, including position-jump) will be skipped for that control tick, whilst the other motors on the bus with no violations will continue to be serviced.

4.3 Multi-Bus and Multi-Motor Validation

Once reliable and safe communications had been established and the multi-motor stack was built, the Jetson Orin Nano with ROS2 Humble onboard TQbot was then used to communicate with multiple motors across multiple buses. Due to limited power cables available, this was first achieved using four motors split across two spine buses. The first step in achieving this was to bind each USB-to-RS485 converter to udev rules under a recognisable device name. For the spine these were /dev/ttySpineA and /dev/ttySpineB, the legs were also added for when they became available under the names /dev/ttyLF, /dev/ttyRF, /dev/ttyLR, /dev/ttyRR.

Through these udev device names, multiple motors were able to be commanded simultaneously across the same and separate buses. After adjusting PD gains, TQbot's spine was commanded to hold a stable flat position under its own weight by incrementally pulling each motor to hold its

fishing wire tauter. The front left leg was then also attached for Demo-Day. With the two spine buses and front left leg proving operational with concurrent commands, I was able to demonstrate a tensioned spine with a moving leg.

4.4 Leg Connection

When the required power cables became available, focus was on attaching all four of TQbot's legs to allow for a stable standing posture to be demonstrated. This required attaching the knee joint belt drive of each leg, screwing the leg together, attaching the leg to TQbot's main body, and wiring up power and communication. As mentioned, the front left leg was attached first for Demo-Day. Upon confirming the multi-bus/multi-motor architecture to be operational with a single leg attached, the remaining three legs were attached.

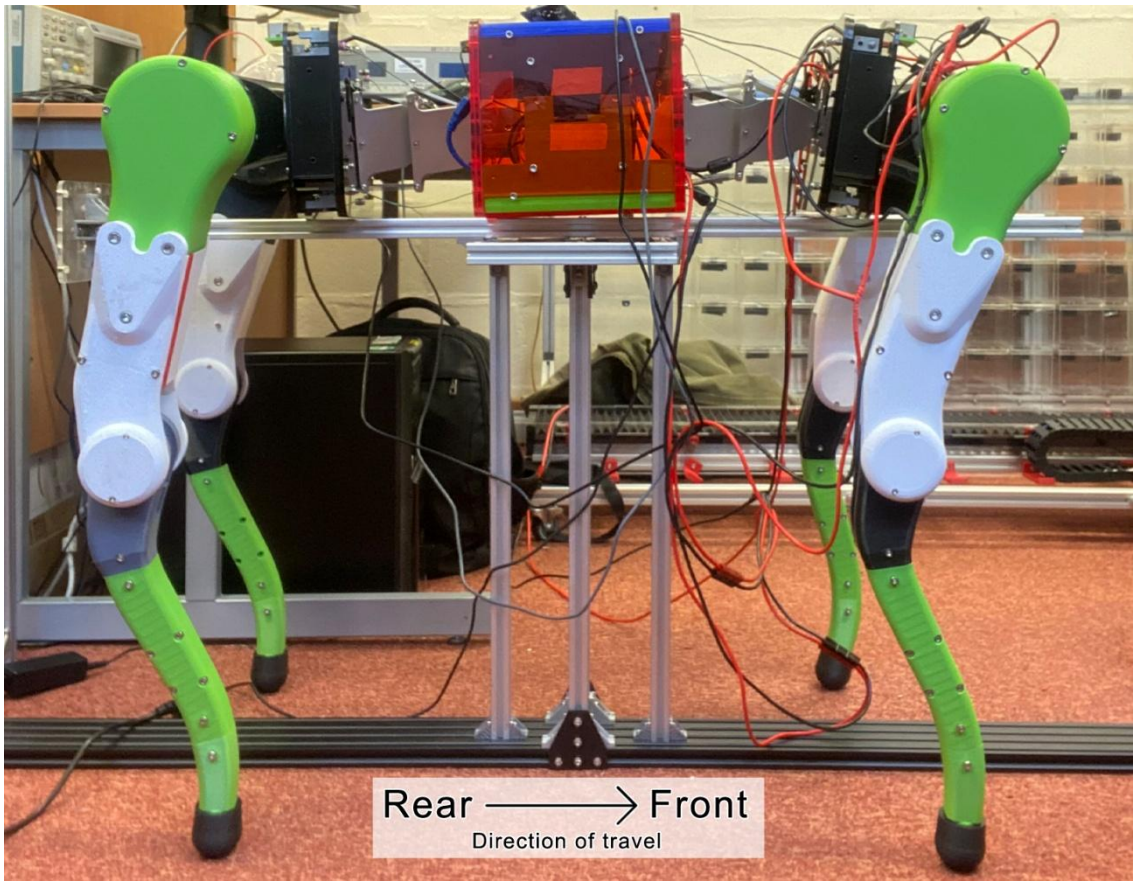


Figure 4.1, TQbot with all legs attached (power off).

4.5 Kinematics Layer

The kinematics solver inherited from Lian used sympy, a library that incurs runtime cost and is undesirable in a real-time control loop. Thus, a numerical reimplementation was developed which evaluates Lian's kinematics solutions [1] (pp. 86-88) through floating point arithmetic.

The layer contains a ‘LegKinematics’ class which accepts a leg identifier (front left, front right etc.) and flips the sign of the lateral hip offset for right-side legs, allowing for users to request the same Cartesian foot position on any leg and receive mirrored joint angles. A ‘KinematicsNode’ class subscribes to per-leg foot target topics and publishes the corresponding joint angles to achieve this foot target.

