The Pennsylvania State University

The Graduate School

College of Engineering

EFFECTIVENESS OF MID-BODY ARTICULATION FOR IMPROVEMENT OF LARGE MANNED QUADRUPED GAIT STABILITY AND TURNING PERFORMANCE

A Thesis in

Mechanical Engineering

by

Kenneth S. Swidwa

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

August 2017

The thesis of Kenneth Swidwa was reviewed and approved∗ by the following:

H. J. Sommer III

Professor of Mechanical Engineering

Thesis Advisor

Stephen Piazza

Professor of Kinesiology, Mechanical Engineering, and Orthopaedics & Rehabilitation

Mary Frecker

Professor of Mechanical Engineering

Professor-in-Charge of MNE Graduate Programs

∗Signatures are on file in the Graduate School

Abstract

This paper provides a framework for evaluating the efficacy of mid-body articulation for large manned quadruped vehicles. Legged vehicles possess distinct mobility advantages over their wheeled and tracked counterparts in rough off-road terrain (1), a benefit that could be useful in areas such as construction and defense.

As improvements continue to be made in energy storage, actuation, and control, there has been a renewed interest in the use of legged vehicles for applications where wheels will not suffice. Small and medium quadrupeds have been a particularly popular concept, with much effort and attention recently being devoted to companion vehicles capable of transporting supplies.

A primary objective of this paper is to serve as a bridge to allow for the growing base of knowledge related to gait development to be more easily adapted to large quadrupeds by identifying and exploring the unique challenges that will exist with vehicles of this size and purpose. Among these are the increased importance of static stability and the development of gaits which not only meet desired performance metrics but also provide a sufficient level of smoothness and comfort for a manned vehicle.

Through computational development of test gaits and experimental validation, it is shown that the use of an articulated torso provides opportunity for improvement of both straight and turning gait performance. In the case of straight gait, an articulated torso allows for an increase in stability without requiring the entire torso to sway. For turning gaits, an increase in stability and the ability of the torso to better align with a turn allows for turning at tighter radii.

Table of Contents

List of Figures vi

Acknowledgements viii

Chapter 1: Introduction 1

1.1 Background and Significance 1

1.2 Overview of Statically and Dynamically Stable Gaits 4

1.3 Specific Goals and Scope 5

Chapter 2: Review of Previous Work 6

2.1 Statically Stable Gait Analysis 6

Chapter 3: Research Methods 12

3.1 Overview of Research 12

3.2 Kinematic Model of Experimental Robot 15

3.3 Computational Approach 17

3.4 Experimental Setup 21

Chapter 4: Results & Discussion 23

4.1 Straight Gait Stability 23

4.2 Computational Turning Gait 27

Chapter 5: Summary 36

Chapter 6: Conclusions 39

Appendix: System Overview 40

Bibliography 58

List of Figures

Figure 1-1: Boston Dynamics' Legged Squad Support System (LS3)... 2

Figure 1-2: Timberjack hexapod forestry harvester by Plustech (6) 3

Figure 2-1: Boston Dynamics' CHEETAH during dynamic running gait (3). 7

Figure 2-2: Support pattern for statically stable and statically unstable quadruped positions (6). . 8

Figure 2-3: Leg numbering convention and potential optimal gaits from McGhee and Frank (7) 9

Figure 2-4: Support polygons for discontinuous statically stable quadruped gait (2). 10

Figure 2-5: Top down two-phase discontinuous gait diagram for a simple quadruped with massless legs. 11

Figure 3-1: Dynamixel-based quadruped, side view (left) and front view (right)........................ 14

Figure 3-2: Two-dimensional robot model structure (left) used for workspace, foot, and COM determination (right). 16

Figure 3-3: Simplified leg model next to experimental leg. 16

Figure 3-4: Gait testing system diagram.. 18

Figure 3-5: Gait diagram for repeatable, stable turning gait at small radius. Movement of frame along circular trajectory is constrained by leg workspace. 19

Figure 3-6: Turning gait with Ry as active workspace constraint (point of intersection between leg workspace for current phase and desired foot trajectory). 20

Figure 3-7: Desired foot trajectory radii for articulated mid-body (left) and rigid frame (right) during turning gait.. 20

Figure 3-8: Dynamixel robot during torso articulated straight gait. Before leg 4 lift off (top left), during body move phase (top right), before leg 3 lift off (bottom left), and before leg 1 lift off (bottom right)... 22

Figure 4-1: Stability margin results as a function of stride length. Stability margin and stride length reported as fraction of frame length. Results shown for multiple length to width ratios. 23

Figure 4-2: Stability margin as a function of leg to body mass ratio. The left plot shows the stability margin fraction for the rigid frame case and the right shows the stability margin when utilizing mid-body articulation (fixed at 1/8th body length). Both plots are shown for multiple length to width ratios for the frame. 25

Figure 4-3: Stability margin for mid-body displacement of 1/8th frame length, reported as percentage increase over rigid frame stability margin. 26

Figure 4-4: Stability Margin as a function of the amount of mid-body articulation, LWR = 3. 27

Figure 4-5: 90-degree turning gait diagram with inner-outer radius envelope for rigid (left) and articulated (right) mid-body. 28

Figure 4-6: Turning gait results as a function of turning radius, shown for rigid frame, rigid frame with sway, articulated frame, and articulated frame with sway. LBMR of 0.1. 29

Figure 4-7: Turning gait results as a function of turning radius, shown for rigid frame, rigid frame with sway, articulated frame, and articulated frame with sway. LBMR of 0.37 30

Figure 4-8: Minimum turning radius as a function of torso articulation for LWR = 1. 32

Figure 4-9: Minimum turning radius as a function of torso articulation for LWR = 2. 33

Figure 4-10: Minimum turning radius as a function of torso articulation for LWR = 3. 34

Figure 4-11: Minimum radius plot for experimental quadruped 35

Acknowledgements

I would like to thank my advisor, Dr. H. J. Sommer III. His guidance during the challenges of this work have provided me with invaluable experience. I am grateful for the opportunity and for his support and advice.

I would also like to thank Dr. Stephen Piazza for his time and insight while serving as the reader for my thesis.

Chapter 1: Introduction

1.1 Background and Significance

Inspired by the limitations of wheeled and tracked vehicles when traversing difficult terrain, legged vehicles have been a topic of interest and experimentation since the 1940's. Early examples of manned vehicles include the General Electric Walking Truck, Ohio State's Hexapod and John Deere's Timberjack walking machine (2). The distinct mobility advantages over their wheeled and tracked counterparts in rough off-road terrain make legged systems an attractive option. Depending on terrain, wheel or track systems may at best be slower and less efficient if able to travel the landscape at all (1). Such a benefit could be advantageous for industries such as construction, defense, and resource harvesting.

On a smaller scale, there has been considerable interest in the robotic companion vehicle concept for military uses. These are perhaps the most widely known and identifiable quadrupeds today, in part due to the technology demonstrations by Boston Dynamics. These Legged Squad Support Systems (LS3), intended to function as robotic mules, were intended to accompany soldiers on foot to transport supplies and were designed for heavy loads (up to 400lbs of gear) relative to their size (see Figure 1-1). Their mechanical design and intended functionality concentrated much of the weight towards the center of the body, justifying the common assumption of massless legs in existing quadruped research (3).

Many of the issues with these early attempts at walking vehicles have been a mismatch in desired functionality and availability of necessary technology. To a large extent, these challenges still exist, particularly with regards to sufficient energy storage (4). Given the strong drive for technological improvements across a wide variety of fields, however, it is not unreasonable to expect legged vehicles to become increasingly more feasible in the coming years.

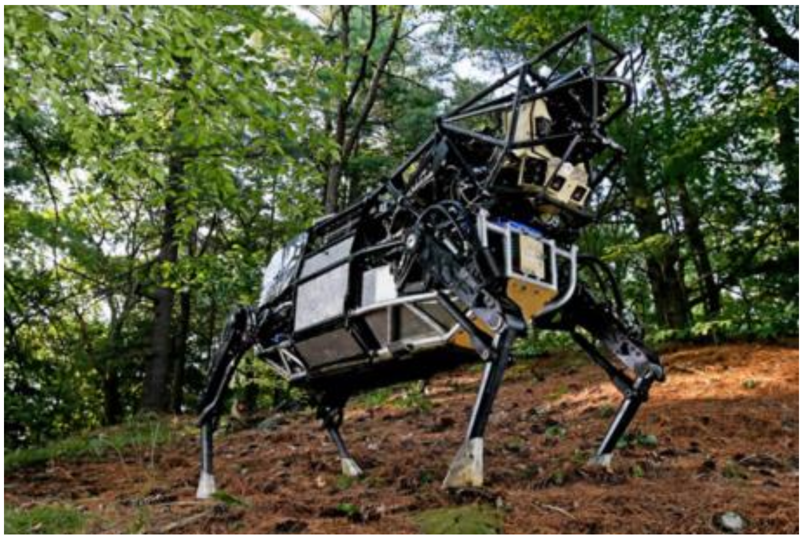

Figure 1-1: Boston Dynamics' Legged Squad Support System (LS3).

Early walker research, particularly before the availability of computer control, was focused on the development of very simple but functional quadruped and hexapod gait. An extreme case is the General Electric's Quadruped, which required manual control of each leg by the driver. While this did allow for impressive demonstrations of the mechanical abilities of the machine, the experience was reportedly too taxing on the human driver and limited its usefulness. It quickly became apparent that automation of the gait would be an essential component of the future of these walkers. The next development, simple electronic state controlled motion, was utilized by Ohio State's Adaptive Suspension Vehicle (ASV). It was "the first computer-coordinated legged system designed to operate in completely unstructured terrain." This hexapod, which unlike the GE quadruped carried its own power source, also carried a driver (5). The biggest change, however, was that the driver was now primarily in charge of navigation rather than coordination of each leg's motion. This separation of control into a system that could be coordinated on a high level led to the realization that the goal of fully autonomous operation could become feasible.

