POLITECNICO DI TORINO

MASTER's Degree in MECHATRONICS ENGINEERING

MASTER's Degree Thesis

Simulation and Control of a Reconfigurable Robot

Supervisors

Prof. Giuliana MATTIAZZO

Prof. Fabio CARAPELLESE

Candidate

Jad ABI HANA

OCTOBER 2025

Abstract

This thesis presents the modeling, simulation, and control of a novel reconfigurable robotic system designed for modular locomotion and adaptable task execution. The robot, composed of multiple identical three-link units, is capable of reassembling into various structures suited for specific tasks such as obstacle climbing, stair ascent, object manipulation, and load lifting. The focus of this work is on the development of simulation tools and control strategies to enable coordinated locomotion of a single unit, with a particular emphasis on a worm-like walking motion.

Using MATLAB and Simscape Multibody, a detailed dynamic model of the robot was built, incorporating joint actuation via DC motors and trajectory generation through cubic polynomial interpolation. A torque-based control method was implemented to drive the robot's joints based on desired angular trajectories. Logical control schemes were designed to switch motor activation on or off depending on real-time joint angle feedback, ensuring energy-efficient movement.

The study also includes the integration of mechanical and electrical subsystems, validation of motion through simulation, and exploration of future extensions such as multi-unit coordination and surface climbing. The results demonstrate the feasibility of smooth, bidirectional worm-like locomotion and provide a foundation for future experimental prototyping and real-world application of modular reconfigurable robotics.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisors, Prof. Giuliana Mattiazzo and Prof. Fabio Carapellese, for their invaluable guidance, support, and encouragement throughout the course of this thesis. Their expertise and vision were fundamental in shaping this project, and their continuous feedback greatly enriched both the technical and conceptual aspects of my work.

I am especially thankful for the opportunity to work under their supervision on a challenging and innovative topic in the field of robotics. Their mentorship has been a source of inspiration and personal growth.

I would also like to acknowledge the faculty and staff at the Department of Mechatronics Engineering for providing a rich academic environment.

Finally, I extend my heartfelt appreciation to my family and friends for their unwavering support, patience, and motivation throughout this journey.

Table of Contents

List of Figures

List of Tables

Acronyms

USB Universal Serial Bus.

Chapter 1

Introduction

Accurate motion control and reconfiguration are pivotal challenges in modern robotics, especially in systems designed for unstructured or dynamic environments. The ability of a robotic system to adapt its morphology and movement strategy is critical.

This thesis explores the design, simulation, and control of a novel reconfigurable walking robot, composed of three-link units. The robot is capable of worm-like locomotion, coordinated through joint-level actuation driven by torque or voltage-controlled motors. The robot is designed to perform complex tasks such as climbing obstacles, ascending stairs, and lifting payloads by reassembling into various topologies depending on mission needs.

Using MATLAB, Simulink, and Simscape Multibody, this work focuses on the generation of smooth angular trajectories, implementation of motor control logic, and the development of simulation environments to analyze the robot's performance. The study not only demonstrates the feasibility of modular robot movement using synchronized joint control but also lays the groundwork for future physical prototyping and adaptive reconfiguration strategies.

1.1 Localization, Sensing, and Motor Control Overview

Accurate localization and motion control are fundamental for the reliable operation of reconfigurable walking robots. These systems must dynamically adapt their configuration to perform a variety of tasks, such as climbing, navigating irregular terrain, or manipulating objects. Achieving this level of adaptability requires a robust understanding of the robot's internal state—such as joint positions and orientations—as well as its interaction with the external environment.

The focus lies not only on the mechanical design and motion planning but also on establishing the foundational framework for real-time feedback and coordination. While detailed sensor selection and integration strategies will be discussed in subsequent sections, it is important to note that the effectiveness of such robots heavily depends on precise feedback mechanisms and reliable state estimation. These are key to enabling smooth locomotion, coordinated movement between modules, and safe operation in unstructured environments.

The actuation of the robot's joints is achieved using compact DC motors, selected for their simplicity and suitability for modular and reconfigurable architectures. The motor behavior is controlled through voltage or torque commands derived from planned joint trajectories. A significant part of this work involves converting angle-based movement plans into appropriate motor control inputs, ensuring synchronization and directional control across all units. The modularity of the design also allows individual motor timing to be independently managed, enabling the generation of complex gait patterns of different motions.

1.2 Mechanical Structure

The robot is designed with simplicity, modularity, and stability in mind. It consists of two base platforms—referred to as "feet"—each connected to a vertical segment representing the lower leg or "shank." Above each shank is a jointed segment forming the "knee," which connects to the upper legs or "thighs." These two thigh segments are joined by an articulated bridge-like structure, allowing coordinated movement between both sides.

This design results in a robot with two feet, two knees, and a central connecting bridge that mimics the extension and contraction phases of different types of motion. By alternating fixed and mobile states on each leg, the robot simulates peristaltic motion—one leg remains stable while the other advances, then the cycle reverses.

The robot's minimal degrees of freedom, symmetrical geometry, and grounded stance contribute to a lightweight and efficient system. This configuration not only enhances mechanical simplicity and control but also provides a robust platform for torque-controlled joint actuation and future upgrades with sensors or grippers.

1.3 Programming and Software

The simulation and control of the robot were developed using **MATLAB** and **Simscape Multibody**, which offer a powerful platform for modeling, visualizing, and testing mechanical systems with realistic physics. Each type of locomotion was implemented in a separate Simscape model, tailored to test different movement strategies and control approaches. Below are the main motion types developed and simulated:

1.3.1 Leg Over Movement (Jump Over)

In this motion, one leg lifts and moves over the other leg, crossing it from above. This type of movement simulates a stepping-over action, and was tested to evaluate the robot's ability to perform vertical displacement and reposition itself on the other side of a stationary leg.

1.3.2 Pivot Rotation Movement (180-Degree Leg Rotation)

This movement involves rotating one leg 180 degrees around the other, effectively transitioning it from a rear position to a front position. During this movement, the knee and thigh segments articulate while the opposite leg remains grounded, acting as a pivot. This strategy could be useful for reorienting the robot or navigating tight spaces.

1.3.3 Worm-Like Movement (Kinematic Control)

This is the classic crawling motion, inspired by worm locomotion, where one leg stays fixed while the other extends forward through joint rotations. The robot alternates between legs, simulating extension and contraction phases in a predefined sequence using imposed trajectories.

1.3.4 Worm-Like Movement (Torque Control)

In this variation, the same worm-like motion is achieved using torque inputs instead of direct angle commands. Torque profiles are applied to each joint to generate motion, allowing for analysis of dynamics, stability, and energy consumption under physics-based control.

1.3.5 Worm-Like Movement (Motor Control)

Here, DC motors are used to drive the joints, simulating real actuator behavior. This implementation introduces voltage inputs and mechanical-electric interfaces in Simscape to emulate practical motor-driven locomotion, bringing the model closer to a real-world prototype.

1.4 Potential Applications in Construction Sites

The robot's design makes it highly suitable for applications in construction sites, particularly for inspection, monitoring, and maintenance tasks in confined or hazardous areas. Its worm-like locomotion enables it to navigate narrow beams, crawl under structures, and operate in spaces where wheeled or legged robots struggle. The lightweight, symmetrical structure ensures stability and low power consumption, while the modular joints allow for adaptability across various terrains. Its ability to be torque- or motor-controlled opens the door for smart automation and integration with sensors, making it an efficient and versatile tool in dynamic and risky environments like scaffolding, pipelines, and steel frameworks.

1.5 Thesis Structure

The chapters of this thesis are organized as follows:

- **Chapter 2: Background and Related Work** – Reviews existing research on climbing robots and various locomotion strategies relevant to this project.
- **Chapter 3: Types of Motions** – Describes the different motion types developed for the robot and the mechanics behind each.
- **Chapter 4: Motor** – Discusses the motors used, their specifications, and how they control the robot's joint movement.
- **Chapter 5: Code and Simscape** – Details the simulation models built in Simscape for visualizing and testing each motion.
- **Chapter 6: Results** – Presents the outcomes of the simulations and evaluates the performance of each movement type.
- **Chapter 7: Conclusion and Future Work** – Summarizes the findings and proposes future developments to enhance the robot's capabilities.

Chapter 2

Background and Related Work

2.1 Related Work

Several two-legged robots demonstrate structural concepts closely aligned with the proposed robot, particularly in the use of a rigid central "bridge" or torso connecting the legs.

- **MABEL** – Developed at the University of Michigan, MABEL features a two-legged design with spring-loaded knees and hips, all connected to a robust central torso structure. This configuration enables energy-efficient walking and dynamic stability. The mechanical principles behind MABEL, especially its compliant joints and simplified symmetry, are reflected in our robot's leg–knee–bridge configuration [1].

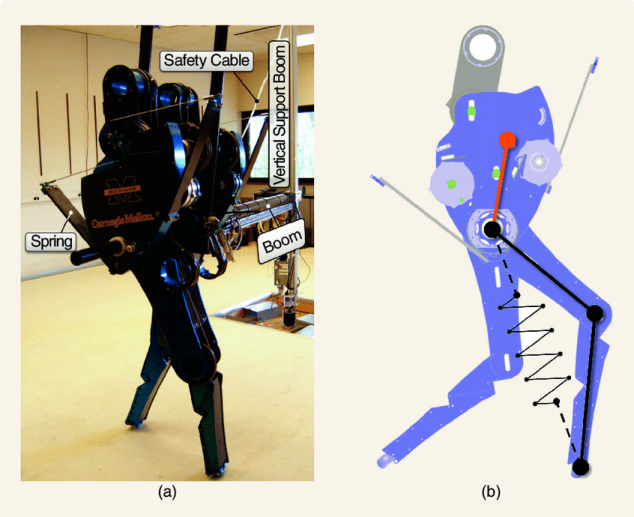

Figure 2.1: Mabel Robot

- **ATRIAS** – ATRIAS is a bird-inspired bipedal robot from Oregon State University, designed with a central body that connects thigh segments and distributes mass evenly. It incorporates spring-mass compliant mechanisms around the knees and ankles to enhance gait transitions. This design supports

the viability of torque-controlled, symmetrical bipedal robots like the one proposed in this thesis [2].

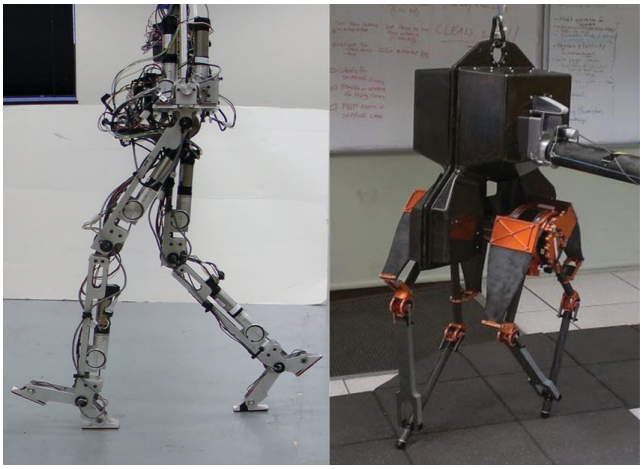

Figure 2.2: Amber 2 Robot

2.2 Related Articles

2.2.1 Robotics and Autonomous Systems

The motion planning strategy implemented in this project is inspired by approaches discussed in a comprehensive survey by Ze et al. [3], which explores locomotion and actuator systems for legged robots. In particular, their treatment of gait planning and foot trajectory generation aligns with the alternating stepping behavior used in our worm-like robot.