4.6 Gantry Assisted Standing Posture and PD Gains Tuning

With all leg buses operational, commanding the legs to a position where they would hold a stable standing posture was the final task. Due to motors now being required to hold the weight of their legs under gravity, further PD gains tuning was required beyond the gains used for a singular bench motor. The PD gains are now tuned to a point whereby the leg will hold a solid position and not move under force of my hand. More on this can be seen in 5.4 along with the final PD gains.

As mentioned in 4.5, kinematics solutions were implemented and validated in software. However, calibrating the legs such that the IK output produced meaningful physical motion proved difficult within the project timeframe. The standing posture demonstrated in this project therefore uses direct motor commands rather than IK foot positioning, whereby the user commands each motor to its required radian value and the PD gains hold the legs in this position under external force, more detail on this can be seen in Section 5.5.

4.7 Control Shell

Commanding TQbot directly through ROS2 requires lengthy operations across multiple terminals, which is not ideal for real-time testing. A Python control shell was therefore developed to wrap these operations into short functions that operate from a single terminal. For example, ‘SpineA(0.5, 0)’ turns motor0 in SpineA by 0.5 radians, replacing the equivalent ROS2 operation ‘ros2 topic pub /tqbot/SpineA/motor_0/target std_msgs/msg/Float64 "{data: 0.5}"’. Additionally, the shell contains a help method, which displays all available functions and what they do. The output of this help method can be seen in the Appendix.

The shell itself is also a ROS2 node. This means publishers and subscribers are initialised once and remain live, so the shell can both observe and command motors directly. It also operates in the same ROS2 network as the motor bus nodes, allowing both to be inspected through standard ROS2 diagnostic tools.

4.8 Validation Tests

Due to the code architecture, validation of the motor stack and kinematics was possible without requiring physical hardware, instead using mock values. For the motor stack, a ‘MockSerialPort’ class simulates the Unitree SDK’s serial interface, populating it with motor state responses. This allows for the motor stack to be fully tested in software. The kinematics layer does not require mock hardware as it is exclusively mathematics, instead test functions are executed across a grid of foot positions within a defined workspace area using TQbot’s link lengths. Allowing for thousands of input combinations to be tested in a relatively small number of test definitions. More on this can be seen in Section 5.

5. Tests and Results

5.1 Motor Stack Validation

To confirm connectivity and to report any faults before committing to commanding motors, a ‘ping_raw.py’ script was generated to allow for each motorID on each bus to be queried at a low torque. In the case of a motor being unreachable, it is flagged to the user and allows for fixes to be made before launching the full stack and potentially causing damage to TQbot.

Beyond this, twenty-two tests are in place for the motor stack under simulated concurrent threads, including unit conversions between SDK and user-friendly values, limit enforcement, emergency stop and reset, brake-on-exit behaviour, and thread-safe bus access under concurrent commands. All twenty-two tests pass; full test output is reproduced in the Appendix.

5.2 Kinematics Validation

Eighty-eight parameterised tests cover the kinematics layer, evaluating round-trip residuals across the defined workspace for all four legs. The residuals are Euclidean distances between commanded foot positions and the positions recovered after applying IK followed by FK, and quantify the numerical accuracy of the solutions. Workspace plots showing the returned foot positions of the front-left leg is shown in Figure 5.1.