While that autonomy was not realized as a part of the ASV project, advances in computing and control theory have made autonomy possible on a variety of small, dynamic quadruped robots. Boston Dynamics' LS3, one of the largest examples, utilizes computer vision to follow a leader without a driver (3).

However, very little additional work has been done on large legged walkers since those early projects. One more recent example was the Plustech/John Deere Timberjack prototype, a hexapod logging robot designed for use in mountainous environments and highlights the potential usefulness of walkers for resource harvesting (6). For a machine of that size, safety and reliability would be of the highest priority, especially for manned operation. The use of a statically stable gait, which does not need to maintain its motion for stability, would be advantageous to that end.

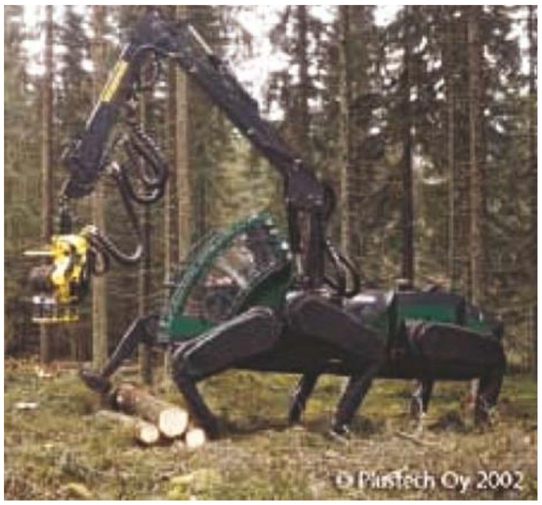

Figure 1-2: Timberjack hexapod forestry harvester by Plustech (6)

1.2 Overview of Statically and Dynamically Stable Gaits

Statically stable gaits were an early point of interest due to limited computational resources and the relative complexity of implementing a dynamically stable gait. While statically stable gaits do allow for simpler actuation timing and control, the reliance on moving a single leg at a time creates many kinematic constraints which can be creatively overcome in a dynamic setting. This allows dynamic gaits to be faster, more versatile, and more agile if allowed by the physical limitations of the actuators being used (2). When considering large vehicles with potentially demanding payload requirements, the necessary speed of actuation may not be feasible. An additional concern is the increased risk in case of actuator or control failure. Depending on the timing of such an incident, the robot would be likely to fall and potentially cause significant damage to itself or the surrounding environment. In the case of a statically stable gait, such a failure would be unlikely to cause as much damage as this gait keeps the robot in a stable position even if motion is unexpectedly stopped (6).

Additional considerations for statically stable gait include whether or not the torso is moved continuously during the gait sequence. For discontinuous gait, the different body and leg move phases must be planned and coordinated. For the simplest rigid torso with massless leg approximation, it is shown that in these cases for a straight gait the stability margin is easily determined (2). Accounting for appreciable leg mass and movement of the mid-body, both as a result of frame sway and mid-body sway, requires more involved analysis for the determination of the quadruped’s stability, especially in turning or circling gaits.

1.3 Specific Goals and Scope

A primary objective in this research was to determine if the addition of mid-body articulation provides sufficient benefit to justify the additional development, complexity, and resources. Assuming a potential for benefit exists, this work is also intended to serve as a framework for developing metrics to assess the extent of the benefits based on desired system build and performance specifications.

Project Goals:

- Develop framework for calculating theoretical stability margin across varying quadruped build and load parameters
- Extend mathematical framework to generate stability-based straight and turning gait for rigid and actuated mid-body
- Utilize framework to quantify potential benefits of mid-body articulation
- Conduct experimental testing on small-scale quadruped to validate simulations

Chapter 2: Review of Previous Work

2.1 Statically Stable Gait Analysis

The unique benefits and challenges associated with robotic legged locomotion have inspired significant research and development since the earliest modern investigations. The methods for studying legged motion are numerous and diverse, and often borrow insight from biological studies of various animal gait. Many of Boston Dynamics robotic products and prototypes show direct or indirect biological influence, with perhaps the most recognizable example being their CHEETAH and WildCat robots. These robots, drawing inspiration from nature, utilize similar galloping and running gaits as well as an articulated back that can pitch to accommodate these higher speeds. These gaits also illustrate the differences between the two fundamentally different types of gait discussed earlier: statically stable gaits and dynamically stable gaits (3).

Legged locomotion is driven by different combinations of leg and body displacements, movements which can occur sequentially or simultaneously. Each of these different combinations of timing and sequence are referred to as a robot's or animal's gait. Much of the recent work has focused on dynamically stable gaits in an effort to push the limits of speed and agility. There has been some interest in torso articulation, though the work has focused primarily on mimicry of spinal motion of animals to increase running speed and efficiency (7). Gaits designed to achieve high speed and maneuverability are almost exclusively dynamically stable gaits and rely on continued motion to provide stability. Figure 2-1 demonstrates these characteristics with time snapshots of CHEETAH's dynamic gait and articulated spine, the use of which allowed the robot to reach speeds in excess of 28 mph (3).

In contrast, statically stable gaits tend to be slower and as such are designed around kinematic assessment of a robot's center of mass, with the assumption being that these gravitational forces dominate those due to the robot's motion (7).

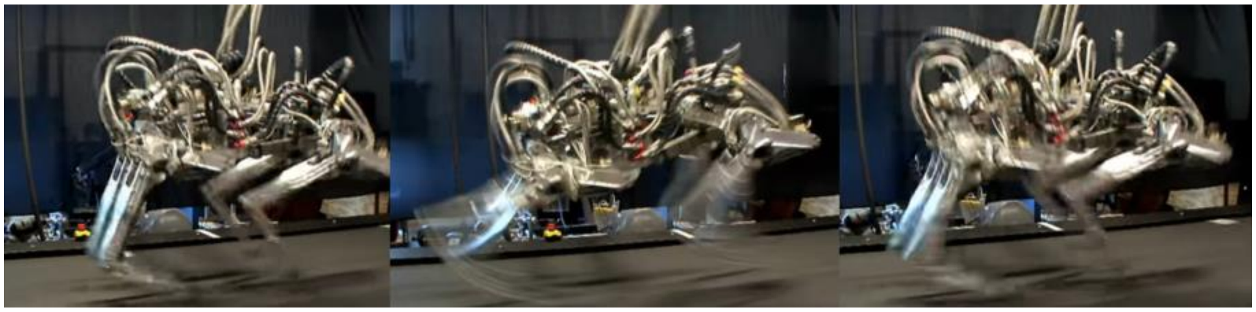
Figure 2-1: Boston Dynamics' CHEETAH during dynamic running gait (3).

Statically stable gaits, or creeping gaits, utilize the idea of a support pattern formed by the feet in contact with the ground. For a quadruped to remain stable, the projection of its center of mass must stay within the planar support pattern at all times during the gait sequence. To satisfy this condition for a quadruped, a creeping gait must always have at least 3 feet in contact with the supporting surface. It is this condition that allows stability to be maintained for arbitrarily slow motion, including complete stops at any time. Support patterns for a statically unstable and statically stable phase are shown in Figure 2-2. Simply satisfying the condition is not necessarily sufficient for real world systems, however, as dynamic effects and other disturbances are not considered. To provide some quantifiable metric with regards to the amount of stability, a stability margin is calculated based on the distance away from instability (7).

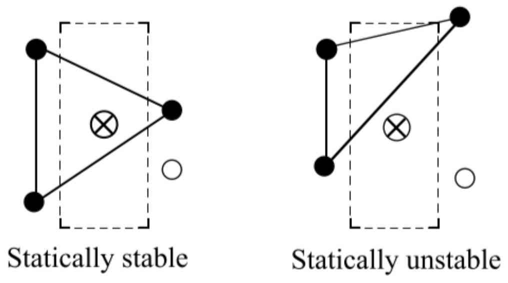

Figure 2-2: Support pattern for statically stable and statically unstable quadruped positions (6).

The magnitude of the stability margin is defined in the early work by McGhee and Frank (7) as being the shortest distance from the center of mass to any point on the supporting surface. For a stable gait pattern, the center of mass is contained within the supporting surface, and the stability margin is positive. Of all the theoretically possible creep gaits, their work focused on identifying and comparing the most stable leg orderings. While many combinations exist, six were realized as potential options and studied in the work. These six gaits, as well as the leg numbering conventions they used are included in Figure 2-3. This number convention has been the standard for quadruped descriptions and will be followed in this paper.

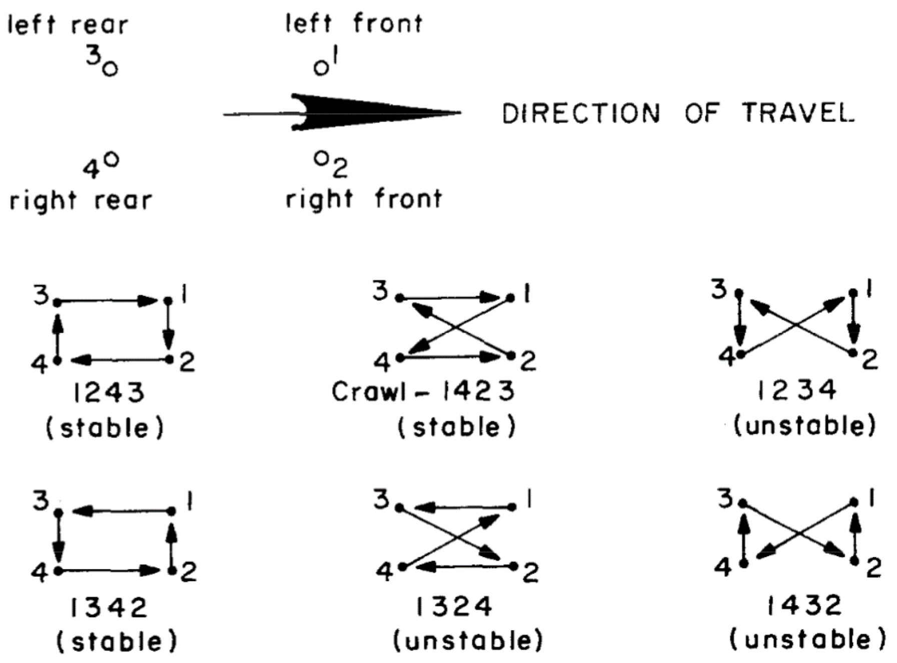

Figure 2-3: Leg numbering convention and potential optimal gaits from McGhee and Frank (7)

One outcome of their work was the determination that the 1-4-2-3 gait was the most stable, which also happens to be the slow speed gait used by quadrupeds and humans. This gait is referred to as crawl, and the terms crawl and creep are often used interchangeably for robotic quadrupeds (7). A key benefit of these creeping gaits is the ability to execute it arbitrarily slowly, including when coming to a stop at any point in the sequence. Zhang and Song (4) extended the work done by McGhee and Frank to study basic turning, also determining that 1243 and 1342 gaits could be used for spinning gaits (rotating about robot z axis).