A key aspect of our robot's motion is the alternation between support and swing phases, with one leg fixed while the other moves forward. This is conceptually similar to predefined gaits in traditional legged robots. We adopted a cubic interpolation scheme to define foot trajectories in the swing phase:

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{2.1}$$

with the boundary conditions:

$$x(0) = x_0, \quad x(T) = x_T, \quad \dot{x}(0) = \dot{x}(T) = 0$$

Additionally, to maintain balance, the center of mass must lie within the support region defined by the current stance leg:

$$CoM_x \in [x_{support1}, x_{support2}] \tag{2.2}$$

This ensures quasi-static stability during each transition. While energy optimization was not a primary goal in this prototype, the trajectory design could be enhanced in future work using cost functions like:

$$J = \int_0^T \left(\tau(t)^2 + \lambda \cdot \dot{q}(t)^2\right) dt \tag{2.3}$$

These considerations guide the logic for motion timing, stability, and actuation used in our robot's crawling behavior.

In addition to trajectory generation, it is essential to account for deviations in both position and heading when planning paths in cluttered or uncertain environments. This is particularly important for non-circular robots, such as the one developed in this project, which includes a rectangular body and articulated legs.

Ze et al. [3] propose a method for estimating the probability of collision $\hat{p}_c$ that accurately incorporates both shape and orientation uncertainties. Unlike conventional methods that approximate the robot using a bounding circle and rely on the regularized gamma function $\Gamma(n/2, r^2/2)$ to estimate collision likelihood, their model computes this probability directly from the full multivariate Gaussian distribution.

This proves especially effective in situations such as corridor navigation or cornering, where the bounding-circle assumption leads to significant estimation errors due to neglecting heading variations. Figure **??** in their paper shows that their method maintains an average estimation error of only 1.5%, making it highly suitable for reliable and safe navigation of non-circular robots like ours.

$$\hat{p}_c \approx p_c, \quad \text{with average error } < 1.5\% \tag{2.4}$$

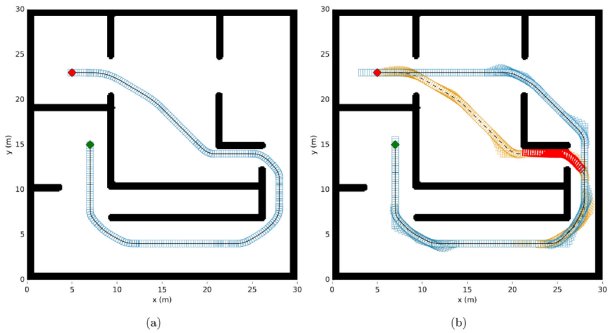

Figure 2.3: Path Planning

2.2.2 Climbing Robots' Mobility for Inspection and Maintenance of 3D Complex Environments

2.2.3 Similarity with ROMA I and ROMA II Climbing Robots

ROMA I and ROMA II are two climbing robots developed by the Robotics Lab at the University Carlos III of Madrid, designed to navigate 3D steel structures such as bridges and building skeletons [4]. Despite being developed for different surfaces and scales, their structural logic and movement mechanisms offer strong parallels to the robot developed in this thesis.

ROMA I features a central body and two extremities (arms), which serve to

anchor and re-anchor the robot during motion. It mimics caterpillar-like locomotion through alternating extension and contraction phases, achieved with a high number of degrees of freedom (8 DOF). This method is structurally and functionally similar to the alternating "worm-like" gait of our robot, in which one foot remains anchored while the other advances, simulating peristaltic movement. ROMA I also demonstrates multi-plane transitions, which validates the concept of using minimal support points to transition between motion and stability—core to our design logic.

ROMA II, a more recent and lightweight iteration, further refines this idea. It uses only 4 DOF and applies the *symmetrical criterion*: a central actuator drives two opposing joints simultaneously. Our robot adopts the same principle by using symmetrical joint trajectories for simplified control, lower weight, and increased mechanical balance. ROMA II's modularity, centralized control, and reduction in actuation complexity mirror our robot's emphasis on minimal DOF and stable, mirrored design.

Both robots use a central "bridge-like" body connecting two anchor platforms—conceptually analogous to the two-legged plus bridge structure of our design. The ROMA platforms use either grasping or suction methods, while our robot uses stable foot contact, but the logic of alternating stability and motion remains consistent.

These parallels confirm the suitability and relevance of symmetrical, alternating locomotion patterns and dual-point anchored motion for navigating constrained or vertical environments.

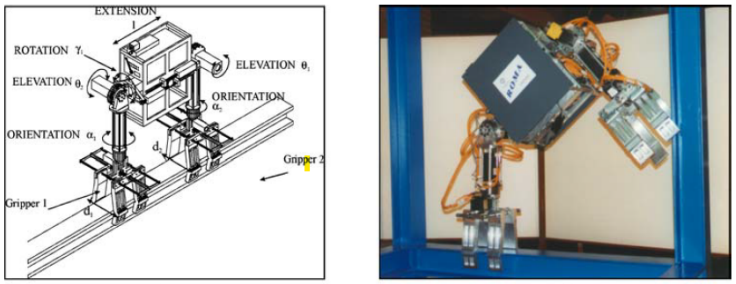

Figure 2.4: ROMA I

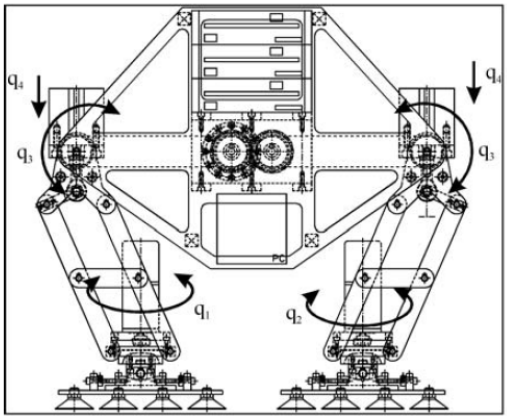

Figure 2.5: ROMA II

Chapter 3

Types of Motion

3.1 Inertial Parameters and Their Role in Simulation

In order to make the simulated motions physically consistent, the robot's main components were assigned mass and inertia properties. These parameters were defined in MATLAB and passed into the Simscape model so that torque, velocity, and energy consumption could be evaluated more realistically.

The following values were used:

- **Foot:** $m = 0.083\,\text{kg}$, $I_{xx} = 0.003$, $I_{yy} = 6.4 \times 10^{-5}$, $I_{zz} = 0.0002$
- **Leg:** $m = 0.812\,\text{kg}$, $I_{xx} = 0.003$, $I_{yy} = 4.8 \times 10^{-4}$, $I_{zz} = 0.0002$
- **Bridge:** $m = 0.055\,\text{kg}$, $I_{xx} = 0.003$, $I_{yy} = 1.73 \times 10^{-4}$, $I_{zz} = 0.0002$
- **Damping and electrical constants:** viscous damping = 0.1, back–EMF constant = 2.5, spatial damping factor = 2000

These values serve several purposes:

1. They ensure that the simulated torques correspond to realistic physical forces required to move the robot's joints.
2. They allow the evaluation of stability and energy efficiency under different gaits.
3. They provide the necessary link between motor voltages and resulting angular accelerations by embedding Newton–Euler dynamics into the simulation.
4. They make it possible to compare results across different motions (jump over, pivot, worm-like) with consistent mechanical properties.

Thus, even though the exact numerical values were simplified for testing, defining approximate inertial parameters was essential to validate whether the proposed gaits are mechanically feasible.

3.1.1 Inertial Parameters and Their Role in Simulation

In order to make the simulated motions physically consistent, the robot's main components were assigned approximate mass and inertia properties. These parameters were defined in MATLAB and passed into the Simscape model so that torque, velocity, and energy consumption could be evaluated more realistically.

Table 3.1: Assigned inertial and damping parameters for the robot components.

Component	**Mass [kg]**	I_{xx}	I_{yy}	I_{zz}
Foot	0.083	0.003	6.4×10^{-5}	0.0002
Leg	0.812	0.003	4.8×10^{-4}	0.0002
Bridge	0.055	0.003	1.73×10^{-4}	0.0002

Additional Parameters	**Value**
Viscous damping (b)	0.1
Back–EMF constant (k_e)	2.5
Spatial damping factor	2000

These values serve several purposes:

1. Ensure that the simulated torques correspond to realistic physical forces required to move the robot's joints.
2. Enable evaluation of stability and energy efficiency under different gaits.
3. Provide the necessary link between motor voltages and resulting angular accelerations by embedding Newton–Euler dynamics into the simulation.
4. Allow fair comparison of results across different motions (jump over, pivot, worm-like) with consistent mechanical properties.

Although simplified, these parameters were essential to validate whether the proposed gaits are mechanically feasible.

3.2 Leg Over Movement (Jump Over)

This motion involves lifting one leg and moving it over the stationary leg, effectively simulating a stepping-over action. It was tested in simulation to evaluate the robot's ability to achieve vertical displacement and reposition itself to the opposite side.

3.2.1 Real-World Applications

Such a motion is particularly beneficial in real-world environments where the robot must traverse over obstacles, such as debris, pipes, or cables. It allows the robot to reposition without requiring body rotation, making it ideal for use in confined industrial or construction spaces.

3.2.2 Mechanical and Control Insights

The leg over motion demonstrates the robot's joint articulation capabilities, especially the flexibility of the knee and thigh joints. It also tests the robot's ability to maintain balance while supported on a single leg. The motion was executed using smooth cubic polynomial trajectories, ensuring stable and realistic movement.

3.2.3 Simulation Purpose

This movement was designed to validate the upper structure of the robot under large joint rotations. It serves as an early test case for evaluating high-amplitude joint movements and synchronization between stationary and moving limbs.

3.2.4 Future Potential

The leg over strategy may be integrated into more complex gaits in the future, enabling adaptive navigation over uneven terrain or use in inspection tasks that require stepping across barriers.

3.2.5 Motion Sequence Visualization

To clearly illustrate the leg over movement, a step-by-step sequence from the simulation is presented below. The images show the progression from the initial stance, through the lifting and crossing phase, and finally to the repositioning of the leg on the opposite side.

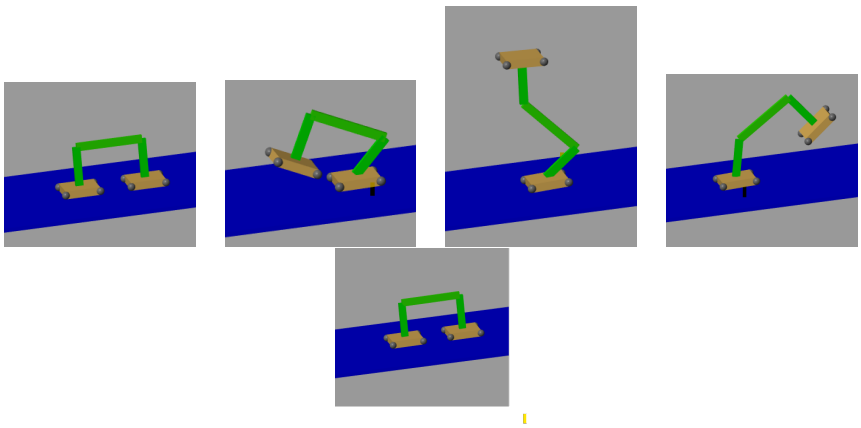

Figure 3.1: Sequential frames showing the execution of the leg over movement.

3.3 Pivot Rotation Movement (180-Degree Leg Rotation)

This movement involves rotating one leg 180 degrees around the other, effectively transitioning it from a rear position to a front position. During this motion, the knee and thigh segments articulate while the opposite leg remains grounded and acts as a pivot point.

3.3.1 Real-World Relevance

The pivot rotation movement is highly useful for reorienting the robot in constrained spaces or when a change in heading is required without lateral translation. It simulates a turning-in-place behavior which could be critical in inspection or maintenance scenarios on scaffolding, narrow walkways, or structural beams.