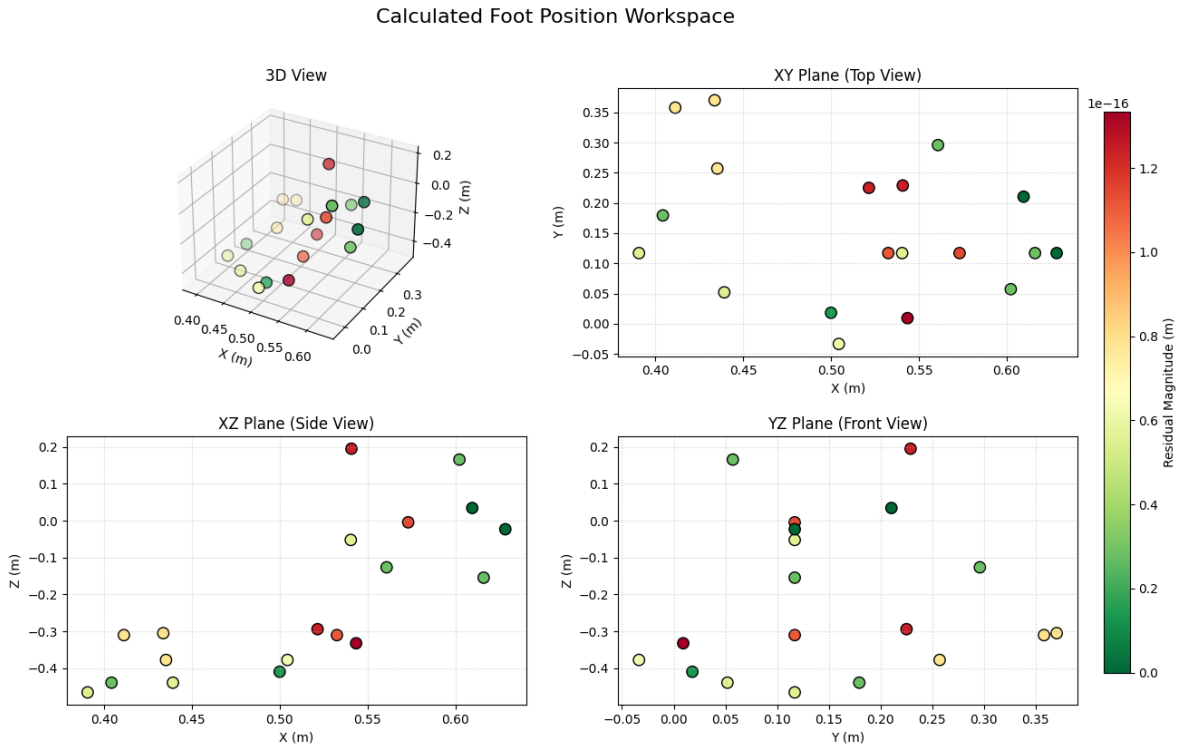


Figure 5.1, returned foot positions of the front-left leg when run through the kinematics solutions. Colour coded to show extremity of residual.

All eighty-eight tests pass; full test output is reproduced in the Appendix.

5.3 Tensioned Spine with Leg Movement Demonstration

On 29th April 2026, a demonstration of TQbot's spine and front-left leg was conducted in PT410 as part of Demo-Day, attended by students, visitors, and staff over a period of approximately two hours. With both spine buses (SpineA, SpineB) and the front-left leg bus operational under the multi-bus architecture described in Section 4.3, the spine was initially commanded to hold a tensioned flat configuration while the front-left leg was commanded to various targets.

The demonstration validated multi-bus concurrent operation in front of an audience for a sustained period, with no unexpected behaviours observed on the leg bus. However, the spine motors overheated during the tension hold, causing the safety infrastructure to automatically disable the spine buses. Exact temperatures were not recorded due to demonstration, but the spine was powered down for the remainder of the demonstration, and focus instead shifted to showcasing position control of the leg.

Whilst the cause of the overheating spine remains unverified, I have two primary hypotheses:

- The PD gains used at this stage of development were overly stiff, causing the motors to become hyper-sensitive and oscillate on a micro scale
- Due to the front-left leg being the only leg attached, a weight imbalance was present. This caused there to be a weight on the spinal motors in one direction with no leg to counterbalance this pull in the other direction.

Ideally, the demo setup would've been replicated in a controlled environment and test data would've been collected to verify the cause of the overheating spine. However, TQbot's spine sustained mechanical damage during transit, and so is out of use. Consequently, further PD tuning took place exclusively on the leg motors.

5.4 PD Gain Tuning Results

As mentioned in 5.3, additional PD tuning took place due to concern that this was the root cause of the spinal motors overheating. The SDK accepts PD gains relative to the rotor-side of the 9.1:1 gearbox, which requires careful scaling as high rotor-side gains get amplified by the gear reduction.

Initial values used on the bench motor were $K_p = 7$ and $K_d = 2$, which achieved stable position holding under no weight. Once using motors onboard TQbot, these values were increased significantly to K_p

= 70, K_d = 7 for the spine and K_p = 50, K_d = 7 for the leg. This was done to combat the pressure on the motors from TQbot's weight.

Following Demo-Day, further tuning with all legs attached and spine disabled took place. It was discovered that the legs required no adjustment to their K_p, remaining at 50. However, their K_d, were successfully reduced from 7 to 3, significantly reducing noisiness of the motors.

Stage	Target	Kp	Kd	Result
Bench Motor	Motor	7.0	2.0	Motor approached position and held under no pressure
Demonstration	Spine	70.0	7.0	Motors overheated
	Leg	50.0	7.0	Motors approached position and held under weight pressure
Further tuning	Leg	50.0	3.0	Motors approached position and held under weight and external pressure

Table 5.1, PD gains and their results.

5.5 Gantry Assisted Standing Posture Demonstration

With all four legs operational and the kinematics layer implemented in software, the final primary objective of the project was to deploy a physical standing posture on TQbot. To achieve this, the physical response of each of the motors was recorded when a positive radian was input.

Leg / Motor	Front-Left	Front-Right	Back-Left	Back-Right
0 (Shoulder)	Abducts leg outwards	Adducts leg towards body	Adducts leg towards body	Abducts leg outwards
1 (Hip)	Points leg backwards	Points leg forwards	Points leg backwards	Points leg forwards
2 (Knee)	Points calf backwards	Points calf forwards	Points calf backwards	Points calf forwards

Table 5.2, physical response to a positive radian input.

The initial strategy for achieving a standing posture involved using 'KinematicsNode' which wraps the kinematics solutions in a ROS2 node and allows for the user to command TQbot via foot positions rather than individual motor radians. However, this required precise calibration to map the physical motor encoder zero-points to the zero-pose defined by the kinematics model. One resting configuration example can be seen in Figure 5.2, whereby the legs buckle inwards and rest on each other. However, during testing this configuration often resulted in residual errors which

would produce unpredictable Cartesian movements, and it became evident that holding the legs in perfectly repeatable reference configurations would not be possible in the given timeframe.



Figure 5.2, Proposed repeatable resting positon whilst on the gantry.

Due to this, it was decided that a standing posture would be demonstrated using direct motor commands rather than full kinematics integration. This was done by iteratively commanding each leg's motors to hold a standing configuration. Once in a standing posture, force was applied to each leg individually and confirmed that each leg could hold its position under force.