There are several different types of creeping gaits that can be utilized. Continuous, periodic gaits are the simplest implementation of the creep gait to understand conceptually. In this case, movement of the body and legs occurs simultaneously. For a gait to be periodic, each of the phases of that gait will occur at the same time during the repeatable sequence. The body moves forward continuously and at a fixed velocity relative to the ground, and each leg is lifted and moves forward one at a time for one-quarter of the total leg move time. These were the gaits used by the early works mentioned above, where models were built around an assumption of massless legs (7).

Another option is for the legs and body to move sequentially rather than simultaneously, offering better stability by keeping the center of mass fixed in the position that maximizes stability margin during leg transfer. These gaits are termed discontinuous, and are characterized by alternating phases where the body moves forward with all legs on the ground and where legs move from the rear to the front of their kinematic limits (their workspace). An example gait diagram is shown in Figure 2-4, detailing these different phases.

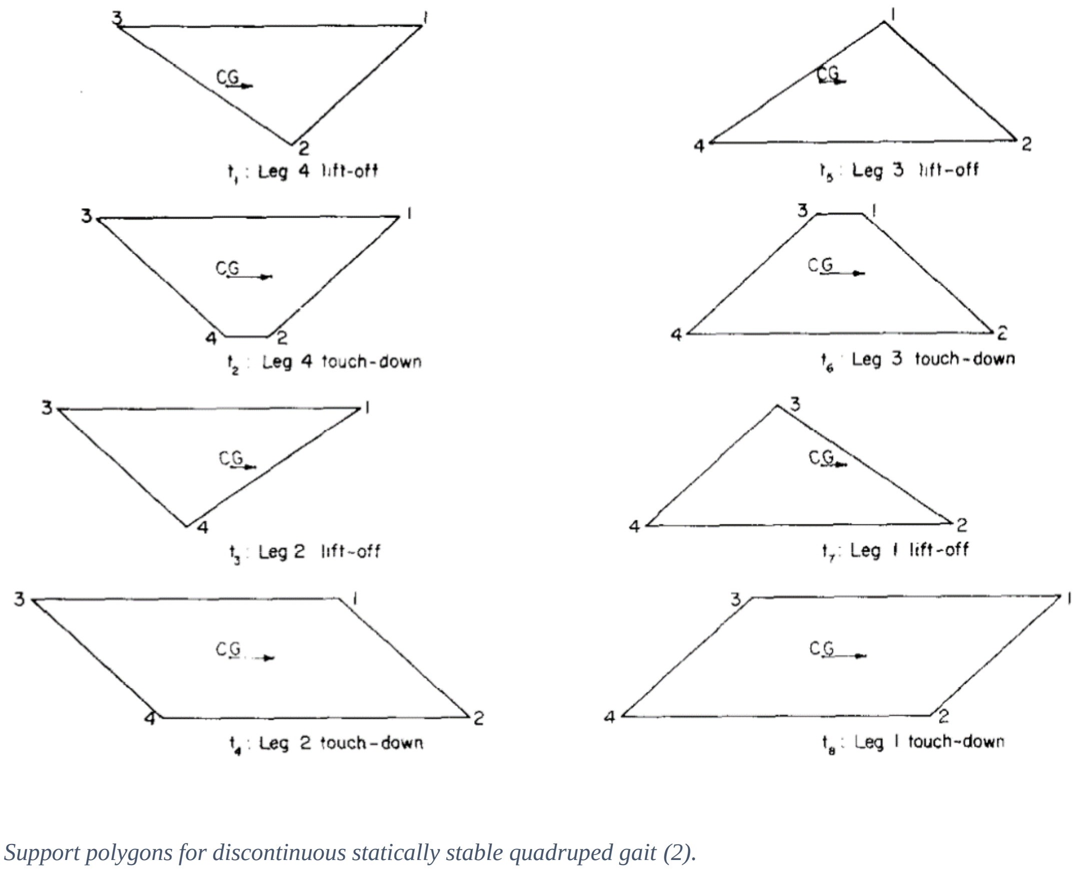

Figure 2-4: Support polygons for discontinuous statically stable quadruped gait (2).

The above example is for a two-phase discontinuous gait, which will be the type of gait used to examine stability changes in this paper. In two phase discontinuous gaits, the body moves by a fixed amount each gait sequence and by half that amount during each body movement phase. These body movement phases occur after both legs on one side of the quadruped move, as depicted in Figure 2-5.

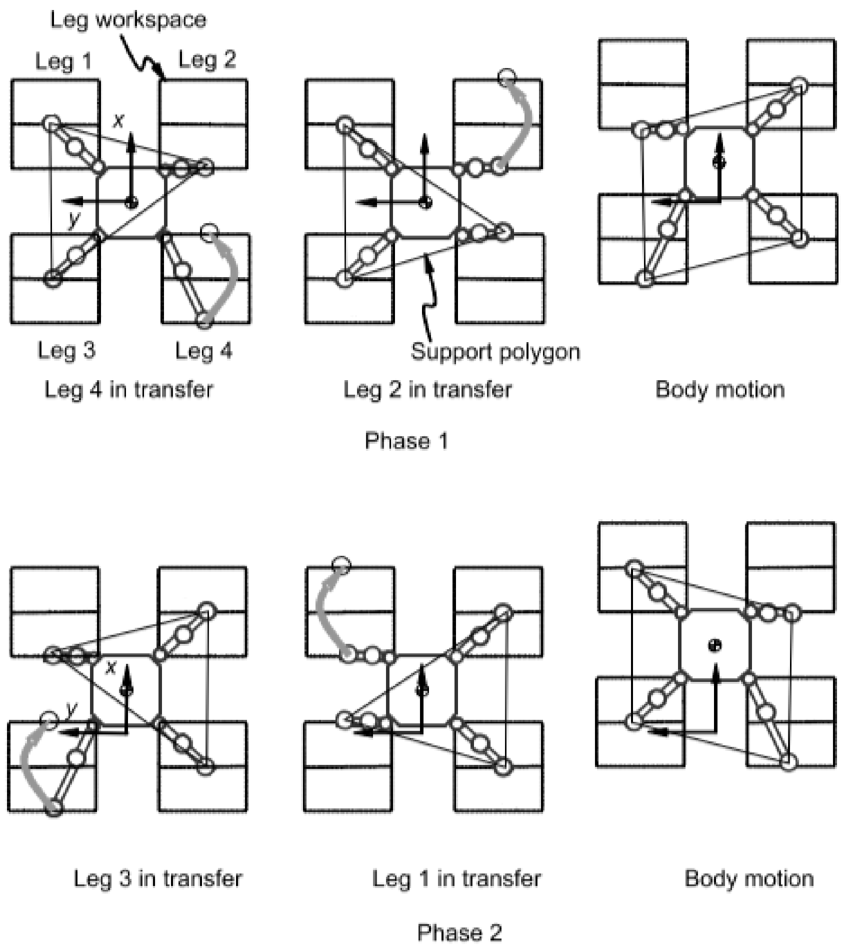

Figure 2-5: Top down two-phase discontinuous gait diagram for a simple quadruped with massless legs.

Chapter 3: Research Methods

3.1 Overview of Research

The strategy for stability assessment and gait comparison between a rigid and an actively articulated mid-body torso consists of both analysis using simulation and verification by experimental testing. The computational framework developed is done in such a way as to be applicable across a range of quadruped build parameters. As such, a key objective is the assessment of how variation across these parameters affects the expected stability margin for different gaits. For computational testing, a robot model with 0.24 m torso length was used to draw comparisons with the experimental build. Given the kinematic nature of the analysis, however, the scale should be generalizable. To capture a variety of physical builds, the following parameters were used and adjusted across testing: Leg to Mass Body Ratio (LMBR), Length to Width Ratio (LWR), Workspace to Body Length Ratio (WSLR), and Maximum Mid-Body Articulation. Across a range of these parameters, straight and turning gaits at different radii were assessed for their stability performance both with and without mid-body articulation and lateral trunk sway. These are two methods of adjusting the position of the center of mass (COM) during gait, with the latter available to builds lacking actuation or a passive joint at the mid-body.

For straight gait, rigid torso performance is compared to performance of a quadruped with a laterally articulated torso and the gait algorithms used were validated on the experimental quadruped. For turning gait, the ability of articulation methods to impact stability margin is assessed. For turning gait, in addition to stability margin impacts, the path radius envelope requirements are also evaluated as a function of the COM manipulation strategies (both mid-body articulation and mid-body sway).

Experimental testing was conducted on a specific build designed to replicate a small-scale model of a large manned transport quadruped. Comparisons in both the computational and experimental settings were drawn from assessments of stability margin for identical trajectories and turning performance (minimum turning radius and turning trajectory path requirements). To achieve this goal, the model was built to incorporate a build with relatively high height to width and length to width ratios for the torso. Another important characteristic which deviates from much existing experimental quadruped work was the use of feet with relatively high contact area. Such a design would be important, both for stability and to minimize potential damage done to walking surfaces by such a large vehicle, but further complicates the development of effective turning gaits. In addition to leg and body trajectory planning, the direction of each leg or of each pair of legs must be deliberately planned and actuated. These additional considerations are not required when feet are instead rounded to a small contact point, as this allows for easy rotation. Smaller, round feet reduce the need for additional actuation but would not be practical for the larger design being considered in this paper.