3.3.2 Technical Insights

This motion helps validate the joint flexibility and structural stability of the robot when only one leg is active. The rotation highlights how well the system handles unbalanced torque and weight distribution, especially during dynamic reorientation.

3.3.3 Motion Sequence Visualization

The following simulation frames illustrate the full execution of the pivot rotation. The moving leg performs a 180-degree sweep to reposition itself while the other leg remains fixed.

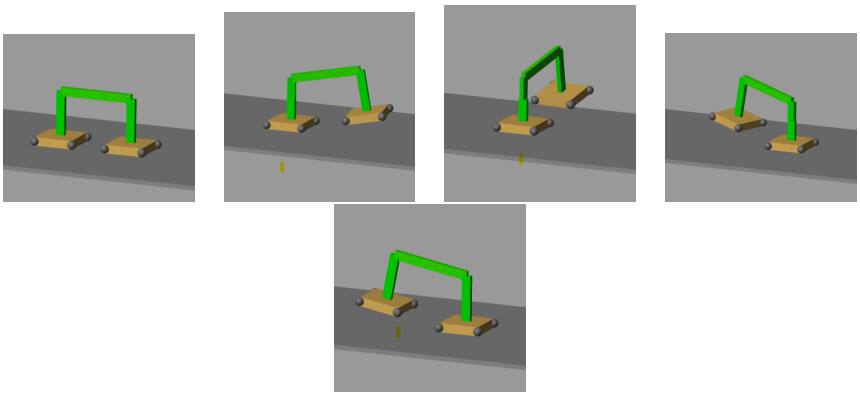

Figure 3.2: Sequential frames showing the execution of the 180-degree pivot rotation movement.

3.4 Worm-Like Movement (Kinematic Control)

This is the classic crawling motion, inspired by worm locomotion, where one leg stays fixed while the other extends forward through joint rotations. The robot alternates between legs, simulating extension and contraction phases in a predefined sequence using imposed joint trajectories.

3.4.1 Real-World Relevance

This crawling strategy allows the robot to navigate linear or confined paths—ideal for pipelines, structural beams, or environments where wheel- or leg-based locomotion is not feasible. It's particularly useful for inspection tasks in tight areas, such as construction sites, ventilation ducts, or collapsed buildings.

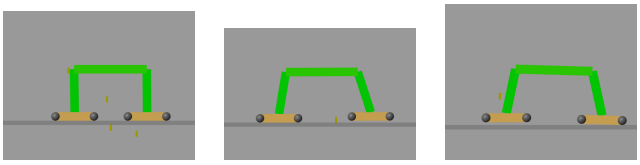

Figure 3.3: Sequential frames showing the kinematic worm-like movement using imposed trajectories.

3.4.2 Control and Motion Strategy

The movement was implemented using **kinematic control**: predefined joint trajectories were generated using cubic polynomial interpolation to drive the robot's joints through each phase. One leg remains stationary, anchoring the system, while the other leg advances through coordinated knee and thigh rotations. The two legs alternate in a looped sequence.

3.4.3 Motion Sequence Visualization

The following simulation frames show the full worm-like movement. The robot begins in a neutral position, then moves the right leg forward while the left leg remains fixed. The pattern is reversed in the following phase to simulate crawling.

Chapter 4

Motor Selection and Control

The motors selected for joint actuation in this robot are crucial to achieving accurate, smooth, and repeatable movement. Two professional-grade options have been considered for future implementation: a **closed-loop NEMA 17 stepper motor** equipped with a magnetic encoder and planetary gearbox, and an **integrated servo joint actuator** featuring a harmonic drive and absolute encoder.

The NEMA 17 stepper motor provides reliable positional accuracy, moderate torque output, and closed-loop feedback correction, making it a strong candidate for lightweight robotic joints. The integrated servo actuator offers superior torque density, minimal backlash, and high control precision, making it ideal for torque-based control and more demanding applications.

Both motors allow for *closed-loop control*, which is essential for preventing missed steps, correcting motion errors in real time, and adapting to variable terrain conditions. In simulation, motors were modeled as ideal actuators driven by either imposed position trajectories or torque commands. In practice, the closed-loop stepper motor can be controlled using step/direction signals with encoder feedback for monitoring, while the servo joint actuator uses advanced protocols such as CANopen or EtherCAT for synchronization and diagnostics.

Ultimately, the selection between these motors depends on specific constraints such as weight, required torque, control architecture, and cost. In future physical implementations, these motors will control the knee and thigh joints of the robot, enabling it to alternate between fixed and mobile legs, execute worm-like crawling motions, and potentially adapt to uneven surfaces through sensor-integrated feedback systems.

Table 4.1: Specifications – NEMA 17 Closed-Loop Stepper Motor

Feature	**Specification**
Motor Type	Stepper, NEMA 17 form factor
Feedback	Magnetic encoder (1000 PPR / 4000 CPR)
Torque (with gearbox)	4.5 Nm (with 10:1 gearbox)
Step Angle	1.8° (0.09° with microstepping)
Control Interface	Step/Direction with Encoder Feedback
Voltage	24V – 48V (typical)
Driver Required	Closed-loop stepper driver
Weight	400–600 g
Use Case	Lightweight robotic joints, low cost, accurate motion

Table 4.2: Specifications – Integrated Servo Joint Actuator (e.g., Tinsmith eRob)

Feature	**Specification**
Motor Type	Brushless DC Servo with Harmonic Drive
Feedback	Absolute encoder (19–20-bit)
Rated Torque	33 Nm (peak torque up to 73 Nm)
Backlash	$< 0.05°$ (harmonic drive)
Control Interface	EtherCAT / CANopen / RS485
Voltage	24V – 48V DC
Built-in Driver	Yes (integrated in the module)
Weight	Varies by model (typically 0.8–1.5 kg)
Use Case	High-torque, low-backlash precision joints

Chapter 5

Code and Simscape

5.1 Overall Robot Model

The robot was modeled and simulated using **Simscape Multibody** in MATLAB/Simulink. The objective was to create a virtual prototype of the worm-like two-legged robot, in which each leg alternates between stability and motion. The control inputs were generated using MATLAB scripts with cubic polynomial interpolation (`cubicpolytraj`), which allowed smooth joint trajectories to be defined in terms of angular displacement, velocity, and acceleration.

5.1.1 Joint and Foot Connections

The subsystem representing the connections between the foot and the leg is shown in Figure 5.1. Each revolute joint (`B-F`) represents a rotational degree of freedom of the foot relative to the leg. The joints are then grouped and linked to the foot block (`Foot3`, `Foot0`), which integrates the outputs of multiple joints into the rigid body structure of the robot.

This configuration allowed the robot to mimic realistic contact between the foot and the ground, as well as maintain rotational flexibility. The grouping blocks (`G-R`) ensure correct transmission of angular motion and constraints between consecutive links.

5.1.2 Trajectory Imposition on Joints

In order to control the robot's motion, joint rotations were imposed as time-dependent trajectories generated in MATLAB. The signals `[tvec, q33]`, `[tvec, q44]`, and `[tvec, q22]` were fed into the revolute joints (`Joint0`, `Joint3`, etc.) to impose the rotation law.

These trajectories are the result of cubic polynomial interpolation and represent smooth angle transitions across simulation time. For example, the `q33` trajectory controls the motion of Joint3 in Leg 1, while `q44` and `q22` correspond to other joints in the opposite leg. This arrangement ensures that one leg remains stable while the other executes the pivoting motion, thereby achieving worm-like locomotion.

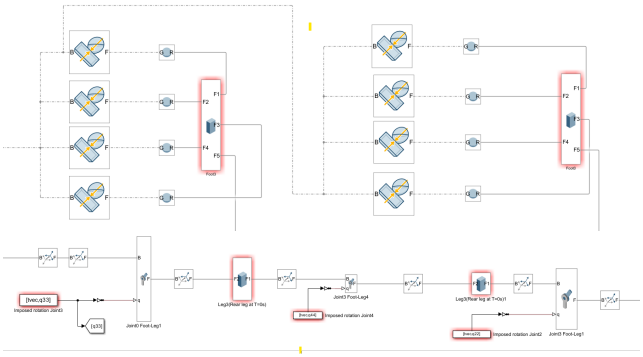

Figure 5.1: Foot–Leg revolute joint subsystem connections (left) and imposed joint trajectories (right). These blocks link the foot with the leg using revolute joints and apply MATLAB-generated signals for controlled motion.

5.1.3 Code Integration

The MATLAB code defined the time vector (`tvec`) and the desired joint angles (`q`) using `cubicpolytraj`. These signals were then passed as inputs to the Simscape revolute joint blocks. This integration allowed testing of both forward and backward motions by reversing the trajectory vectors, as well as experimenting with bidirectional locomotion control.

In summary, this subsystem established the fundamental interaction between MATLAB code and the Simscape physical model. The combination of imposed motion and revolute joint structures enabled the robot to replicate its designed locomotion cycle in simulation.

5.2 From Joint Angles to Motor Torques

Given the desired joint trajectories $\mathbf{q}_d(t)$ generated in MATLAB using `cubicpolytraj` (with piecewise-cubic position, velocity, and acceleration profiles), the control objective is to produce motor torques that drive the simulated joints in Simscape to track these references. The implementation follows a standard computed–torque (inverse–dynamics) structure with PD feedback and viscous/Coulomb friction terms.

5.2.1

ectionSignals from the trajectory generator

For each joint i, the trajectory block outputs

$$q_{d,i}(t), \qquad \dot{q}_{d,i}(t), \qquad \ddot{q}_{d,i}(t)$$

stacked as vectors $\mathbf{q}_d(t)$, $\dot{\mathbf{q}}_d(t)$, $\ddot{\mathbf{q}}_d(t) \in \mathbb{R}^n$ for n joints. The measured (or simulated) joint states are $\mathbf{q}(t), \dot{\mathbf{q}}(t)$, and the tracking errors are

$$\mathbf{e}(t) = \mathbf{q}_d(t) - \mathbf{q}(t), \qquad \dot{\mathbf{e}}(t) = \dot{\mathbf{q}}_d(t) - \dot{\mathbf{q}}(t).$$

5.2.2 Computed–torque (inverse–dynamics) controller

Let $\mathbf{M}(\mathbf{q})$ be the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ the Coriolis/centrifugal matrix, and $\mathbf{g}(\mathbf{q})$ the gravity vector of the leg–rod system. With positive–definite diagonal gains $\mathbf{K}_p, \mathbf{K}_d \in \mathbb{R}^{n\times n}$ and friction parameters $\mathbf{F}_v$ (viscous) and $\mathbf{F}_c$ (Coulomb), the commanded joint torque vector is

$$\boldsymbol{\tau} = \underbrace{\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}}_d}_{\text{feedforward inertia}} + \underbrace{\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_d}_{\text{Coriolis/centrifugal}} + \underbrace{\mathbf{g}(\mathbf{q})}_{\text{gravity}} + \underbrace{\mathbf{K}_d\,\dot{\mathbf{e}} + \mathbf{K}_p\,\mathbf{e}}_{\text{PD feedback}} + \underbrace{\mathbf{F}_v\,\dot{\mathbf{q}} + \mathbf{F}_c\,\operatorname{sgn}(\dot{\mathbf{q}})}_{\text{joint friction}}. \tag{5.1}$$

5.2.3 Saturation and limits

To protect actuators and maintain numerical stability, we apply elementwise torque limits

$$\boldsymbol{\tau}_{\text{cmd}} = \operatorname{sat}\big(\boldsymbol{\tau},\, -\boldsymbol{\tau}_{\max},\, \boldsymbol{\tau}_{\max}\big),$$

and enforce joint position/velocity limits at the Simscape joint blocks.