6. Planning and Outcomes

6.1 Initial Plan

Six objectives were defined in the initial report, building from singular motor development to a full trot gait, these are outlined in Table 6.1. Phase one of the plan covered objectives 1-3 using a single Unitree A1 motor on a Raspberry Pi 4B with ROS2 Jazzy. Phase two covered objectives 4-6 using all sixteen Unitree motors and the Jetson Orin Nano onboard TQbot. The time frames I aimed to achieve these objectives can be seen in the Gantt chart below.

Shortly after the initial plan was set, it was observed that the Jetson runs ROS2 Humble rather than Jazzy. This was a slight deviation from the plan but did not affect work flow.

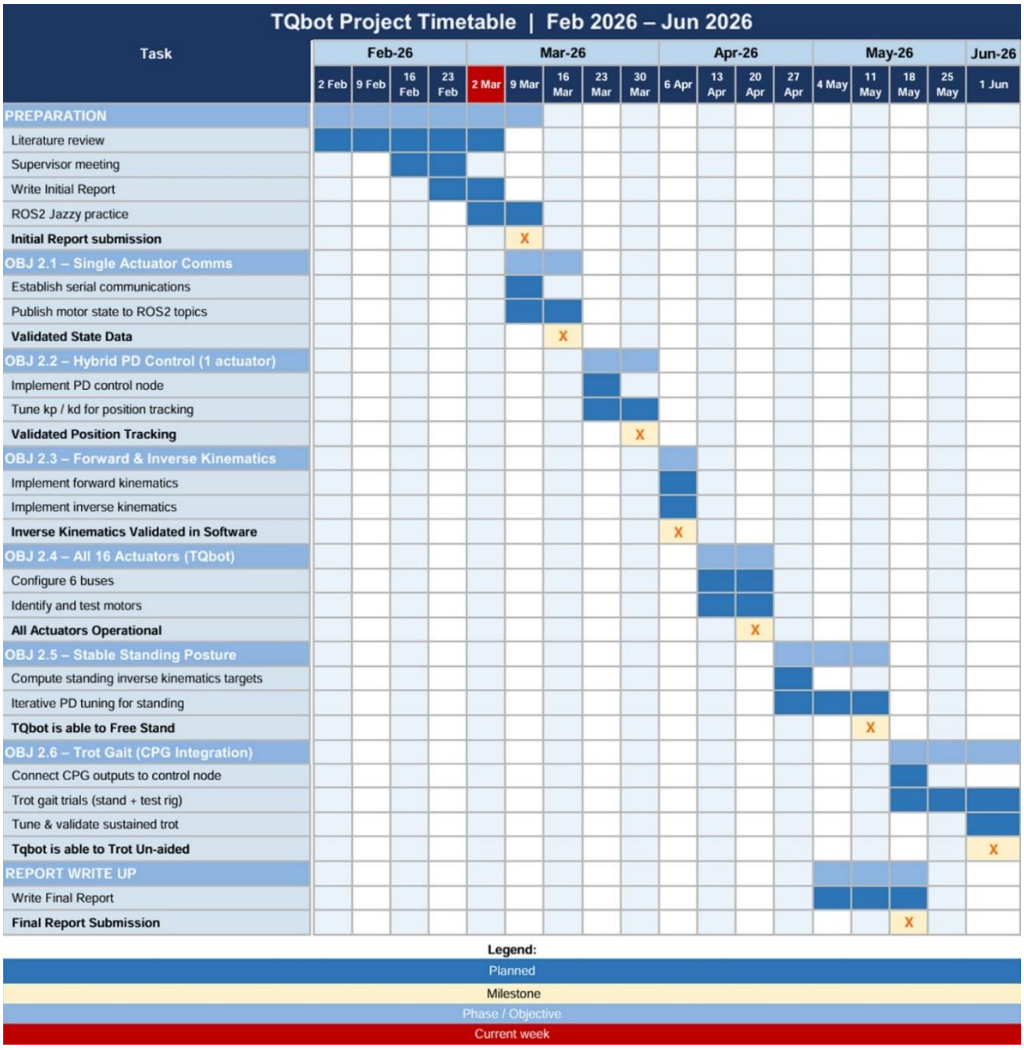


Figure 6.1, Proposed Gantt chart.

6.2 Outcomes Against Objectives

Objective	Outcome
1: Singular motor communication	**Met (late March)**, Pi reads and publishes motor state via ROS2 Jazzy.
2: Hybrid PD control on single motor	**Met (mid-April)**, Bench motor can be commanded to a desired position within a measured error margin
3: Forward and inverse kinematics	**Met in software (mid-April)**, All 88 round-trip residuals are below 1.3×10^{-16} m. Physical calibration to the kinematic zero-pose was not achieved.
4: Control of all sixteen actuators	**Partially met (late April)**, Multi-bus architecture validated across three buses at Demo-Day, and across four buses to demonstrate the legs at a standing posture. There should be no issue controlling all motors simultaneously, however due to spine damage in transit, the spine has not officially been controlled at the same time as all four legs.
5: Stable standing posture	**Partially met (mid-May),** Gantry-assisted standing achieved via direct motor commands, not kinematics (Section 5.5). Free-standing not demonstrated due to spinal damage.
6: Basic trot gait from CPG	**Not met,** not attempted due to spinal damage and kinematics calibration not being completed in time.

Table 6.1, Objectives and their final outcomes.