Dynamixel servos were used for actuation to provide a versatile environment for development which could be easily replicated, modified and expanded in the future. The experimental model consists of 29 servos, seven for each leg and one at the mid-section as shown in Figure 3-1.

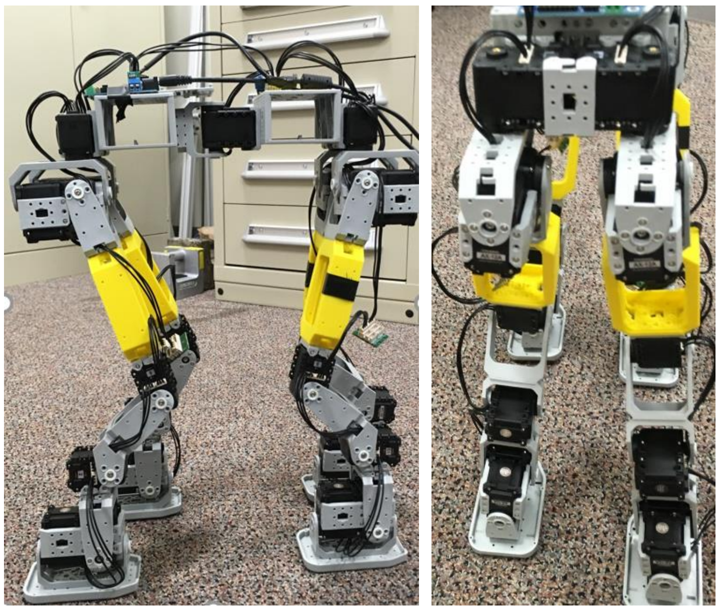

Figure 3-1: Dynamixel-based quadruped, side view (left) and front view (right).

The experimental model is a useful tool to validate the results of the computational analysis. To determine a practical interpretation of the stability margin data, rigid torso tests of straight and turning gaits were conducted with the purpose of identifying a minimum stability margin that allowed for sufficiently accurate and repeatable travel. Stability margin was then manipulated under a variety of testing conditions, including differing turning radii, to assess the extent to which gait performance could be predicted based on the generated stability data. Another metric of interest for turning gait was the radius envelope required for turning, and the results were compared between the computational and experimental tests.

Assumptions:

- Deliberate small-scale kinematic stability analysis will provide insight about kinematic requirements and stability conditions at the larger scale
- Approximate torso as two non-interesting uniformly distributed masses
- Static stability analysis is sufficient approximation for operation at slow speed

- Results and conclusions drawn for statically-stable two-phase discontinuous gait valid for predicting performance in additional gaits of varying complexity

3.2 Kinematic Model of Experimental Robot

The experimental model is notably more complicated than the simple massless leg approximation used for gait comparison, adding complexity in the form of additional joints and legs of nontrivial mass. The frame design also incorporates two body segments connected by an actuator at the mid-body. The active torso articulation allows for assessment of mid-body articulated gaits in addition to rigid torso gaits (locked mid-body actuator). The experimental build represents a specific leg to body mass ratio (LBMR) of 0.37:1 and torso length to width ratio (LWR) of 3:1. In addition to desired performance metrics, these two build-related ratios are used to determine the variations in theoretical testing conditions.

A two-dimensional kinematic model was used to track positions of the frame, feet, mid-body, and COM for gait planning and is depicted in Figure 3-2. Desired trajectories for each phase of the gait are determined based on the structure of the two-dimensional model, the desired/allowable stride, and the stability requirements. To translate to an actionable gait for the robot, the necessary leg and body movements are used to determine the position set points for each leg relative to the corresponding shoulder. These position vectors not only account for the desired foot pattern of the gait, but also correct for the change in shoulder position and orientation as a result of the torso articulation. This data is then converted based on the physical parameters of the leg model to joint commands for actuation control.

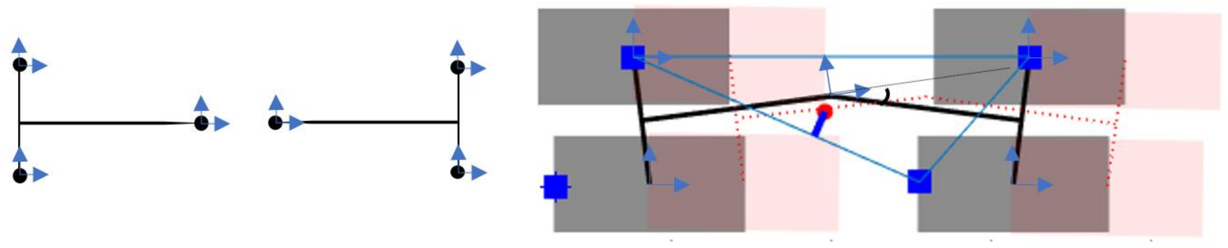

Figure 3-2: Two-dimensional robot model structure (left) used for workspace, foot, and COM determination (right).

Similarly, a simplified kinematic leg model (Figure 3-3) is used to implement position and orientation requests for each foot as actuator commands. The model features 3 degree of freedom (DOF) joint at the shoulder/hip and rotation joints at the knee/elbow and wrist/ankle. The foot also incorporates a 2 DOF joint, allowing for flexion/extension and adduction/abduction.

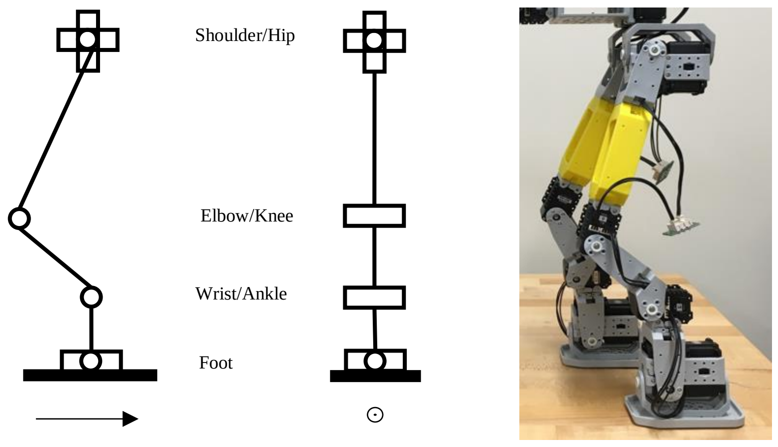

Figure 3-3: Simplified leg model next to experimental leg.

3.3 Computational Approach

To develop and assess potential gait strategies, the two dimensional computational models were controlled and manipulated based on specific configuration and gait objectives. These models were used to numerically assess stability margins across a range of build parameters and gait characteristics. These assessments were conducted both for straight gaits and circling gaits, providing expected performance as a function of body sway, allowable mid-body articulation, leg workspace, length to width ratio, and leg to body mass ratio.

In both computational and experimental evaluations, a two-phase discontinuous gait was used to compare performance across the different testing parameters. The overall testing structure and system organization is illustrated in Figure 3-4 below. To generate test gaits for both numerical and experimental assessment, the kinematic gait planner utilizes the system parameters and desired body centerline trajectory to map out required movements. This is done by creating smooth desired trajectory curves for necessary body and leg movements to satisfy workspace constraints and maintain continuity between gait phases. These paths are followed by generating repeatable steps, and each unique step is assessed for stability to determine the least stable positions.

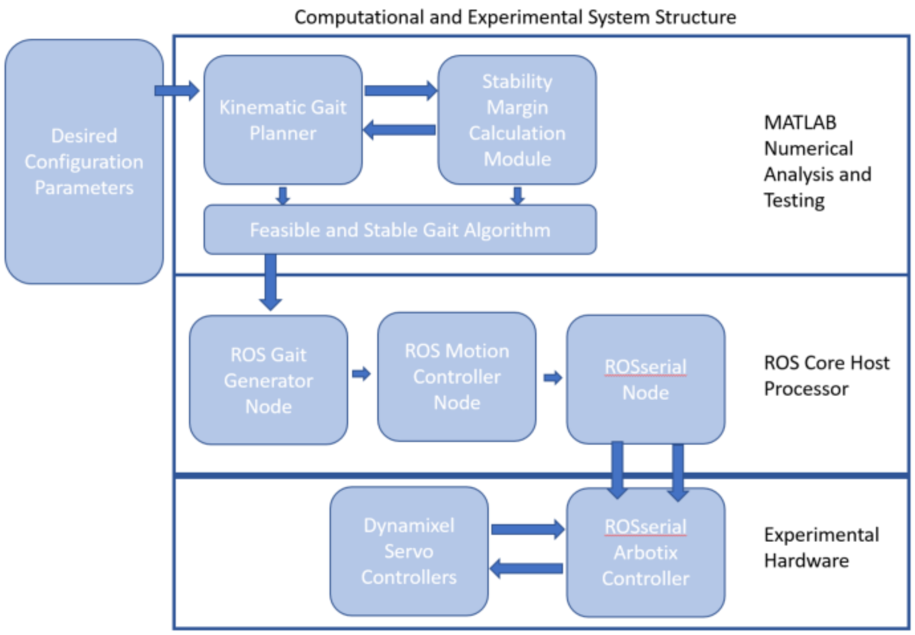
Computational and Experimental System Structure
Desired Configuration Parameters
Kinematic Gait Planner
Stability Margin Calculation Module
Feasible and Stable Gait Algorithm
MATLAB Numerical Analysis and Testing
ROS Gait Generator Node
ROS Motion Controller Node
ROSserial Node
ROS Core Host Processor
Dynamixel Servo Controllers
ROSserial Arbotix Controller
Experimental Hardware

Figure 3-4: Gait testing system diagram

For straight gaits, testing was done across the range of specified configuration parameters. For each configuration set, the stability margin at each phase of the gait was calculated and the lowest margin reported as the margin for that gait. Additional calculations were performed to determine the impact of utilizing mid-body articulation to adjust the location of the COM. Gaits generated with and without mid-body articulation for the experimental build were sent to the gait controller for the robot, and testing was completed to confirm that the computational models were able to create valid gaits.