5.2.4 Mapping joint torque to motor torque and voltage

If a motor with gear ratio N (motor speed to joint speed) and efficiency η drives a joint, the motor shaft torque $\boldsymbol{\tau}_m$ and speed $\boldsymbol{\omega}_m$ relate to joint variables by

$$\boldsymbol{\tau}_m = \frac{\boldsymbol{\tau}_{\text{cmd}}}{\eta\, N}, \qquad \boldsymbol{\omega}_m = N\,\dot{\mathbf{q}}. \tag{5.2}$$

For a DC motor with torque constant k_t, back-EMF constant k_e, and armature resistance R, the electrical dynamics give the drive voltage command

$$\mathbf{V} = R\,\frac{\boldsymbol{\tau}_m}{k_t} + k_e\,\boldsymbol{\omega}_m \;=\; R\,\frac{\boldsymbol{\tau}_{\text{cmd}}}{k_t\,\eta\,N} + k_e\,N\,\dot{\mathbf{q}}. \tag{5.3}$$

Equations (5.1)–(5.3) are the basis of the "angles → torque (and voltage)" conversion used in this work: the trajectory generator supplies $\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d$, the computed–torque law produces the joint torques $\boldsymbol{\tau}_{\text{cmd}}$, and (5.2)–(5.3) translate them into motor–level commands for Simscape actuators.

5.3 Motor Direction Control: From Angle/Torque to CW/CCW Motion

This subsection explains how the joint angle/torque commands are realized as clockwise (CW) and counterclockwise (CCW) motion at the motor using piecewise–constant voltage profiles. In Simulink/Simscape (Fig. 5.2), each joint is driven by a DC motor block. A time–series voltage command $V(t)$ is applied to the motor terminals; its *sign* determines the rotation direction following the right–hand rule of the motor's positive axis.

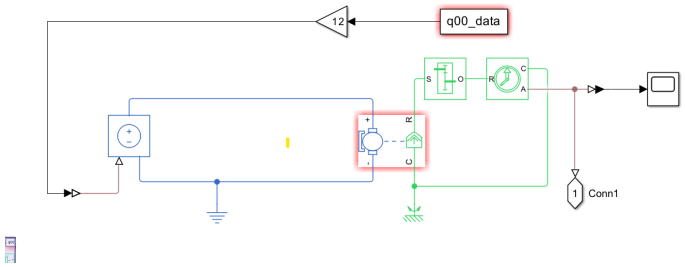

Figure 5.2: Motor drive in Simulink/Simscape. From left to right: voltage source fed by a time–series signal (e.g., `q00_data`), DC motor (electrical `+ -` ports, mechanical `C,R`), sensors, and logging. Positive voltage commands produce positive torque and speed in the motor's positive rotation sense (CCW by Simscape convention for the shown orientation).

5.3.1 DC motor model and direction convention

For the DC motor, the standard electromechanical relations are

$$V = R\,i + L\,\dot{i} + k_e\,\omega, \tag{5.4}$$

$$\tau_m = k_t\,i, \qquad J\,\dot{\omega} + b\,\omega + \tau_{\text{load}} = \tau_m, \tag{5.5}$$

where V is terminal voltage, i the armature current, ω the motor angular speed, τ_m the motor torque, R and L the armature resistance and inductance, k_e and k_t the back–EMF and torque constants, J the rotor inertia, b the viscous friction, and τ_{load} the external load torque (referred to the motor shaft).

From (5.4)–(5.5) (neglecting L and at steady state with $\dot{\omega} = 0$ and $\tau_{\text{load}} = 0$),

$$\omega_{\text{ss}} \;=\; \frac{\left(\frac{k_t}{R}\right) V}{b + \frac{k_t k_e}{R}}, \qquad \tau_m \;=\; \frac{k_t}{R}\,(V - k_e\,\omega). \tag{5.6}$$

Hence, with the usual positive parameters $R, b, k_t, k_e > 0$, the *sign* of the commanded voltage V dictates the sign of both motor torque τ_m and speed ω:

$$V > 0 \Rightarrow \text{CCW (positive) rotation}, \qquad V < 0 \Rightarrow \text{CW (negative) rotation}.$$

If a gear train with ratio N and direction sign $s \in \{+1, -1\}$ is used, the joint variables relate by

$$\dot{q} \;=\; s\,\frac{\omega}{N}, \qquad \tau_{\text{joint}} \;=\; s\,\eta\,N\,\tau_m, \tag{5.7}$$

where η is the efficiency. The sign s is determined by the Simscape gear block orientation; $s = +1$ preserves direction and $s = -1$ inverts it.

5.3.2 Voltage sequences that encode CW/CCW motion

The voltage commands are generated offline in MATLAB as piecewise–constant segments and exported to Simulink as $[t, V(t)]$ arrays. Each motion pattern is defined as a two–column matrix with rows [duration (s), voltage (V)]. For joint "00" (as an

example):

$$\texttt{q00\_motion} = \begin{bmatrix} 1 & 0 \\ 1 & -2 \\ 1 & 0 \\ 1 & 0 \\ 1 & 4 \\ \vdots & \vdots \end{bmatrix}$$

The helper function `build_traj` expands these segments at sampling interval dt to produce a full voltage trajectory:

$$\texttt{q00\_data} = [t_0, V_0;\ t_1, V_1;\ \dots].$$

By construction,

$V(t) > 0 \Rightarrow$ joint drives in the positive (CCW) sense, $V(t) < 0 \Rightarrow$ joint drives in the negative (

The same procedure is applied to the other joints to create `q11_data`, `q22_data`, and `q33_data`. All series are zero–padded to equal length so the simulation stop time is shared.

5.3.3 Simulink implementation (link to Fig. 5.2)

Each $[t, V]$ array is fed to a *From Workspace* or *Signal Builder* block that drives a controlled voltage source connected to the DC motor. The motor's mechanical port is coupled to the joint via the gear (ratio N, sign s). Current and speed sensors close the loop for monitoring or protection. The resulting direction of motion follows (5.6)–(5.7); no additional sign blocks are required if the motor axis and gear sign s are set consistently with the desired joint positive direction.

5.3.4 Practical notes (deadband, limits, back–EMF)

To improve robustness:

- **Voltage limits:** saturate V to $\pm V_{\max}$ of the driver.
- **Deadband:** small voltages may not overcome static friction; enforce $|V| < V_{\text{db}} \Rightarrow V = 0$.
- **Back–EMF compensation:** for near–constant speeds, adding $k_e\, N\, \dot{q}$ to the command (cf. (5.4)) helps maintain direction and speed.

Summary. The sequence matrices (`q00_motion`, `q11_motion`, `q22_motion`, `q33_motion`) encode CW/CCW directly through the sign of the voltage command. Via the DC motor model (5.4)–(5.6) and the gear mapping (5.7), positive voltages yield CCW joint motion (for $s = +1$), and negative voltages yield CW motion, as used to realize the alternating worm–like gait.

Chapter 6

Results

6.1 Flipping Motion (Leg Over Movement)

6.1.1 Joint Angles vs Time

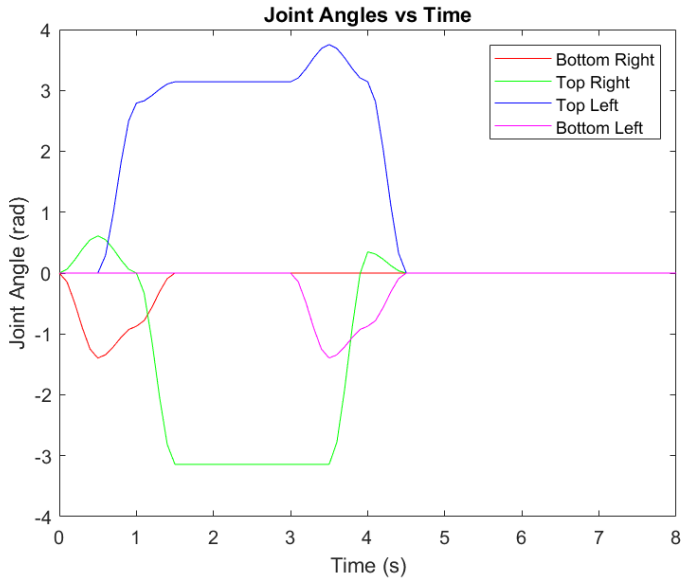

Figure 6.1: Angular displacement of each joint in radians during the flipping (leg over) motion.

Description: This graph tracks the angular displacement of each joint in radians as the motion progresses.
Interpretation: The top left joint experiences the largest positive angular change, approaching around 3.5 radians ($\sim 200°$), indicating a full extension used to push and flip the robot. The top right joint shows a mirrored, negative angular range, confirming that it moves in the opposite direction. The bottom joints (especially the bottom right and bottom left) remain mostly at zero, highlighting their stabilizing role during the flip.

6.1.2 Joint Velocities vs Time

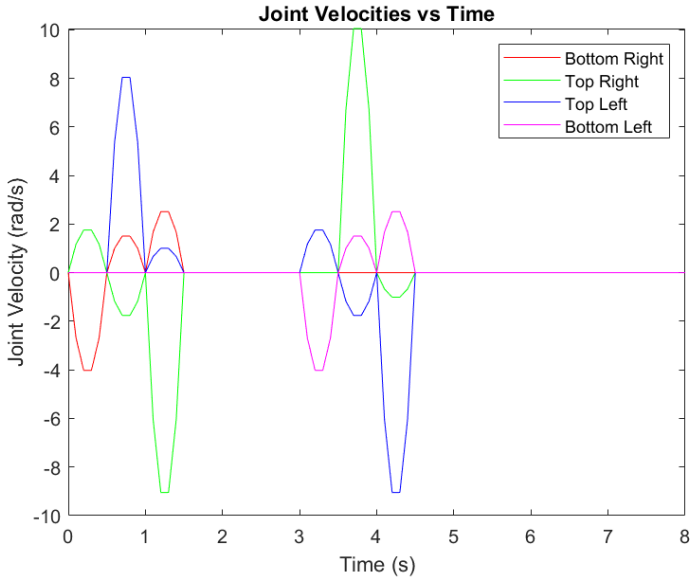

Figure 6.2: Angular velocity of each joint during the flipping (leg over) motion.

Description: This graph shows how fast each joint is rotating, in radians per second. **Interpretation:** Velocity spikes correspond to the moments when joints rapidly switch position to initiate or complete the flip. The top joints again dominate, peaking around $\pm 9\,\text{rad/s}$, while the bottom joints remain mostly static. This reinforces that the flipping motion is driven primarily by the upper part of the structure, with the lower joints providing stability.

6.1.3 Joint Accelerations vs Time

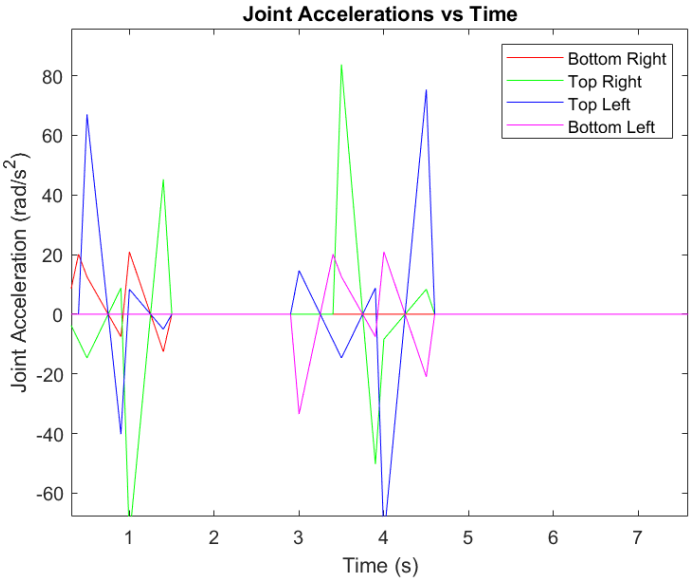

Figure 6.3: Angular acceleration of each joint during the flipping (leg over) motion.