6.3 Limitations and Reflections

Phase one began approximately four weeks late due to delays gaining building access to the ISA. Key-card access to the building required a safety induction, which took three weeks before a slot was confirmed, and card permissions took an additional week to be applied. This was not anticipated in the initial plan, and put a strain on the first three objectives. Objective 1 was completed three weeks behind schedule, objective 2 was completed two weeks behind schedule, and objective 3 was completed on time in mid-April.

Upon completing phase one in mid-April and migrating to the Jetson for phase two, a second setback was identified. Lian's inherited code required more refactoring than anticipated to achieve an operational ROS2 control stack. The additional three-layer architecture, safety infrastructure, and test files were necessary to begin phase two objectives.

Phase two was further constrained by a shortage of power cables, which meant the front-left leg was the only leg attached during Demo-Day. After Demo-Day Mark Post was able to make the additional wires required, and I was able to properly start phase two in early May, approximately two weeks behind schedule. This is when it was noticed that TQbot's spine had been damaged in transit, requiring a change in plan when proceeding with the following objectives. After notifying Mark Post of the damaged spine, he modified the gantry to hold the spine flat when powered off. This allowed for phase two to be attempted without risking further damage to TQbot's spine, but meant that operation of all sixteen motors would not be possible.

On reflection, these setbacks provided significantly less time to achieve the objectives than initially planned, and is a main reason the final objectives were not fully met. However, the layered motor architecture, kinematics software, test files, and control shell will remain useful to anyone completing future work once the spine is repaired.

7. Statement of Ethics

As this project involved a 30kg robot with moving parts, the primary ethical concern related to safety. The Unitree A1 motors used on TQbot are capable of producing torque of up to 33.5 Nm [1] (p. 74), which posed a risk of injury during testing if the robot behaved unexpectedly. To mitigate this risk, the University of York's Institute for Safe Autonomy (ISA) was used for the majority of work on TQbot, under controlled laboratory environments. All commandment of TQbot took place with the assistance of a gantry, and this should remain the standard for any future work before moving to free-standing tests, to guide safe actuator usage before commitment under the full weight of TQbot.

TQbot is typically powered by lithium-ion batteries [1] (p. 77), though all of my work was powered with a bench power supply, so these posed no impact. However, under typical work, lithium-ion batteries carry a small fire risk if damaged or improperly charged, thus battery safety procedures should be understood via a risk assessment and enforced.

Prior to completing any work on TQbot, a general and safety induction to the ISA were completed, as well as a risk assessment for the bench power supply. A risk assessment for handling TQbot was also created when TQbot was demonstrated at the school of Physics, Engineering, and Technology (PET). These should be completed by any person handling TQbot or working on TQbot in the ISA in the future.

8. Conclusions

This project intended to develop the motor control software required to bring TQbot from a partially assembled quadruped with simulated CPGs, to a platform capable of a standing posture and basic trot gait. The work delivered a three-layer ROS2 control architecture modelling TQbot's bus topology, a safety infrastructure spanning per-motor limits and bus-level control, a numerical reimplementation of Lian's kinematics solutions, and a Python control shell that collapses ROS2 commands into simple function calls. Twenty-two motor stack tests pass against simulated hardware, and eighty-eight kinematics tests pass within 0.1 femtometres.

Of the six objectives set initially, three were fully met, two were partially met, and one was not met. This resulted in a gantry-assisted standing posture demonstrated using direct motor commands, with legs capable of holding position under external force. Kinematics based standing and trot gait were not achieved, primarily due to spinal damage and time constraints outlined in Section 6.

Whilst the original final goal for TQbot was not fully realised, this project has bridged the gap left in Lian's prior work. The layered architecture, kinematics layer, safety infrastructure, control shell, and test suites are designed to be extended, and provide a starting point for the standing and locomotion work that follows.

9. Future Work

While this dissertation has seen motor control and a standing posture implemented on TQbot, a tensegrity-spine quadruped robot developed at the University of York, future work is necessary to progress TQbot to an autonomous platform with locomotion.

The most crucial piece of future work is to demonstrate calibration of TQbot's legs relative to the kinematics solutions so that movement can occur in terms of foot position rather than per-motor radian inputs. This would allow for a repeatable standing posture configuration which is reachable on every boot of TQbot. From this, the CPG framework established by Lian [1] should be integrated to demonstrate locomotion gaits.

Once the spine has been fixed, future work should also focus on getting TQbot off of the gantry; demonstrating free standing postures and gaits which are held under its 30kg weight. To do this, the IMU's available on TQbot's rear/front body sections should be utilised to ensure there is no sagging in the body. Researchers may then progress into developing TQbot to be able to traverse complex real-world terrains, attaching devices such as depth sensors to enable TQbot to intelligently interact with its environment.

10. References

[1] Y. Lian, "Design, modelling and control of a quadruped robot with a bionic spine," Ph.D. thesis, Dept. of Physics, Engineering and Technology, Univ. of York, York, UK, 2024. [Online]. Available: https://etheses.whiterose.ac.uk/id/eprint/36206

[2] Y. Zhong, R. Wang, H. Feng, and Y. Chen, “Analysis and research of quadruped robot’s legs: A comprehensive review,” *International Journal of Advanced Robotic Systems*, vol. 16, no. 3, 2019, doi: 10.1177/1729881419844148.

[3] S. Nakajima and K. Ietomi, "Concept of adaptive gait for leg-wheel robot, RT-Mover," 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), Guangzhou, China, 2012, pp. 293-300, doi: 10.1109/ROBIO.2012.6490982.