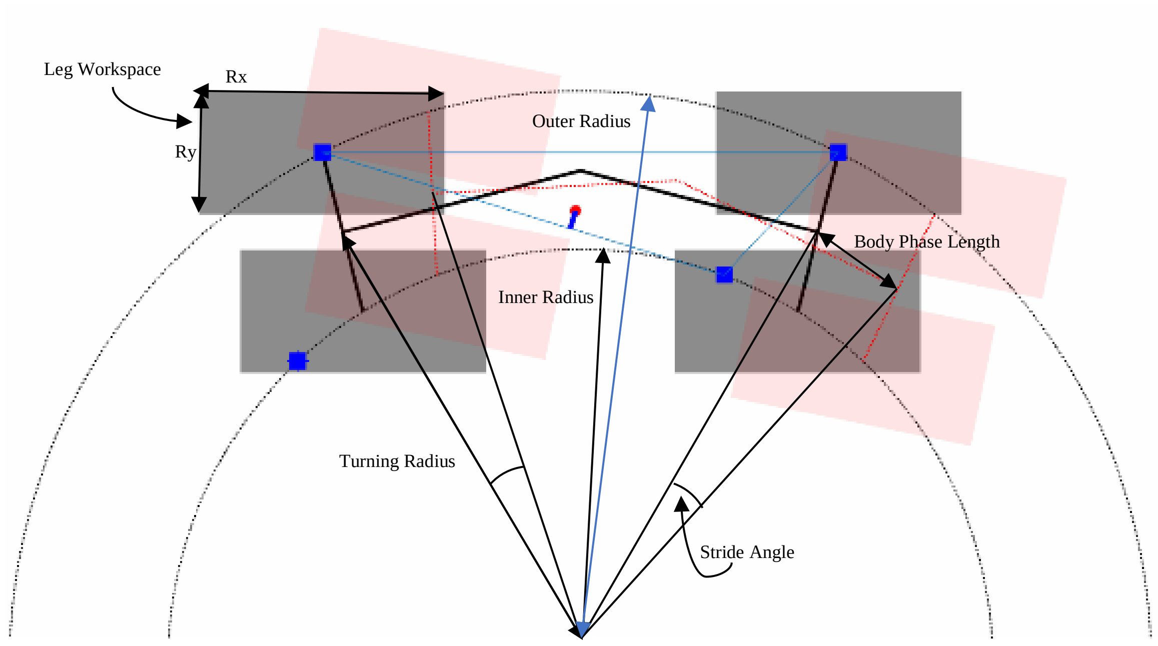

Figure 3-5: Gait diagram for repeatable, stable turning gait at small radius. Movement of frame along circular trajectory is constrained by leg workspace.

For standard turning gaits generated as described, the stability margin during the fore and aft leg movements of each phase are equivalent due to symmetry. The inner phase, however, naturally exceeds that of the outer phase. To achieve the maximum possible stability margin without requiring continued alternating lateral body sway during each phase, the center of mass is adjusted to a radius and build-specific value such that the stability margins for the inner and outer phases are equivalent. For a fixed mid-body system, this COM shift is attempted by shifting the body outward, though this lateral motion is subject to workspace constraints of the leg. For larger radius turning this constraint does not come into effect. However, as turns become tighter the movement is eventually restricted by the lateral component of the leg workspace (Ry) as can be seen in Figure 3-6. As an alternative to mid-body sway, articulation at the mid-body is used to manipulate the COM location. This method allows for the fore and aft frame centers to maintain a fixed radius throughout the duration of the turn, even if the mid-body articulation is adjusted between inner and outer lift phases. Examples of frame orientation and foot placement (desired foot trajectory radii) to maintain repeatable gait are included in Figure 3-7 for both articulated and fixed mid-body.

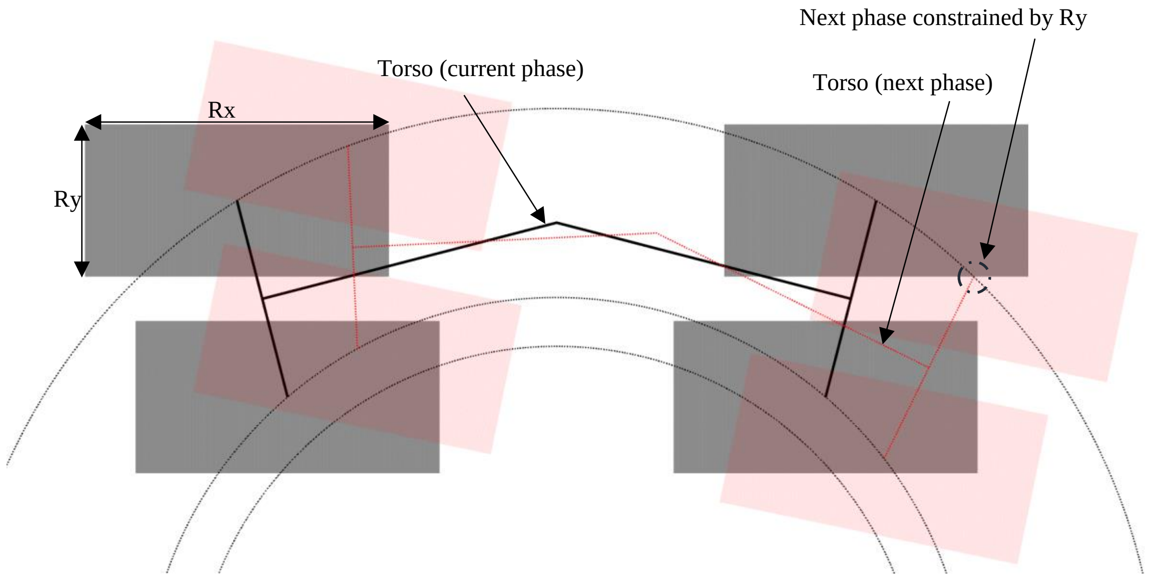

Figure 3-6: Turning gait with Ry as active workspace constraint (point of intersection between leg workspace for current phase and desired foot trajectory).

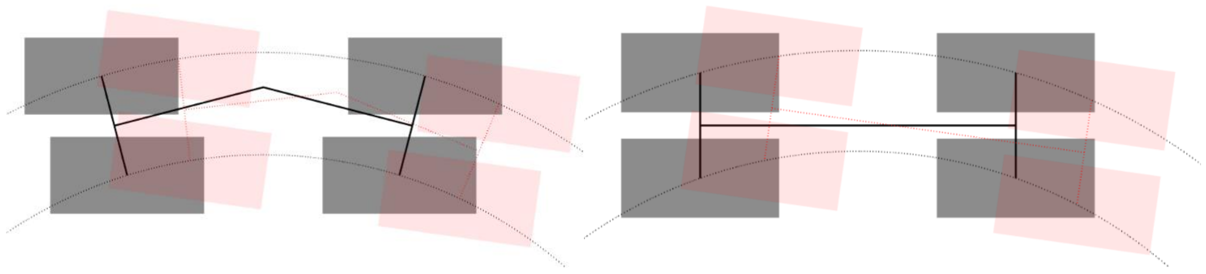

Figure 3-7: Desired foot trajectory radii for articulated mid-body (left) and rigid frame (right) during turning gait.

In the computational turning tests, at each testing radius the following configuration parameters were tested across a specified range: Leg to body mass ratio (LBMR), length to width ratio (LWR), maximum allowable body sway, and maximum allowable articulation angle. Articulated turning performance is assessed based on changes to the stability margin curves, as well as the point at which these curves reach zero. While operation at or near a stability margin of zero may not be stable for a physical system under the influence of disturbances and modeling errors, the stability margin is still useful for comparing relative stability across different gaits.

3.4 Experimental Setup

Experimental testing was conducted on the Dynamixel quadruped model to validate the feasibility of the generated gaits. Initial validation was performed with straight gait, and stability was assessed based on lateral movement and deviation from the desired path to provide a stability metric in addition to gait failure (robot tipping). Figure 3-8 shows the experimental robot during different phases of an articulated straight gait. For turning gait testing, experimental gait tests were performed with 90-degree turns to find the minimum radius achievable based on allowable articulation distance.