Description: This graph displays how joint accelerations (in rad/s^2) vary over time for the bottom right, top right, top left, and bottom left joints.
Interpretation: Sharp spikes in acceleration appear at the beginning and midpoint of the motion. These peaks reflect the high-torque, rapid rotational efforts required to flip the robot over. The top joints (particularly the top left and top right) show the highest fluctuations, confirming their role in generating the flipping momentum. The bottom joints remain nearly flat, indicating that they serve more as pivot anchors than active movers.

6.1.4 Summary of Flipping Motion Results

These plots confirm that the robot's flipping behavior is highly dependent on rapid and synchronized angular changes in the top joints, while the bottom joints provide grounding and stability. The coordinated role of active upper joints and stabilizing lower joints ensures the feasibility of the leg over (jump over) movement.

6.2 Pivot Motion Results

6.2.1 Joint Angles vs Time

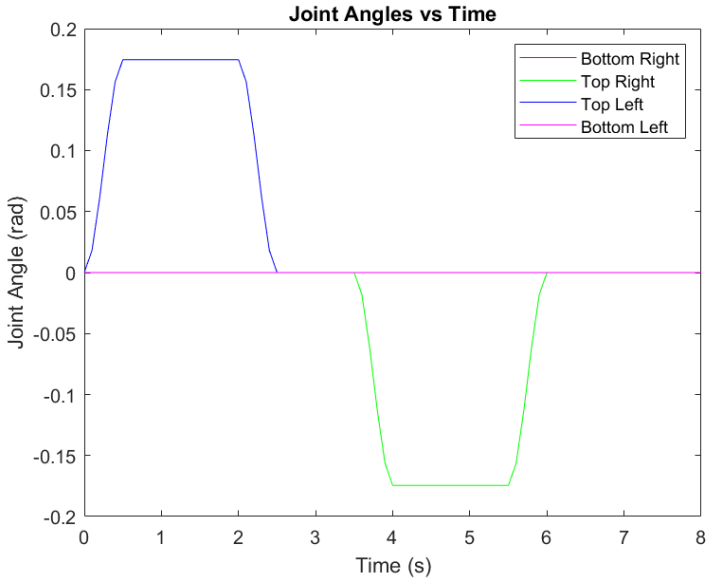

Figure 6.4: Angular displacement of each joint in radians during the pivot motion.

Description: The plot shows how the joints move during the pivot motion, where one side rotates while the other remains anchored.
Interpretation:

- The **Top Left joint** rotates first (~ 0.18 rad), holds, and then returns to zero.
- The **Top Right joint** rotates later in the opposite direction.
- The **Bottom joints** remain fixed (0 rad).

This confirms that the pivot is performed by alternating the top joints while the bottom joints provide stable support.

6.2.2 Joint Velocities vs Time

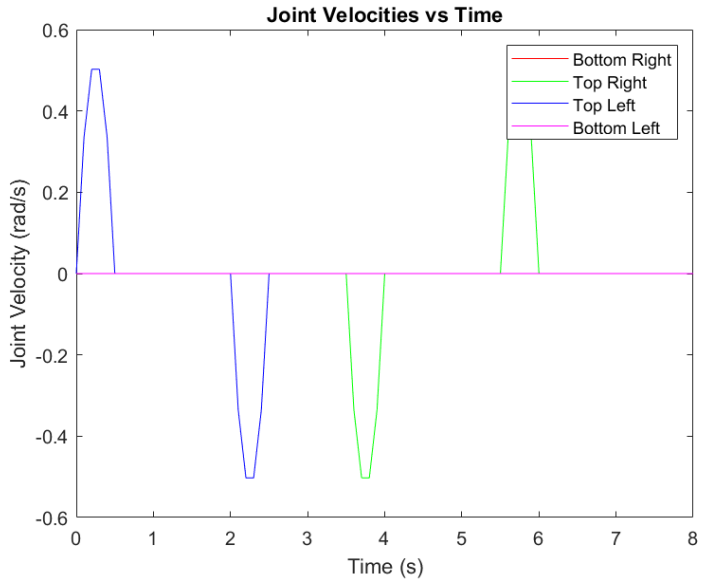

Figure 6.5: Angular velocity of each joint during the pivot motion.

Description: This plot shows the angular velocity of each joint in radians per second during the pivot.
Interpretation:

- **Velocity spikes** occur whenever a joint starts or stops moving.
- The **Top joints** move one after the other, while the bottom joints remain near zero.

This behavior confirms alternating joint activation and controlled reorientation during the pivot motion.

6.2.3 Joint Accelerations vs Time

Description: This graph displays how joint accelerations (in rad/s^2) vary over time for the top and bottom joints during the pivot motion.
Interpretation:

- Sharp spikes appear when rotation starts or stops, reflecting the effort needed to initiate or terminate movement.
- The **Top joints** accelerate in turn, confirming alternating activation.
- The **Bottom joints** remain almost flat, showing they act as stable anchors.

Overall, the motion profile indicates the use of cubic trajectories with smooth transitions.

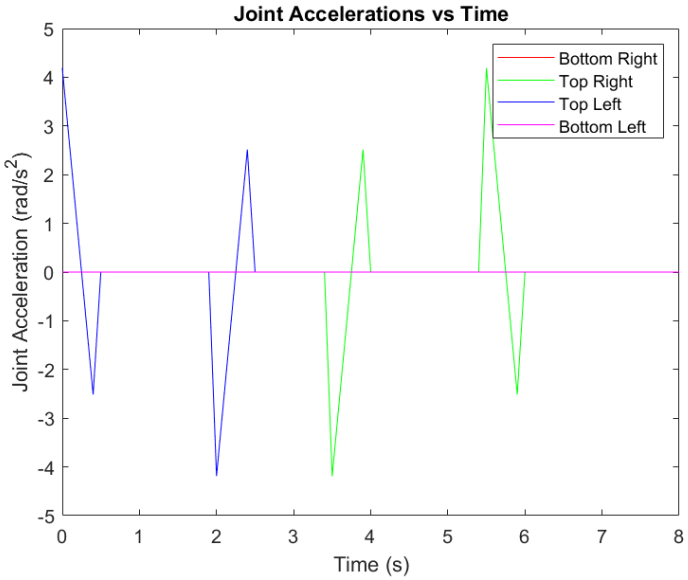

Figure 6.6: Angular acceleration of each joint during the pivot motion.

6.2.4 Summary of Pivot Motion Results

The results show that the pivot motion is both controlled and feasible. The joint angle plots confirm that only the upper joints are active while the bottom joints remain stable. Velocity and acceleration spikes are limited and occur only during switching phases, demonstrating smooth transitions rather than abrupt changes. These findings validate that the pivot motion can be reliably executed with minimal energy waste and sufficient structural stability, making it a practical movement strategy for the robot.

6.3 Worm-Like Motion Results

6.3.1 Joint Angles vs Time

Description: This graph shows the angular displacements of all joints as the robot performs the worm-like motion.
Interpretation:

- Smooth sinusoidal transitions indicate controlled, interpolated motion.
- Top joints move in the opposite direction to bottom joints, showing coordinated limb motion (push–pull alternation).
- The return to zero suggests the robot completes its motion sequence and comes to rest.

6.3.2 Joint Velocities vs Time

Description: This graph shows the angular velocities of all joints during the worm-like motion.

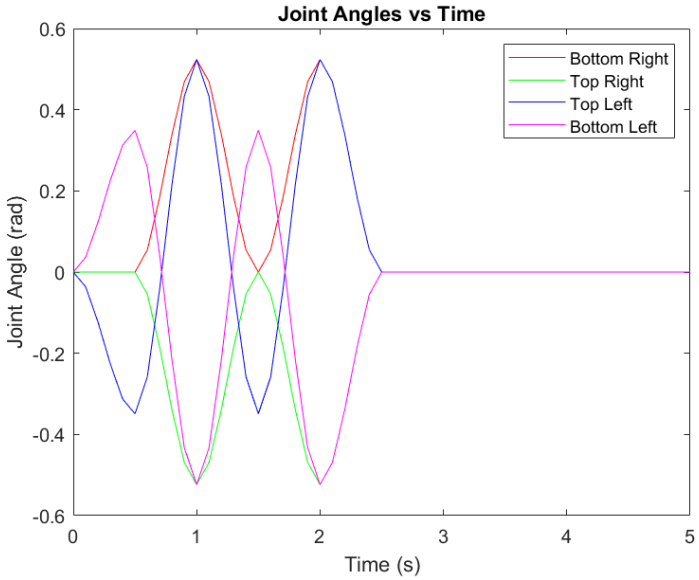

Figure 6.7: Angular displacement of each joint during the worm-like motion.

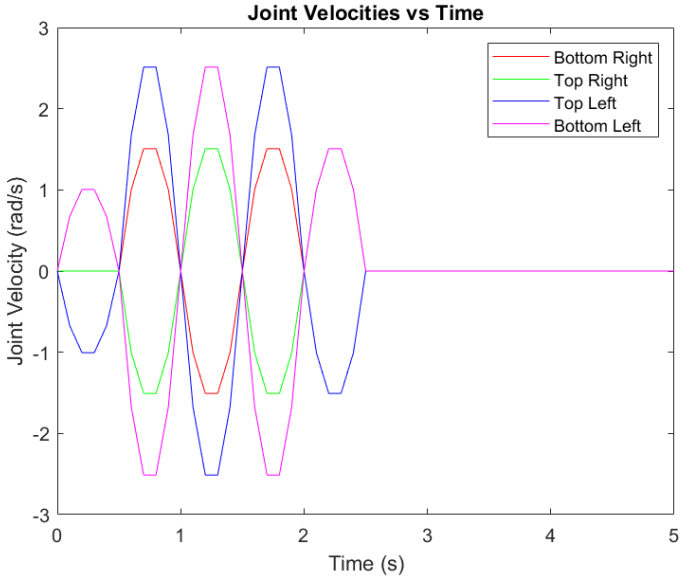

Figure 6.8: Angular velocity of each joint during the worm-like motion.

Interpretation:

- Peaks appear around transitions (e.g., from rest to peak angles).
- The symmetry of the curves indicates a well-balanced gait pattern.
- Sudden flat regions at the end (zero velocity) mean the robot joints stop moving after the motion completes.
- Peaks and troughs correspond to the fastest angular changes during the swing phases.

6.3.3 Joint Accelerations vs Time

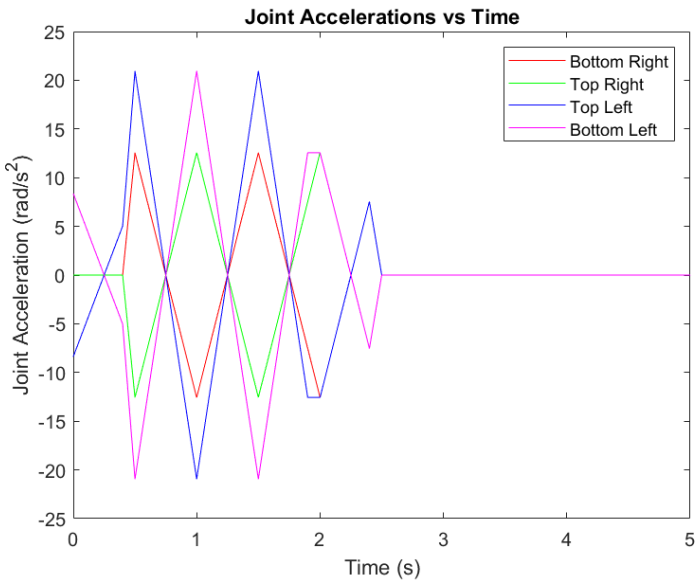

Figure 6.9: Angular acceleration of each joint during the worm-like motion.