[4] Boston Dynamics, “Spot robot.” [Online]. Available: https://bostondynamics.com/products/spot/

[5] R. D. Quinn, J. T. Offi, D. A. Kingsley and R. E. Ritzmann, "Improved mobility through abstracted biological principles," IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, 2002, pp. 2652-2657 vol.3, doi: 10.1109/IRDS.2002.1041670.

[6] J. M. Morrey, B. Lambrecht, A. D. Horchler, R. E. Ritzmann and R. D. Quinn, "Highly mobile and robust small quadruped robots," Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), Las Vegas, NV, USA, 2003, pp. 82-87 vol.1, doi: 10.1109/IROS.2003.1250609.

[7] Unitree Robotics, “Unitree Robotics.” [Online]. Available: https://www.unitree.com

[8] Unitree A1, “Unitree A1 quadruped robot.” [Online]. Available: https://www.unitree.com/a1

[9] A. J. Ijspeert, “Central pattern generators for locomotion control in animals and robots: A review,” *Neural Networks*, vol. 21, no. 4, pp. 642–653, 2008, doi: 10.1016/j.neunet.2008.03.014.

[10] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, 2022, doi: 10.1126/scirobotics.abm6074.

[11] Robot Operating System 2 Documentation, “ROS 2 Jazzy documentation.” [Online]. Available: https://docs.ros.org/en/jazzy/index.html

[12] Robot Operating System 2 Documentation, “ROS 2 Humble documentation.” [Online]. Available: https://docs.ros.org/en/humble/index.html

[13] Skelton, Robert & Oliveira, Mauricio. (2009). Tensegrity Systems. 10.1007/978-0-387-74242-7.

[14] A. P. Sabelhaus et al., "System design and locomotion of SUPERball, an untethered tensegrity robot," 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 2015, pp. 2867-2873, doi: 10.1109/ICRA.2015.7139590.

[15] A. P. Sabelhaus et al., “Design, Simulation, and Testing of a Flexible Actuated Spine for Quadruped Robots,” 2018. arXiv:1804.06527

[16] T. Kugelstadt, "The RS-485 Design Guide," Application Report SLLA272, Texas Instruments. [Online]. Available: https://www.ti.com/lit/pdf/slla272

Appendix

Output of 'help()' Function

```
TQbot shell

-------------------------

QUICK START

  start_all()                       launch + enable every configured bus

  status()                          confirm every bus shows OK, motors have data

  spine(-2, -2, -2, -2)             send commands to all spine motors

  fl(0.0, 0.5, -1.0)                send commands to all FL leg motors

  stand()                           execute staggered standing posture routine

  stop_all()                        disable + kill every bus

  exit()                            leave the shell


LAUNCH / KILL

  launch('FL')                      spawn one bus node, default gains

  launch('FL', default_kp=500)      spawn with a parameter override

  launch_all()                      spawn every bus in LAUNCH_CONFIGS

  kill('FL')                        SIGTERM a node we launched

  kill_all()                        kill every node we launched

  processes()                       list our launched processes + PIDs


ENABLE / DISABLE  (engage or brake PD on a running node)

  enable('FL')                      engage PD control

  disable('FL')                     brake (soft, not latched)

  enable_all() / disable_all()      every configured bus


FULL START/STOP

  start('FL')                       launch + enable in one go

  stop('FL')                        disable + kill in one go

  start_all() / stop_all()          every bus in LAUNCH_CONFIGS


EMERGENCY STOP

  estop('FL')                       latch the bus into brake

  estop_all()                       e-stop every configured bus

  reset_estop('FL')                 clear latch (bus stays disabled)
```

```
STATE INSPECTION
  status()                          full state of every motor (pos / vel / eff)
  status_summary('FL', 'FR')        compact one-line per listed bus
  hold()                            status() in a loop, Ctrl-C to stop

LIVE PARAMETER EDITING
  set_param('FL', 'default_kp', 400.0)
  set_param('FL', 'ramp_rate', 0.5)
  get_param('FL', 'default_kp')

  Editable parameters (all doubles):
    default_kp, default_kd          PD gains
    ramp_rate                       max command slew (rad/s, output-side)
    max_position_jump               safety: allowed gap cmd vs actual (rad)
    position_min, position_max      soft position limits (rad)

COMMAND MOTOR POSITIONS  (all values in radians, output-side)
  target('FL', 0, 1.5)
      single motor: bus, motor_id, position

  spine(a0, a1, b0, b1)
      all four spine motors at once:
        a0: SpineA motor 0    a1: SpineA motor 1
        b0: SpineB motor 0    b1: SpineB motor 1
      e.g. spine(-2, -2, -2, -2)

  fl(m0, m1, m2) / fr(...) / rl(...) / rr(...)
      all three leg motors at once (m0: shoulder, m1: hip, m2: knee)
      e.g. fl(0.0, 0.5, -1.0)

KINEMATICS & CARTESIAN CONTROL (X, Y, Z in meters inside local hip frame)
  show_kin_tables()                 print structural mappings, signs, and offsets
  calibrate('FL')                   save current motor angles as IK zero pose
```

```
  calibrate_all()                   interactive step-by-step wizard to calibrate all
legs
  foot('FL', x, y, z)               send leg to 3D coordinate using Inverse Kinematics


LOGS
  Each launched bus writes to  /tmp/tqbot_<busname>.log
  Watch in a separate terminal:
      tail -f /tmp/tqbot_FL.log


EXIT
  stop_all(), exit()                shutdown
```