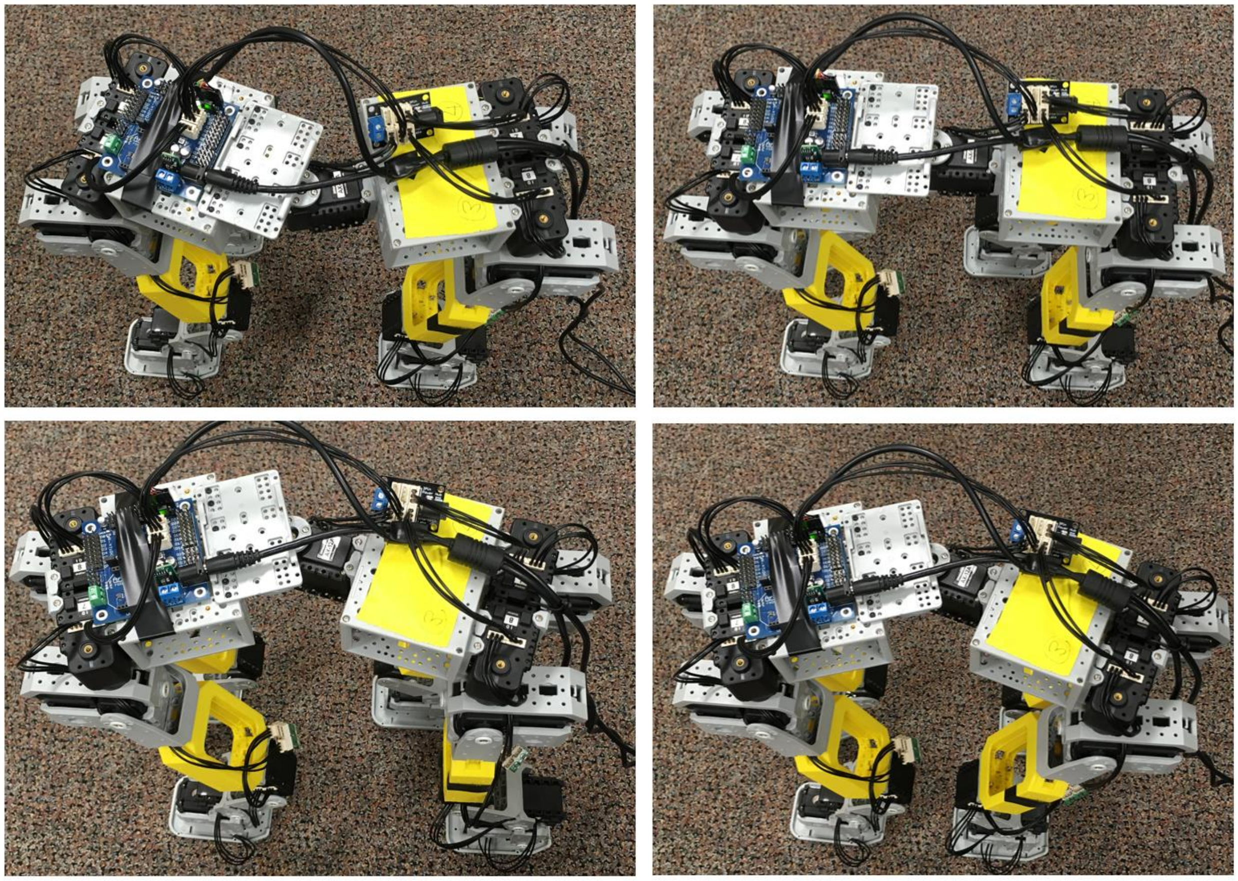

Figure 3-8: Dynamixel robot during torso articulated straight gait. Before leg 4 lift off (top left), during body move phase (top right), before leg 3 lift off (bottom left), and before leg 1 lift off (bottom right).

Chapter 4: Results & Discussion

4.1 Straight Gait Stability

In Pablo (2) the longitudinal stability margin, a simplified stability metric, is examined for a regular and discontinuous gait on a quadruped with massless legs and a 1:1 length to width ratio. It was shown that longitudinal stability margin in this case was a constant dependent only on the stride length. For the true stability margin, measured as the shortest distance between the center of mass and the support polygon, a similar trend is seen under equivalent parameters (LWR = 1, LBMR = 0, Figure 4-1). For scaling to larger systems, all stability margins that follow are presented as a fraction of the frame length. Expanding this analysis to additional length to width ratios shows a more nonlinear but still increasing relationship, and for simplicity, stride length will be fixed to the maximum allowed by the leg workspace.

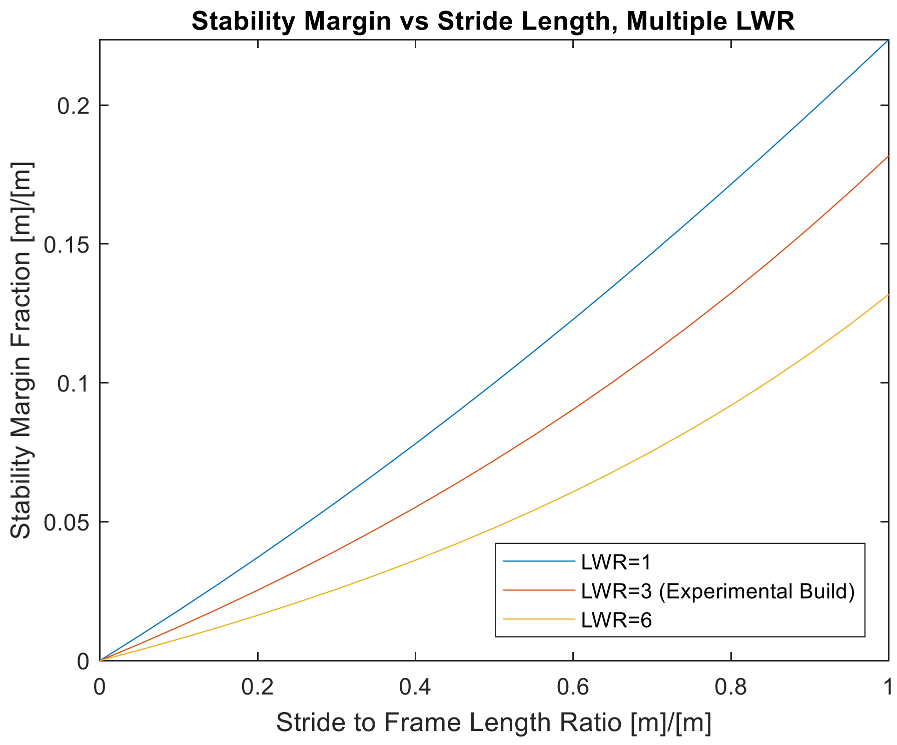

Figure 4-1: Stability margin results as a function of stride length. Stability margin and stride length reported as fraction of frame length. Results shown for multiple length to width ratios.

Holding stride and build fixed to examine stability as a function of leg to body mass ratio (LBMR), an otherwise unmodified center of mass moves away from the center of the frame and approaches but does not reach the support polygon with increasing LBMR. This result is seen for both rigid frame and articulated mid-body (mid-body displacement fixed at 1/8th body length) quadrupeds and across multiple LWR in Figure 4-2. This would be more akin to an unsteady equilibrium, however, and not an ideal operation area. This result is consistent with expected behavior, as in the limit for an increasing LBMR, the mass distribution becomes more similar to a parallelogram with equal mass distributed at each corner. The stability margin calculated is for the gait phase after leg 4 lift-off. From symmetry, in the straight gait this margin is the same as the margin after leg 3 lift-off, before leg 1 touch-down, and before leg 2 touch-down. The phases immediately after leg 3 and leg 4 lift-off have the lowest stability margin for a crawling gait. As such, they are referred to as the Gait Stability Margin (GSM) for each configuration-specific gait that is tested. Moving forward, stability margin and gait stability margin are used interchangeably to mean the lowest stability margin at any point during a gait.

Stability Margin vs Leg to Body Mass Ratio (LBMR)

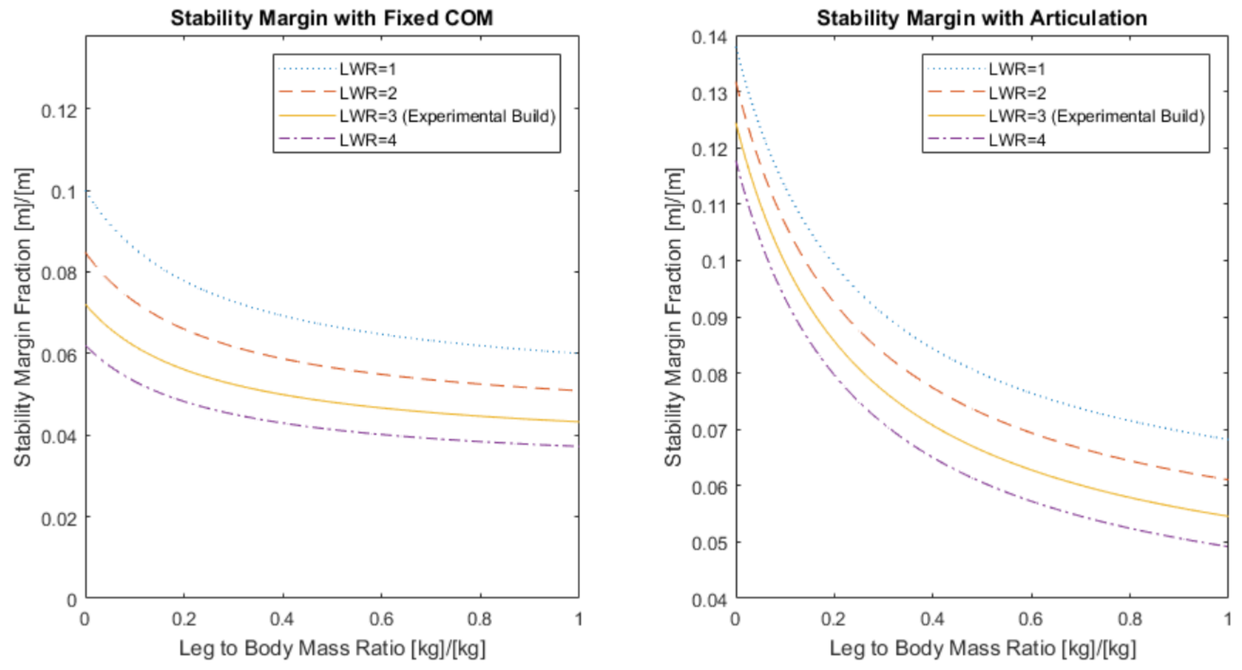

Figure 4-2: Stability margin as a function of leg to body mass ratio. The left plot shows the stability margin fraction for the rigid frame case and the right shows the stability margin when utilizing mid-body articulation (fixed at 1/8th body length). Both plots are shown for multiple length to width ratios for the frame.

In Figure 4-2, the stability margin tangentially approaches a non-zero value because the COM for each leg on the robot is located between the contact point at the foot and the position of the shoulder. Figure 4-3 shows this change in stability from adding articulation as a percentage increase compared to the rigid frame case. Consistently across builds (varying LWR), a change from the ideal case of zero LBMR (massless legs) to 0.2 (each leg 2/10th the mass of the frame) results in a 23% drop in stability margin. A change from the massless leg case to a LBMR of 1 results in a 40% drop, and in the limit increasing LBMR results in roughly a 50% drop in stability margin. As was shown above, stability margin increases with increasing stride length and as such for all tests the stride length is set to the maximum allowable by the workspace.