Description: This graph reflects how quickly the joints speed up or slow down during movement, measured in rad/s^2.
Interpretation:

- Sharp spikes indicate rapid directional changes.
- Regular alternating peaks and troughs show the rhythm of gait phases (lift, swing, land).
- Accelerations reach up to $\pm 20\,\mathrm{rad/s^2}$, implying quick responsiveness.

6.3.4 Overall Insights on Worm-Like Motion

- The data confirms the robot performs a **cyclic, coordinated movement** — an essential step toward natural locomotion.
- Acceleration and velocity spikes occur at the **transition points** of the motion, as expected.
- The flat tails in all three plots (angles, velocities, accelerations) confirm the robot stops after completing its motion sequence.

6.3.5 Average Torque per Interval

Description: This graph shows the average torque at each joint during the worm-like crawling motion, computed over 0.5-second intervals.
Interpretation:

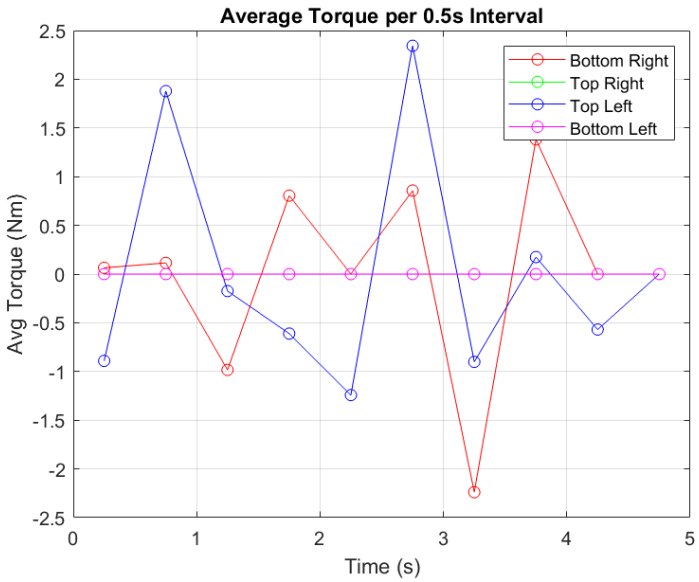

Figure 6.10: Average joint torque values per 0.5 s interval during the worm-like motion.

1. **Top Left (Blue) and Bottom Right (Red) — Alternating Peaks** These joints show dominant, alternating torque patterns, peaking between ±2.5 Nm. This reflects the worm-like "pull–push" strategy:

   - One leg (e.g., left) pulls the robot forward (positive torque).
   - The opposite leg (e.g., right) pushes against the ground or stabilizes (negative torque).

   Their mirrored peaks confirm that while one joint activates, the other relaxes, supporting forward advancement.

2. **Top Right (Green) and Bottom Left (Magenta) — Near-Zero Torques** These joints remain close to zero, meaning:

   - The top right joint was not actively lifting or rotating.
   - The bottom left joint served mainly as a stable support.

   In worm-like locomotion, only selected joints produce torque at each phase, while others anchor the body.

3. **Torque Peaks Around 1 s, 2.5 s, and 3.5 s** These times mark cycle transitions where the robot switches phase (extension vs. contraction). Bursts of torque occur when:

   - A leg is lifted.
   - The body pivots or shifts weight.
   - Force is applied to push forward.

4. **Oscillating Torque Pattern** The alternating up–down torque values every 0.5–1 s show rhythmic, cyclic behavior. This confirms that locomotion is dynamic rather than static, with continuous effort to overcome inertia, gravity, and damping.

5. **Torque Magnitudes are Realistic** The $\pm 2.5\,\mathrm{Nm}$ range is physically reasonable for a small robot with:

   - Link masses below 1 kg.
   - Joint inertias on the order of $10^{-2}\,\mathrm{kg} \cdot \mathrm{m}^2$.
   - Damping between 0.01–0.1 $\mathrm{N} \cdot \mathrm{m} \cdot \mathrm{s/rad}$.

   This ensures the simulated torques remain mechanically feasible without exceeding actuator capacity.

6.3.6 Joint Torques vs Time

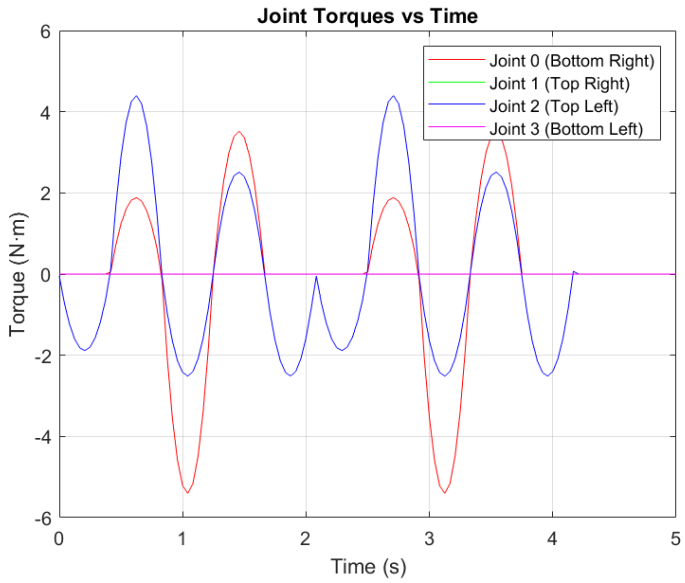

Figure 6.11: Joint torque profiles during the worm-like motion.

Description: This plot shows the instantaneous torque values of each joint over time as the robot performs the worm-like motion.
Interpretation: The torque plot shows that Joint 0 (Bottom Right) and Joint 2 (Top Left) are the primary contributors, generating oscillating torques around $\pm 5\,\mathrm{Nm}$. This is consistent with the alternating push–pull behavior required for worm-like crawling. Their torque waveforms are synchronized but phase-shifted, meaning one joint pushes while the other resets — a classic dual-phase gait pattern.

By contrast, Joint 1 (Top Right) and Joint 3 (Bottom Left) remain close to zero, confirming that they act as passive stabilizers rather than active drivers in this motion cycle.

The torque magnitudes are realistic for small robotic limbs with low inertia, and the smooth sinusoidal shapes align with the expected effort needed to sustain continuous, coordinated locomotion.

6.4 Motor-Level Results for Worm-Like Motion

6.4.1 Cumulative Electrical and Mechanical Energy

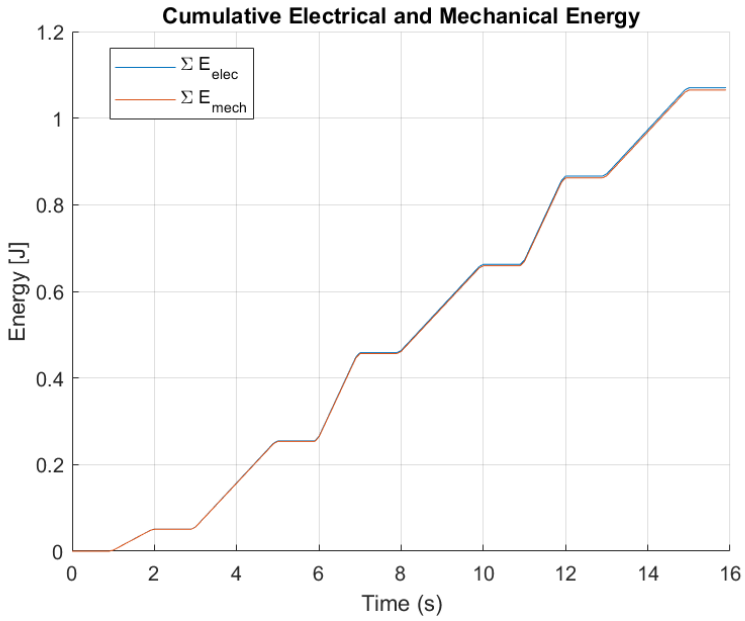

Figure 6.12: Cumulative electrical and mechanical energy during worm-like motion.

Description: This plot shows the total electrical energy supplied to the motors and the corresponding mechanical energy delivered at the shafts, both integrated over time.

Interpretation:

- Both energy curves increase in a **step-like fashion**, reflecting the segmented nature of the gait (1 s of actuation followed by idle phases).
- **Flat regions** indicate idle intervals where no voltage was applied, meaning no further energy consumption.
- The electrical and mechanical energy curves nearly **overlap**, confirming that resistive and frictional losses are minimal under the chosen motor parameters.
- The final cumulative energy (1 J) is realistic for an 8 s worm-like crawl with lightweight links and moderate torques (±2.5 Nm).

6.4.2 Power and Efficiency

Description: The top panel shows the instantaneous electrical and mechanical power, while the bottom panel shows the efficiency profile of the motors during the

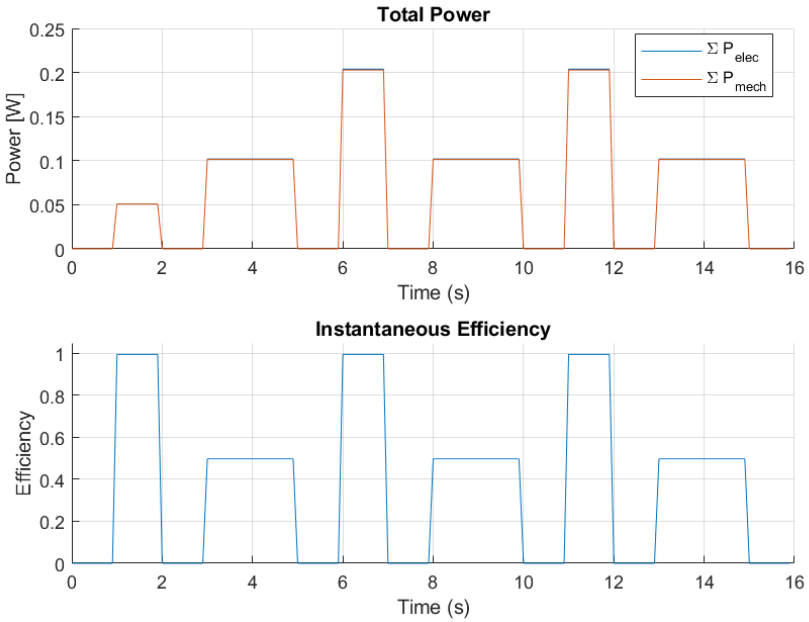

Figure 6.13: Total power consumption and instantaneous efficiency during worm-like motion.

worm-like motion.

Interpretation:

- **Power:**
  - Electrical and mechanical power spikes occur only when voltage is applied, corresponding to push–pull phases of the gait.
  - Flat intervals at zero confirm periods of rest when the legs are idle.
  - Peak powers (0.1–0.2 W) match the expected torque ($\sim$ 2 Nm) and low joint speeds ($\sim 0.05{-}0.1$ rad/s).
- **Efficiency:**
  - Efficiency reaches near **unity (1.0)** during steady movement, consistent with an efficient DC motor model.
  - It temporarily drops to **0–0.5** during stall or low-speed conditions when current flows but little mechanical work is produced.
  - This alternating efficiency pattern is correct for a **stop–and–go gait**: bursts of effective motion followed by resets.

6.4.3 Overall Insights

- The results confirm that the **motor model behavior is consistent** with the worm-like gait: segmented actuation, alternating phases, and rhythmic torque production.