Motor Stack Test Outputs

```
### Motor Stack Test Results
| Test ID | Suite | Target Method / Functional Case | Status | Notes |
|---|---|---|---|---|
| 01 | Bus | test_brake_all | PASS | Nominal execution |
| 02 | Bus | test_concurrent_calls_serialised | PASS | Nominal execution |
| 03 | Bus | test_context_manager_brakes_on_exit | PASS | Nominal execution |
| 04 | Bus | test_duplicate_motor_id_rejected | PASS | Nominal execution |
| 05 | Bus | test_emergency_stop_forces_brake | PASS | Nominal execution |
| 06 | Bus | test_estop_reset_clears_flag | PASS | Nominal execution |
| 07 | Bus | test_find_motor | PASS | Nominal execution |
| 08 | Bus | test_motor_error_detected | PASS | Nominal execution |
| 09 | Bus | test_overheating_detected | PASS | Nominal execution |
| 10 | Bus | test_position_converges_toward_target | PASS | Nominal execution |
| 11 | Bus | test_send_all_commands_touches_every_motor | PASS | Nominal execution |
[TEST] motor 0 sendRecv failed: [MockSerialPort /dev/fake_bus] injected failure
[TEST] motor 1 sendRecv failed: [MockSerialPort /dev/fake_bus] injected failure
[TEST] motor 2 sendRecv failed: [MockSerialPort /dev/fake_bus] injected failure
| 12 | Bus | test_sendrecv_failure_returns_false | PASS | Nominal execution |
| 13 | Bus | test_three_motors_registered | PASS | Nominal execution |
| 14 | Unit | test_constructs_cleanly | PASS | Nominal execution |
| 15 | Unit | test_hybrid_command_preserves_torque | PASS | Nominal execution |
| 16 | Unit | test_kd_conversion_within_sdk_bounds | PASS | Nominal execution |
| 17 | Unit | test_kp_conversion_within_sdk_bounds | PASS | Nominal execution |
| 18 | Unit | test_output_to_rotor_roundtrip_angle | PASS | Nominal execution |
| 19 | Unit | test_output_to_rotor_roundtrip_velocity | PASS | Nominal execution |
| 20 | Unit | test_prepare_stop_command_always_safe | PASS | Nominal execution |
[Motor 0] position jump 4.550 > allowed 0.910 (rotor)
| 21 | Unit | test_safety_rejects_large_position_jump | PASS | Nominal execution |
[Motor 0] q_out 2.000 outside [-1.0, 1.0]
| 22 | Unit | test_safety_rejects_out_of_range_position | PASS | Nominal execution |

Total test executions: 22
```

Kinematics Test Outputs

Kinematics Test Results
Target residual tolerance: 1e-09 m

Test ID	Leg	Type	theta_1 (deg)	theta_2 (deg)	theta_3 (deg)	Residual Error (m)	Status
01	FL	Grid	+0.0	+45.8	-28.6	1.1102e-16	PASS
02	FR	Grid	+0.0	+45.8	-28.6	1.1102e-16	PASS
03	RL	Grid	+0.0	+45.8	-28.6	1.1102e-16	PASS
04	RR	Grid	+0.0	+45.8	-28.6	1.1102e-16	PASS
05	FL	Grid	+0.0	+68.8	-34.4	5.5511e-17	PASS
06	FR	Grid	+0.0	+68.8	-34.4	5.5511e-17	PASS
07	RL	Grid	+0.0	+68.8	-34.4	5.5511e-17	PASS
08	RR	Grid	+0.0	+68.8	-34.4	5.5511e-17	PASS
09	FL	Grid	+0.0	+17.2	-5.7	2.7756e-17	PASS
10	FR	Grid	+0.0	+17.2	-5.7	2.7756e-17	PASS
11	RL	Grid	+0.0	+17.2	-5.7	2.7756e-17	PASS
12	RR	Grid	+0.0	+17.2	-5.7	2.7756e-17	PASS
13	FL	Grid	+0.0	+40.1	-63.0	5.5511e-17	PASS
14	FR	Grid	+0.0	+40.1	-63.0	5.5511e-17	PASS
15	RL	Grid	+0.0	+40.1	-63.0	5.5511e-17	PASS
16	RR	Grid	+0.0	+40.1	-63.0	5.5511e-17	PASS
17	FL	Grid	+0.0	+28.6	-51.6	1.1444e-16	PASS
18	FR	Grid	+0.0	+28.6	-51.6	1.1444e-16	PASS
19	RL	Grid	+0.0	+28.6	-51.6	1.1444e-16	PASS
20	RR	Grid	+0.0	+28.6	-51.6	1.1444e-16	PASS
21	FL	Grid	+0.0	+11.5	-17.2	0.0000e+00	PASS
22	FR	Grid	+0.0	+11.5	-17.2	0.0000e+00	PASS
23	RL	Grid	+0.0	+11.5	-17.2	0.0000e+00	PASS
24	RR	Grid	+0.0	+11.5	-17.2	0.0000e+00	PASS
25	FL	Grid	-5.7	+0.0	-28.6	2.7756e-17	PASS
26	FR	Grid	-5.7	+0.0	-28.6	1.1444e-16	PASS
27	RL	Grid	-5.7	+0.0	-28.6	2.7756e-17	PASS
28	RR	Grid	-5.7	+0.0	-28.6	1.1444e-16	PASS
29	FL	Grid	+11.5	+0.0	-34.4	1.2413e-16	PASS
30	FR	Grid	+11.5	+0.0	-34.4	3.1032e-17	PASS
31	RL	Grid	+11.5	+0.0	-34.4	1.2413e-16	PASS
32	RR	Grid	+11.5	+0.0	-34.4	3.1032e-17	PASS
33	FL	Grid	+8.6	+0.0	-5.7	0.0000e+00	PASS
34	FR	Grid	+8.6	+0.0	-5.7	6.9389e-18	PASS
35	RL	Grid	+8.6	+0.0	-5.7	0.0000e+00	PASS
36	RR	Grid	+8.6	+0.0	-5.7	6.9389e-18	PASS
37	FL	Grid	-11.5	+40.1	+0.0	1.3878e-17	PASS
38	FR	Grid	-11.5	+40.1	+0.0	0.0000e+00	PASS
39	RL	Grid	-11.5	+40.1	+0.0	1.3878e-17	PASS
40	RR	Grid	-11.5	+40.1	+0.0	0.0000e+00	PASS
41	FL	Grid	+28.6	+28.6	+0.0	7.8505e-17	PASS
42	FR	Grid	+28.6	+28.6	+0.0	5.5511e-17	PASS
43	RL	Grid	+28.6	+28.6	+0.0	7.8505e-17	PASS
44	RR	Grid	+28.6	+28.6	+0.0	5.5511e-17	PASS
45	FL	Grid	+17.2	+11.5	+0.0	2.7756e-17	PASS
46	FR	Grid	+17.2	+11.5	+0.0	1.1857e-16	PASS
47	RL	Grid	+17.2	+11.5	+0.0	2.7756e-17	PASS
48	RR	Grid	+17.2	+11.5	+0.0	1.1857e-16	PASS
49	FL	Grid	+17.2	+51.6	-25.8	7.8505e-17	PASS
50	FR	Grid	+17.2	+51.6	-25.8	6.2063e-17	PASS
51	RL	Grid	+17.2	+51.6	-25.8	7.8505e-17	PASS
52	RR	Grid	+17.2	+51.6	-25.8	6.2063e-17	PASS
53	FL	Grid	-17.2	+51.6	-25.8	6.2063e-17	PASS
54	FR	Grid	-17.2	+51.6	-25.8	7.8505e-17	PASS