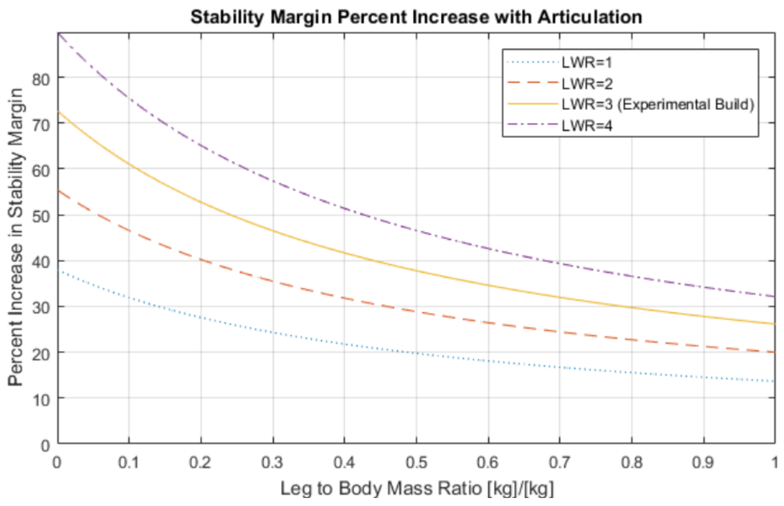

Figure 4-3: Stability margin for mid-body displacement of 1/8th frame length, reported as percentage increase over rigid frame stability margin.

Looking specifically at the experimental frame length and width, Figure 4-4 shows the impact of increasing the allowable mid-body lateral displacement. This relationship is displayed for four specific LBMRs, represented by the different curves. Stability margin is also reported as percentage increase. The experimental build is characterized by a LBMR of approximately 0.37. Straight gait testing was performed on the experimental robot to validate the gaits developed. Articulated straight gaits were tested, however stable straight gait performance was achieved without the need for articulation.

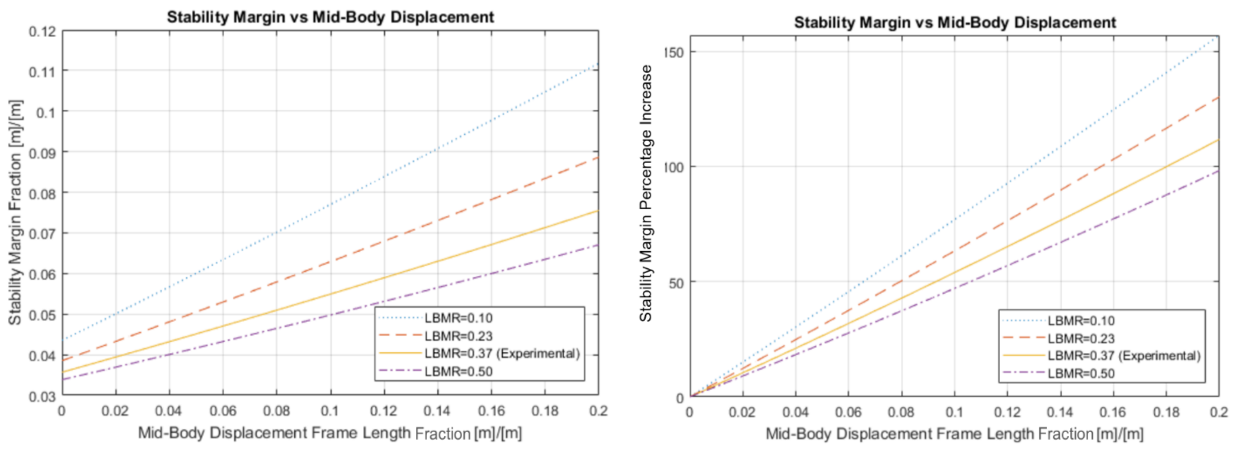

Figure 4-4: Stability Margin as a function of the amount of mid-body articulation, LWR = 3.

4.2 Computational Turning Gait

Assessment of turning gaits was conducted similarly to that of straight gaits, with the differences being that multiple radii were tested. In addition to stability margin, another metric for evaluating turning was the path envelope requirements to complete a turn. This envelope dictates the path or road requirements to allow the quadruped to complete an unobscured 90-degree circling turn, and is shown in Figure 4-5. For an equivalent turning radius, the impacts of articulation on the turning radius envelope proved to be small. As detailed in the results below, however, there is a nontrivial impact when comparing the stability margin for a fixed radius as well as when looking at the minimum achievable radius for a given build configuration.

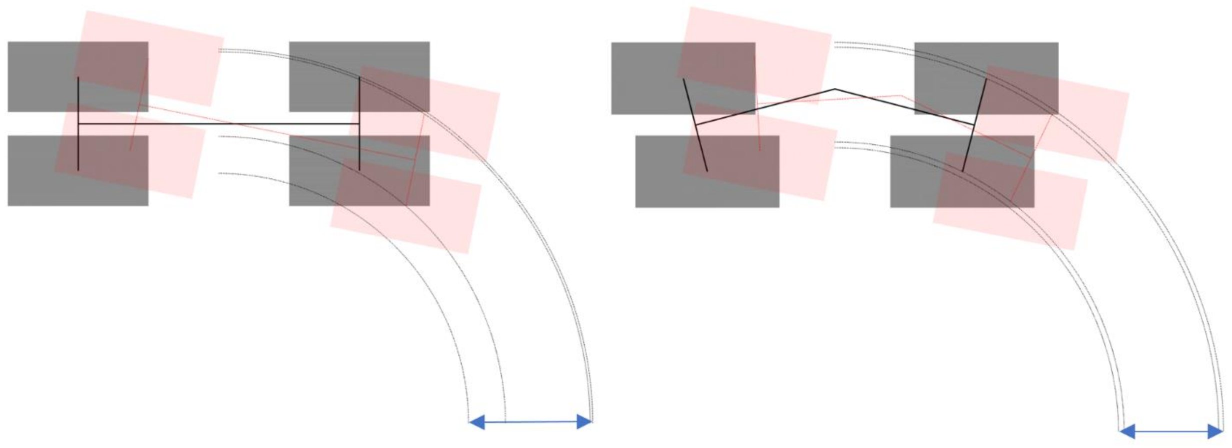

Figure 4-5: 90-degree turning gait diagram with inner-outer radius envelope for rigid (left) and articulated (right) mid-body.

The first set of results displayed below in Figure 4-6 are for a LBMR = 0.1 (each leg 1/10th the mass of the frame) across a range of different widths (LWR = 1,2,3,4). The margin is reported as a fraction of the length of the frame and is the gait margin, or the lowest stability margin of any phase during the specified gait. The plot shows the change in margin as a function of turning radius, as well as the radius at which the gait becomes theoretically unstable. Radii are reported as a fraction of body length. The four plots show the results for rigid body (top left), rigid body with corrective sway (top right), articulated body without sway (bottom left), and articulated body with corrective sway (bottom right). Similarly, the second set of plots show in Figure 4-7 displays these results for a LBMR = 0.37. As the radius decreases, it begins to more closely resemble spinning gait (turning radius/frame length = 0.5) in function and at a certain point it may be more efficient to switch to such a gait. From a stability perspective, it is seen that it is only possible to approach this behavior when utilizing mid-body articulation. Any configuration of build parameters and maximum articulation distance which maintain stability for all turning radii of half the frame length and higher can be considered able to make turns of an arbitrarily small radius.

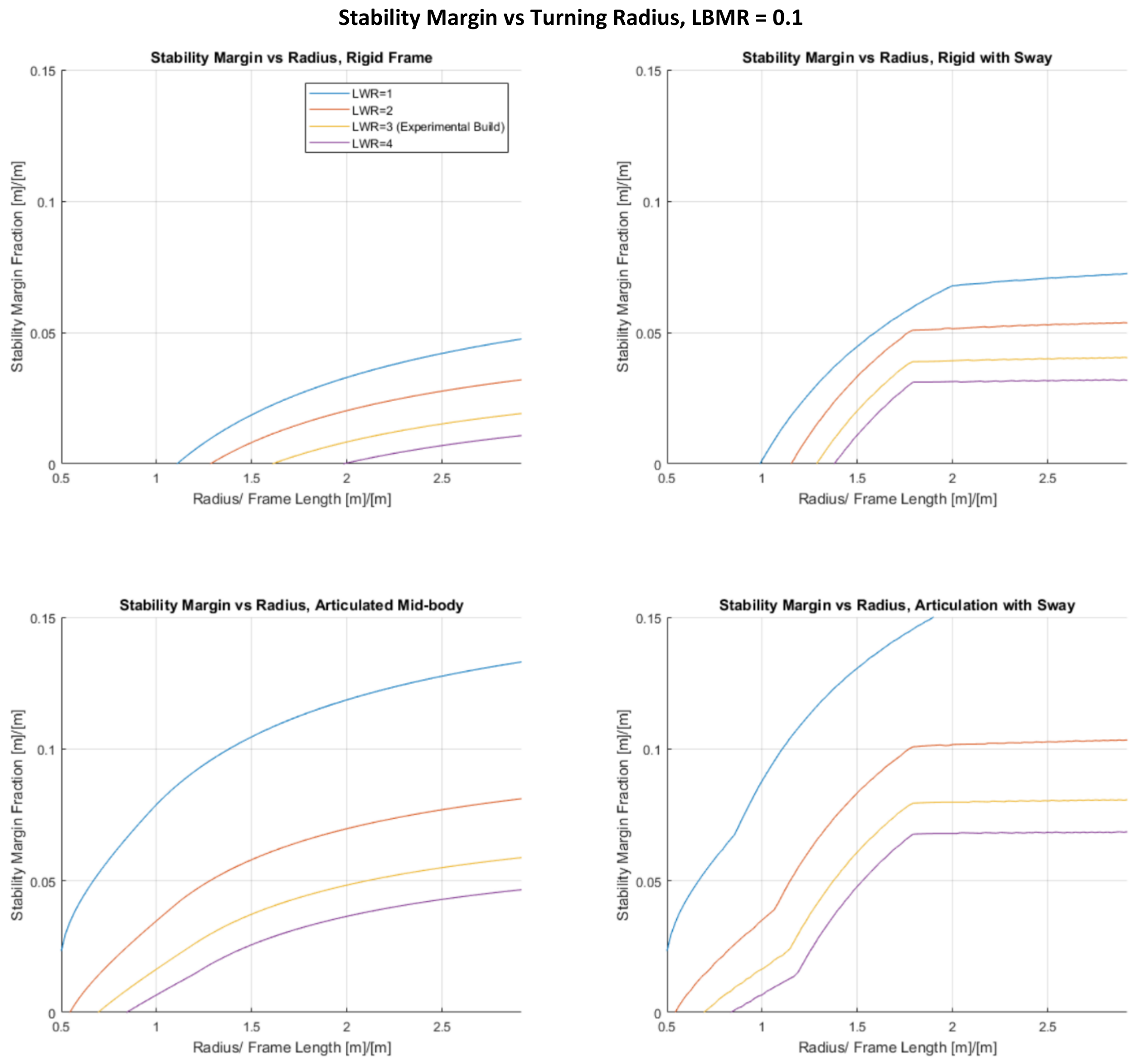

Figure 4-6: Turning gait results as a function of turning radius, shown for rigid frame, rigid frame with sway, articulated frame, and articulated frame with sway. LBMR of 0.1.