- The close overlap of electrical and mechanical energies shows that **losses are small**, validating the feasibility of the actuation strategy.
- The efficiency plot highlights the difference between active motion and idle phases, proving that energy is only consumed when needed.
- Together, these graphs demonstrate that the chosen motor actuation profile is **physically realistic, energy-efficient, and well suited for worm-like locomotion**.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presented the design, modeling, and simulation of a two-legged worm-like robot capable of executing multiple locomotion strategies, including flipping (leg-over motion), pivot rotation, and worm-like crawling. The robot was developed and tested in MATLAB/Simulink and Simscape environments, where joint trajectories were generated using cubic polynomial interpolation and later converted into torque-based motor commands.

The results confirmed the feasibility of the proposed motions:

- **Flipping motion** demonstrated the ability of the robot to achieve large angular displacements through synchronized top-joint actuation, while bottom joints provided stabilization.
- **Pivot motion** validated the robot's capability to reorient itself in confined spaces by alternating top-joint movements while anchoring the bottom joints.
- **Worm-like crawling** showcased rhythmic, cyclic push–pull joint activations, leading to forward progression with minimal energy expenditure.

At the motor level, analysis of voltage, current, torque, and efficiency confirmed that the chosen actuation strategy is realistic for small-scale robots. The cumulative energy plots and power-efficiency results showed that energy consumption occurs only during active phases, ensuring operational efficiency.

Overall, the research demonstrated that worm-inspired locomotion can be successfully simulated and controlled through torque-driven actuation, validating the initial design concept.

7.2 Future Work

While the results are promising, several directions can further extend this work:

- **Hardware Implementation:** Building a physical prototype to validate the simulated results and evaluate real-world performance under friction, compliance, and terrain variations.
- **Advanced Control Strategies:** Implementing closed-loop feedback using sensors (encoders, IMUs, force sensors) and applying control methods such as PID, model predictive control, or reinforcement learning for adaptive locomotion.
- **Energy Optimization:** Refining motor control to minimize energy losses, incorporating regenerative braking, and testing alternative actuation mechanisms (e.g., pneumatic or tendon-driven systems).
- **Terrain Adaptability:** Extending locomotion to irregular or inclined surfaces, enabling applications in inspection, search-and-rescue, or industrial environments.
- **Scalability:** Investigating multi-segmented worm robots for enhanced mobility, where multiple modules coordinate their motion to cover longer distances efficiently.

This research forms the foundation for worm-inspired robotic locomotion, bridging the gap between bio-inspired concepts and practical robotic systems. By combining simulation, torque-based motor control, and energy analysis, it opens the path toward future development of lightweight, efficient, and adaptive crawling robots.

Appendix A

MATLAB Codes

A.1 Flipping Motion – Trajectory Generation Script

Listing A.1: MATLAB script for flipping (leg-over) motion: cubic polynomial trajectories and inertial/friction parameters.

```
clear all
close all
clc

% Define waypoints and compute cubic polynomial trajectories
wpts00 = [0 -80 -50 0 0 0 0 0 0 0] * pi/180;

num  = length(wpts00)-1;
tpts = 0:0.5:num/2;
tvec = (0:0.1:num+2)';
[q00, qd00, qdd00, ~] = cubicpolytraj(wpts00, tpts, tvec);

wpts11 = [0 35 0 -180 -180 -180 -180 -180 20 0] * pi/180;
[q11, qd11, qdd11, ~] = cubicpolytraj(wpts11, tpts, tvec);

wpts22 = [0 0 160 180 180 180 180 215 180 0] * pi/180;
[q22, qd22, qdd22, ~] = cubicpolytraj(wpts22, tpts, tvec);

wpts33 = [0 0 0 0 0 0 0 -80 -50 0] * pi/180;
[q33, qd33, qdd33, ~] = cubicpolytraj(wpts33, tpts, tvec);

% Initialize mode vector (example: switch sign midway)
mode = zeros(111,1);
mode(50:end) = -1;

% Transpose data for consistency (column vectors)
q00 = q00'; qd00 = qd00'; qdd00 = qdd00';
q11 = q11'; qd11 = qd11'; qdd11 = qdd11';
q22 = q22'; qd22 = qd22'; qdd22 = qdd22';
q33 = q33'; qd33 = qd33'; qdd33 = qdd33';

Tsim  = tpts(end);

```

```
%% Inertial parameters
foot_mass   = 0.083;
foot_Ixx    = 0.003;
foot_Iyy    = 0.000064;
foot_Izz    = 0.0002;

leg_mass    = 0.212;
leg_Ixx     = 0.003;
leg_Iyy     = 0.00048;
leg_Izz     = 0.0002;

bridge_mass = 0.055;
bridge_Ixx  = 0.003;
bridge_Iyy  = 0.000173;
bridge_Izz  = 0.0002;

%% Friction parameter
damping_par = 0.01;
```

A.2 Pivot Motion – Trajectory Generation and Joint Profiles

Listing A.2: MATLAB script for pivot motion (180-degree leg rotation): cubic polynomial trajectories, inertial/friction parameters, and plots of joint responses.

```
clear all
close all
clc

% Define waypoints and compute cubic polynomial trajectories
bottom_right_joint = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] * pi/180;
num  = length(bottom_right_joint)-1;
tpts = 0:0.5:num/2;
tvec = (0:0.1:num+2)';
[q00, qd00, qdd00, ~] = cubicpolytraj(bottom_right_joint, tpts, tvec);

top_right_joint = [0 0 0 0 0 0 0 0 -10 -10 -10 -10 0 0] * pi/180;
[q11, qd11, qdd11, ~] = cubicpolytraj(top_right_joint, tpts, tvec);

top_left_joint = [0 10 10 10 10 0 0 0 0 0 0 0 0 0] * pi/180;
[q22, qd22, qdd22, ~] = cubicpolytraj(top_left_joint, tpts, tvec);

bottom_left_joint = [0 0 0 0 0 0 0 0 0 0 0 0 0 0] * pi/180;
[q33, qd33, qdd33, ~] = cubicpolytraj(bottom_left_joint, tpts, tvec);

mid_left_joint = [0 0 180 180 180 180 180 180 180 180 180 180 180 180]
    * pi/180;
[q44, qd44, qdd44, ~] = cubicpolytraj(mid_left_joint, tpts, tvec);

mid_right_joint = [0 0 0 0 0 0 0 0 0 0 180 180 180 180] * pi/180;
```

```
[q55, qd55, qdd55, ~] = cubicpolytraj(mid_right_joint, tpts, tvec);

% Initialize mode vector
mode = zeros(111,1);
mode(50:end) = -1;

% Transpose data for consistency
q00 = q00'; qd00 = qd00'; qdd00 = qdd00';
q11 = q11'; qd11 = qd11'; qdd11 = qdd11';
q22 = q22'; qd22 = qd22'; qdd22 = qdd22';
q33 = q33'; qd33 = qd33'; qdd33 = qdd33';
q44 = q44'; qd44 = qd44'; qdd44 = qdd44';
q55 = q55'; qd55 = qd55'; qdd55 = qdd55';

Tsim  = tpts(end);

%% Inertial parameters
foot_mass   = 0.083;
foot_Ixx    = 0.003;
foot_Iyy    = 0.000064;
foot_Izz    = 0.0002;

leg_mass    = 0.212;
leg_Ixx     = 0.003;
leg_Iyy     = 0.00048;
leg_Izz     = 0.0002;

bridge_mass = 0.055;
bridge_Ixx  = 0.003;
bridge_Iyy  = 0.000173;
bridge_Izz  = 0.0002;

%% Friction parameter
damping_par = 0.01;

% Plot joint angles
figure;
plot(tvec, q00, 'r', tvec, q11, 'g', tvec, q22, 'b', tvec, q33, 'm');
legend('Bottom Right','Top Right','Top Left','Bottom Left');
xlabel('Time (s)');
ylabel('Joint Angle (rad)');
title('Joint Angles vs Time');
xlim([0 8]);

% Plot joint velocities
figure;
plot(tvec, qd00, 'r', tvec, qd11, 'g', tvec, qd22, 'b', tvec, qd33, 'm
    ');
legend('Bottom Right','Top Right','Top Left','Bottom Left');
xlabel('Time (s)');
ylabel('Joint Velocity (rad/s)');
title('Joint Velocities vs Time');
xlim([0 8]);
```

```
77
78 % Plot joint accelerations
79 figure;
80 plot(tvec, qdd00, 'r', tvec, qdd11, 'g', tvec, qdd22, 'b', tvec, qdd33,
       'm');
81 legend('Bottom Right','Top Right','Top Left','Bottom Left');
82 xlabel('Time (s)');
83 ylabel('Joint Acceleration (rad/s^2)');
84 title('Joint Accelerations vs Time');
85 xlim([0 8]);
```

A.3 Worm-Like Motion – Angle-Based Trajectory Script

Listing A.3: MATLAB script for worm-like motion: forward and reverse angle trajectories combined into a continuous sequence for Simulink.

```
1  clear all
2  close all
3  clc
4
5  %% 1. Define original joint waypoints (forward)
6  bottom_right_joint = [0 5 22 0 5 18 0 0 5 18 0 0 0] * pi/180;
7  top_right_joint    = [0 -5 -22 0 -5 -18 0 0 -5 -18 0 0 0] * pi/180;
8  top_left_joint     = [0 -10 16 0 -10 16 0 0 -10 16 0 0 0] * pi/180;
9  bottom_left_joint  = [0 10 -16 0 10 -16 0 0 10 -16 0 0 0] * pi/180;
10 mid_left_joint     = [0 0 0 0 0 0 0 0 0 0 0 0 0] * pi/180;
11 mid_right_joint    = [0 0 0 0 0 0 0 0 0 0 0 0 0] * pi/180;
12
13 %% 2. Define time vectors
14 num  = length(bottom_right_joint) - 1;
15 tpts = 0:0.5:num/2;        % Waypoints time
16 tvec = (0:0.1:num+2)';     % Full forward time vector
17
18 %% 3. Build reversed trajectories with negated angles
19 bottom_right_joint_rev = -fliplr(bottom_right_joint);
20 top_right_joint_rev    = -fliplr(top_right_joint);
21 top_left_joint_rev     = -fliplr(top_left_joint);
22 bottom_left_joint_rev  = -fliplr(bottom_left_joint);
23 mid_left_joint_rev     = -fliplr(mid_left_joint);
24 mid_right_joint_rev    = -fliplr(mid_right_joint);
25
26 %% 4. Generate cubic polynomial trajectories (forward and reverse)
27 [q00_fwd, ~, ~, ~] = cubicpolytraj(bottom_right_joint, tpts, tvec);
28 [q11_fwd, ~, ~, ~] = cubicpolytraj(top_right_joint, tpts, tvec);
29 [q22_fwd, ~, ~, ~] = cubicpolytraj(top_left_joint, tpts, tvec);
30 [q33_fwd, ~, ~, ~] = cubicpolytraj(bottom_left_joint, tpts, tvec);
31 [q44_fwd, ~, ~, ~] = cubicpolytraj(mid_left_joint, tpts, tvec);
32 [q55_fwd, ~, ~, ~] = cubicpolytraj(mid_right_joint, tpts, tvec);
33
34 [q00_rev, ~, ~, ~] = cubicpolytraj(bottom_right_joint_rev, tpts, tvec);
35 [q11_rev, ~, ~, ~] = cubicpolytraj(top_right_joint_rev, tpts, tvec);
```

```
[q22_rev, ~, ~, ~] = cubicpolytraj(top_left_joint_rev, tpts, tvec);
[q33_rev, ~, ~, ~] = cubicpolytraj(bottom_left_joint_rev, tpts, tvec);
[q44_rev, ~, ~, ~] = cubicpolytraj(mid_left_joint_rev, tpts, tvec);
[q55_rev, ~, ~, ~] = cubicpolytraj(mid_right_joint_rev, tpts, tvec);

%% 5. Combine trajectories
q00_total = [q00_fwd'; q00_rev'];
q11_total = [q11_fwd'; q11_rev'];
q22_total = [q22_fwd'; q22_rev'];
q33_total = [q33_fwd'; q33_rev'];
q44_total = [q44_fwd'; q44_rev'];
q55_total = [q55_fwd'; q55_rev'];

%% 6. Build combined time vector
tvec_total = linspace(0, 17, length(q00_total))';  % ~17 seconds total

%% 7. Format for Simulink (time + values together)
q00_data = [tvec_total, q00_total];
q11_data = [tvec_total, q11_total];
q22_data = [tvec_total, q22_total];
q33_data = [tvec_total, q33_total];
q44_data = [tvec_total, q44_total];
q55_data = [tvec_total, q55_total];

%% 8. Simulation time
Tsim = tvec_total(end);

%% 9. Inertial parameters
foot_mass   = 0.083;
foot_Ixx    = 0.003;
foot_Iyy    = 0.000064;
foot_Izz    = 0.0002;

leg_mass    = 0.212;
leg_Ixx     = 0.003;
leg_Iyy     = 0.00048;
leg_Izz     = 0.0002;

bridge_mass = 0.055;
bridge_Ixx  = 0.003;
bridge_Iyy  = 0.000173;
bridge_Izz  = 0.0002;

%% 10. Friction parameter
damping_par = 0.01;
```