55	RL	Grid	-17.2	+51.6	-25.8	6.2063e-17	PASS
56	RR	Grid	-17.2	+51.6	-25.8	7.8505e-17	PASS
57	FL	Grid	+8.6	+63.0	-31.5	2.7756e-17	PASS
58	FR	Grid	+8.6	+63.0	-31.5	5.5943e-17	PASS
59	RL	Grid	+8.6	+63.0	-31.5	2.7756e-17	PASS
60	RR	Grid	+8.6	+63.0	-31.5	5.5943e-17	PASS
61	FL	Grid	-8.6	+63.0	-31.5	5.5943e-17	PASS
62	FR	Grid	-8.6	+63.0	-31.5	2.7756e-17	PASS
63	RL	Grid	-8.6	+63.0	-31.5	5.5943e-17	PASS
64	RR	Grid	-8.6	+63.0	-31.5	2.7756e-17	PASS
65	FL	Grid	+11.5	+37.2	-17.2	1.2413e-16	PASS
66	FR	Grid	+11.5	+37.2	-17.2	5.5943e-17	PASS
67	RL	Grid	+11.5	+37.2	-17.2	1.2413e-16	PASS
68	RR	Grid	+11.5	+37.2	-17.2	5.5943e-17	PASS
69	FL	Grid	-11.5	+43.0	-20.1	1.3329e-16	PASS
70	FR	Grid	-11.5	+43.0	-20.1	8.3267e-17	PASS
71	RL	Grid	-11.5	+43.0	-20.1	1.3329e-16	PASS
72	RR	Grid	-11.5	+43.0	-20.1	8.3267e-17	PASS
73	FL	Grid	+28.6	+45.8	-28.6	7.8505e-17	PASS
74	FR	Grid	+28.6	+45.8	-28.6	7.8505e-17	PASS
75	RL	Grid	+28.6	+45.8	-28.6	7.8505e-17	PASS
76	RR	Grid	+28.6	+45.8	-28.6	7.8505e-17	PASS
77	FL	Stance	+0.0	+49.2	+74.1	1.1102e-16	PASS
78	FR	Stance	+0.0	+49.2	+74.1	1.1102e-16	PASS
79	RL	Stance	+0.0	+49.2	+74.1	1.1102e-16	PASS
80	RR	Stance	+0.0	+49.2	+74.1	1.1102e-16	PASS
81	FL	Origin	0.0	0.0	0.0	0.0000e+00	PASS
82	FR	Origin	0.0	0.0	0.0	0.0000e+00	PASS
83	RL	Origin	0.0	0.0	0.0	0.0000e+00	PASS
84	RR	Origin	0.0	0.0	0.0	0.0000e+00	PASS
85	FL	Bound	Clamp	Clamp	Clamp	0.0000e+00	PASS
86	FR	Bound	Clamp	Clamp	Clamp	0.0000e+00	PASS
87	RL	Bound	Clamp	Clamp	Clamp	0.0000e+00	PASS
88	RR	Bound	Clamp	Clamp	Clamp	0.0000e+00	PASS

Total test executions: 88