A.4 Worm-Like Motion – Torque-Based Simulation Script

Listing A.4: MATLAB script for worm-like motion: torque computation and interval-averaged torque analysis for Simulink input.

```
clear all
close all
clc

%% Define joint trajectories (in degrees converted to radians)
bottom_right_joint = [0  0   15  -28  0   0   0  15  -28  0   0] * pi
    /180;
top_right_joint    = [0  0  0  0  0  0   0  0  0  0  0] * pi/180;
top_left_joint     = [0 -15 20 0 20  0  -15  20  0  20  0] * pi/180;
bottom_left_joint  = [0 0  0  0  0 0  0  0  0  0  0] * pi/180;

mid_left_joint     = zeros(1, 11);
mid_right_joint    = zeros(1, 11);

%% Timing for trajectory generation
num  = length(bottom_right_joint) - 1;
tpts = 0:0.5:5;              % waypoint time steps
tvec = (0:0.05:6)';          % full time vector for smoother
    interpolation

%% Generate joint trajectories
[q00, qd00, qdd00, ~] = cubicpolytraj(bottom_right_joint, tpts, tvec);
[q11, qd11, qdd11, ~] = cubicpolytraj(top_right_joint,    tpts, tvec);
[q22, qd22, qdd22, ~] = cubicpolytraj(top_left_joint,     tpts, tvec);
[q33, qd33, qdd33, ~] = cubicpolytraj(bottom_left_joint,  tpts, tvec);
[q44, qd44, qdd44, ~] = cubicpolytraj(mid_left_joint,     tpts, tvec);
[q55, qd55, qdd55, ~] = cubicpolytraj(mid_right_joint,    tpts, tvec);

% Time vector matching qdd00
t_trimmed = linspace(tpts(1), tpts(end), size(qdd00, 2))';

%% Inertial parameters
foot_mass   = 0.083;
foot_Ixx    = 0.003;
foot_Iyy    = 0.000064;
foot_Izz    = 0.0002;

leg_mass    = 0.812;
leg_Ixx     = 0.003;
leg_Iyy     = 0.00048;
leg_Izz     = 0.0002;

bridge_mass = 0.055;
bridge_Ixx  = 0.003;
bridge_Iyy  = 0.000173;
bridge_Izz  = 0.0002;

damping_par = 0.06;

%% Compute torques (column vectors)
tau00 = (foot_Izz   * qdd00(1,:) + damping_par * qd00(1,:))';
tau11 = (leg_Izz    * qdd11(1,:) + damping_par * qd11(1,:))';
tau22 = (leg_Izz    * qdd22(1,:) + damping_par * qd22(1,:))';
```

```
tau33 = (foot_Izz    * qdd33(1,:) + damping_par * qd33(1,:))';
tau44 = (bridge_Izz * qdd44(1,:) + damping_par * qd44(1,:))';
tau55 = (bridge_Izz * qdd55(1,:) + damping_par * qd55(1,:))';

% Scaling for realistic torque values
scale = 40;
tau00 = scale * tau00;
tau11 = scale * tau11;
tau22 = scale * tau22;
tau33 = scale * tau33;
tau44 = scale * tau44;
tau55 = scale * tau55;

%% Create [time, torque] matrices for Simulink
torque00_matrix = [t_trimmed, tau00];
torque11_matrix = [t_trimmed, tau11];
torque22_matrix = [t_trimmed, tau22];
torque33_matrix = [t_trimmed, tau33];
torque44_matrix = [t_trimmed, tau44];
torque55_matrix = [t_trimmed, tau55];

%% Plot torque profiles for validation
figure;
plot(t_trimmed, tau00, '-r', t_trimmed, tau11, '-g', ...
     t_trimmed, tau22, '-b', t_trimmed, tau33, '-m');
legend('Joint 0 (Bottom Right)', 'Joint 1 (Top Right)', ...
       'Joint 2 (Top Left)', 'Joint 3 (Bottom Left)');
xlabel('Time (s)'); ylabel('Torque (N m)');
title('Joint Torques vs Time'); grid on;

%% Interval-based average torque computation
dt = 0.5;
t_start = t_trimmed(1);
t_end = t_trimmed(end);
intervals = t_start:dt:t_end;

% Preallocate average torque arrays
avg_tau00 = zeros(length(intervals)-1, 1);
avg_tau11 = zeros(length(intervals)-1, 1);
avg_tau22 = zeros(length(intervals)-1, 1);
avg_tau33 = zeros(length(intervals)-1, 1);

% Compute average torque in each interval
for i = 1:length(intervals)-1
    idx = t_trimmed >= intervals(i) & t_trimmed < intervals(i+1);
    avg_tau00(i) = mean(tau00(idx));
    avg_tau11(i) = mean(tau11(idx));
    avg_tau22(i) = mean(tau22(idx));
    avg_tau33(i) = mean(tau33(idx));
end

% Plot average torques
mid_times = intervals(1:end-1) + dt/2;
```

```
figure;
plot(mid_times, avg_tau00, '-or', mid_times, avg_tau11, '-og', ...
     mid_times, avg_tau22, '-ob', mid_times, avg_tau33, '-om');
legend('Bottom Right','Top Right','Top Left','Bottom Left');
xlabel('Time (s)'); ylabel('Avg Torque (Nm)');
title('Average Torque per 0.5s Interval'); grid on;
```

A.5 Worm-Like Motion – Motor Voltage Input Script

Listing A.5: MATLAB script defining worm-like motion using motor voltage commands, with Simulink-compatible input arrays.

```
clear all
close all
clc

%% 1. Time Vector Setup
T = 8;                  % Total simulation time [s]
dt = 0.1;               % Time step [s]
tvec = (0:dt:T)';       % Time vector column

% Number of steps
N = length(tvec);

%% 2. Define motion patterns for each joint
% Each row: [duration_in_sec, voltage_value]

q00_motion = [...
    1, 0;
    1, -2;
    1, 0;
    1, 0;
    1, 4;
    1, 0;
    1, -4;
    1, 0;
    1, 0;
    1, 4;
    1, 0;
    1, -4;
    1, 0;
    1, 0;
    1, 4;
];

q11_motion = [...
    1, 0;
    1, 2;
    1, 0;
    1, 0;
    1, -4;
```

```
    1, 0;
    1, 4;
    1, 0;
    1, 0;
    1, -4;
    1, 0;
    1, 4;
    1, 0;
    1, 0;
    1, -4;
    1, 0;
];

q22_motion = [...
    1, 0;
    1, 2;
    1, 0;
    1, -4;
    1, 0;
    1, 0;
    1, 4;
    1, 0;
    1, -4;
    1, 0;
    1, 0;
    1, 4;
    1, 0;
    1, -4;
    1, 0;
];

q33_motion = [...
    1, 0;
    1, -2;
    1, 0;
    1, 4;
    1, 0;
    1, 0;
    1, -4;
    1, 0;
    1, 4;
    1, 0;
    1, 0;
    1, -4;
    1, 0;
    1, 4;
    1, 0;
];

%% 3. Define helper function to expand patterns into full trajectory
build_traj = @(pattern, dt) vertcat(...
    cell2mat(arrayfun(@(i) ...
        repmat(pattern(i,2), round(pattern(i,1)/dt), 1), ...
```

```
93          (1:size(pattern,1))', 'UniformOutput', false)) ...
94 );
95
96 %% 4. Generate trajectories for each joint
97 q00_traj = build_traj(q00_motion, dt);
98 q11_traj = build_traj(q11_motion, dt);
99 q22_traj = build_traj(q22_motion, dt);
100 q33_traj = build_traj(q33_motion, dt);
101
102 %% 5. Pad shorter vectors with zeros (to match lengths)
103 max_len = max([length(q00_traj), length(q11_traj), length(q22_traj),
        length(q33_traj)]);
104 q00_traj(end+1:max_len) = 0;
105 q11_traj(end+1:max_len) = 0;
106 q22_traj(end+1:max_len) = 0;
107 q33_traj(end+1:max_len) = 0;
108
109 % Update time vector accordingly
110 tvec = (0:dt:(max_len-1)*dt)';
111
112 %% 6. Create Simulink-compatible [time, value] arrays
113 q00_data = [tvec, q00_traj];
114 q11_data = [tvec, q11_traj];
115 q22_data = [tvec, q22_traj];
116 q33_data = [tvec, q33_traj];
117
118 %% 7. Set simulation stop time
119 Tsim = tvec(end);
120
121 %% 8. Inertial Parameters (unchanged, optional use)
122 foot_mass   = 0.083;
123 foot_Ixx    = 0.003;
124 foot_Iyy    = 0.000064;
125 foot_Izz    = 0.0002;
126
127 leg_mass    = 0.812;
128 leg_Ixx     = 0.003;
129 leg_Iyy     = 0.00048;
130 leg_Izz     = 0.0002;
131
132 bridge_mass = 0.055;
133 bridge_Ixx  = 0.003;
134 bridge_Iyy  = 0.000173;
135 bridge_Izz  = 0.0002;
136
137 damping_par        = 0.1;
138 back_emp_constant  = 2.5;
139 spation_damping    = 2000;
```

Bibliography

[1] Jessy W. Grizzle, Jonathan Hurst, Koushil Sreenath, and Hae-Won Park. "MABEL, a new robotic bipedal walker and runner". In: *Proceedings of the American Control Conference*. University of Michigan. IEEE. 2009, pp. 2030–2036 (cit. on p. 5).

[2] Christian Hubicki, Mason Grimes, Morgan Jones, Daniel Renjewski, Alexander Sprowitz, Alessandro Abate, and Jonathan W. Hurst. "ATRIAS: Design and validation of a tether-free 3D-capable spring-mass bipedal robot". In: *The International Journal of Robotics Research* 35.12 (2016), pp. 1497–1521 (cit. on p. 6).

[3] Xinyu Ze, Bo Yang, Shuai Guo, Zhihong Xiang, Han Xu, and Zhiqiang Deng. "A survey on locomotion systems and actuators in legged robots". In: *Robotics and Autonomous Systems* 108 (2018), pp. 1–12 (cit. on pp. 6, 7).

[4] Carlos Balaguer, Antonio Gimenez, and Alberto Jardon. "Climbing robots' mobility for inspection and maintenance of 3D complex environments". In: *Autonomous Robots* 18.2 (2005), pp. 157–169 (cit. on p. 7).

Dedications

To my beloved family, whose constant love and encouragement gave me the strength to reach this point.

To my professors and mentors, for their guidance and inspiration throughout my academic journey.

And to my friends, who reminded me that perseverance and support make every challenge worthwhile.

This work is dedicated to all of you.