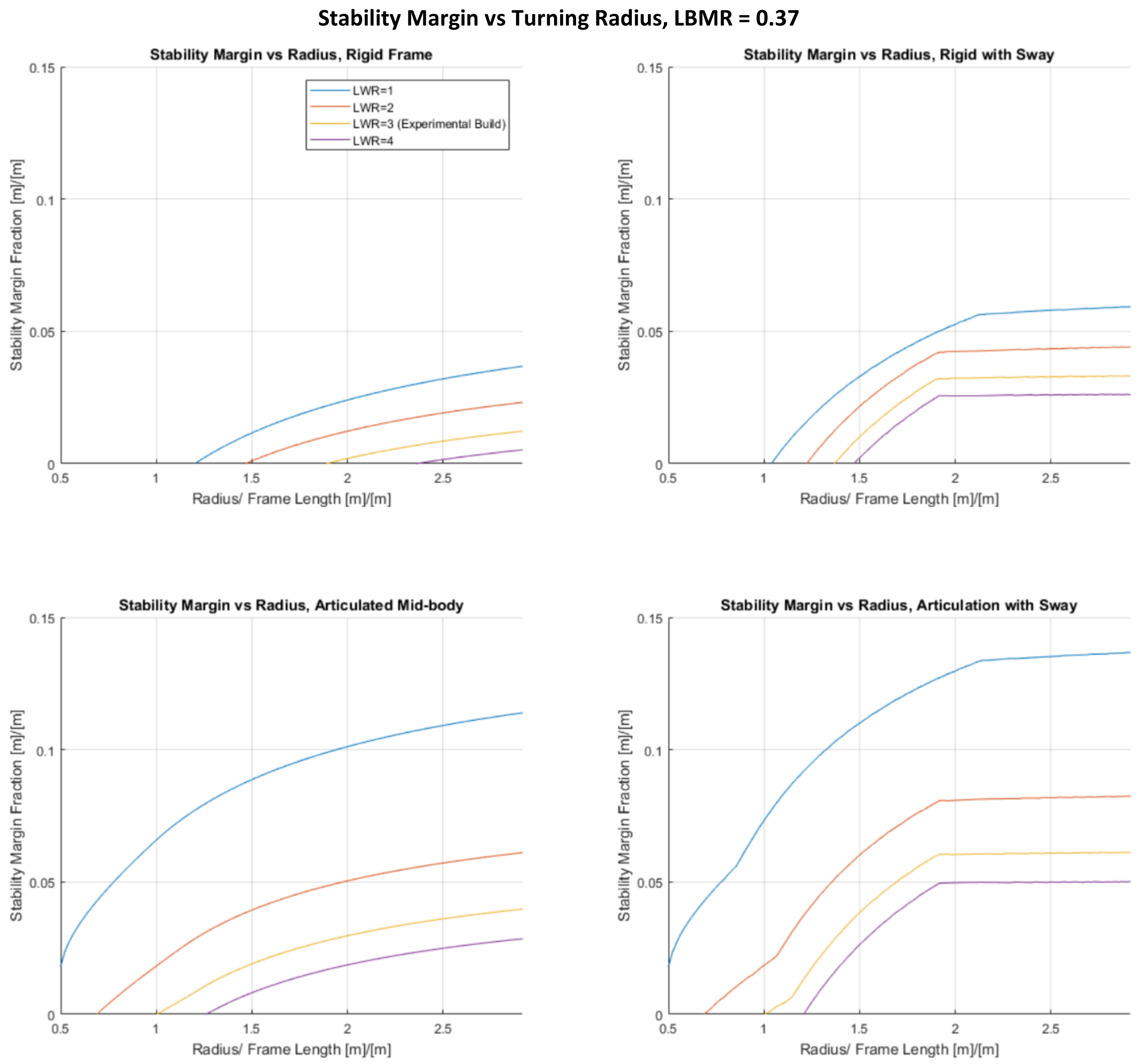

Figure 4-7: Turning gait results as a function of turning radius, shown for rigid frame, rigid frame with sway, articulated frame, and articulated frame with sway. LBMR of 0.37.

To assess the marginal stability impact of increasing mid-body articulation, Figure 4-8, Figure 4-9, and Figure 4-10 show the minimum theoretically stable turning radius (radius at which stability margin approaches 0) as a function of the amount of articulation. For a system with an articulated frame, the amount of articulation may still be constrained by build requirements, and the intention with this data is to provide a metric for assessing the potential benefit of adding articulation given such a constraint. Also apparent across these figures, is the requirement of increasingly large radii for quadrupeds with higher LWR (smaller width). In each case, it is also seen that a higher LBMR requires a larger minimum radius, though the impacts of changes to mass distribution within a given build are less significant than the proportions of the frame itself.

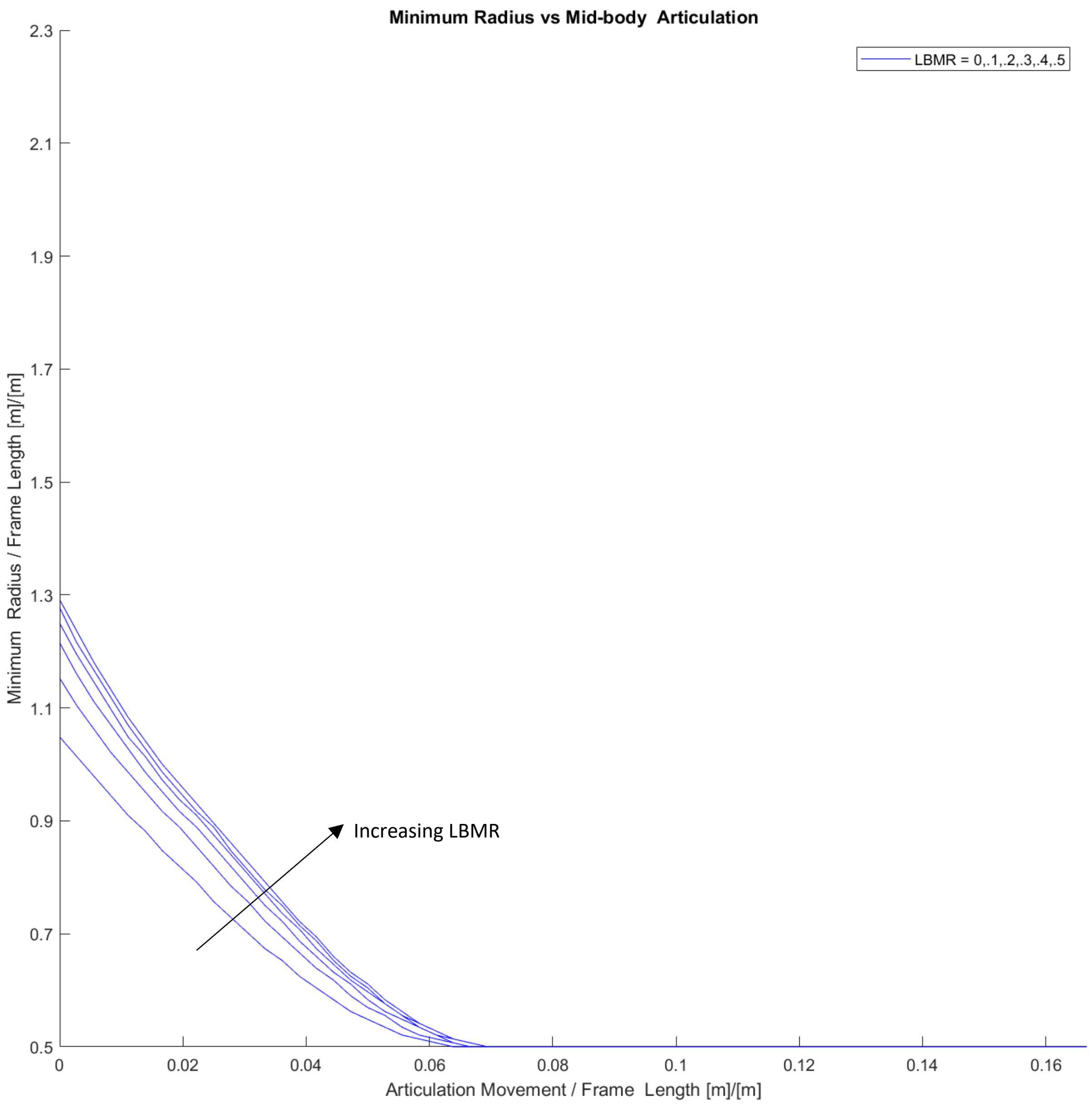

Figure 4-8: Minimum turning radius as a function of torso articulation for LWR = 1.

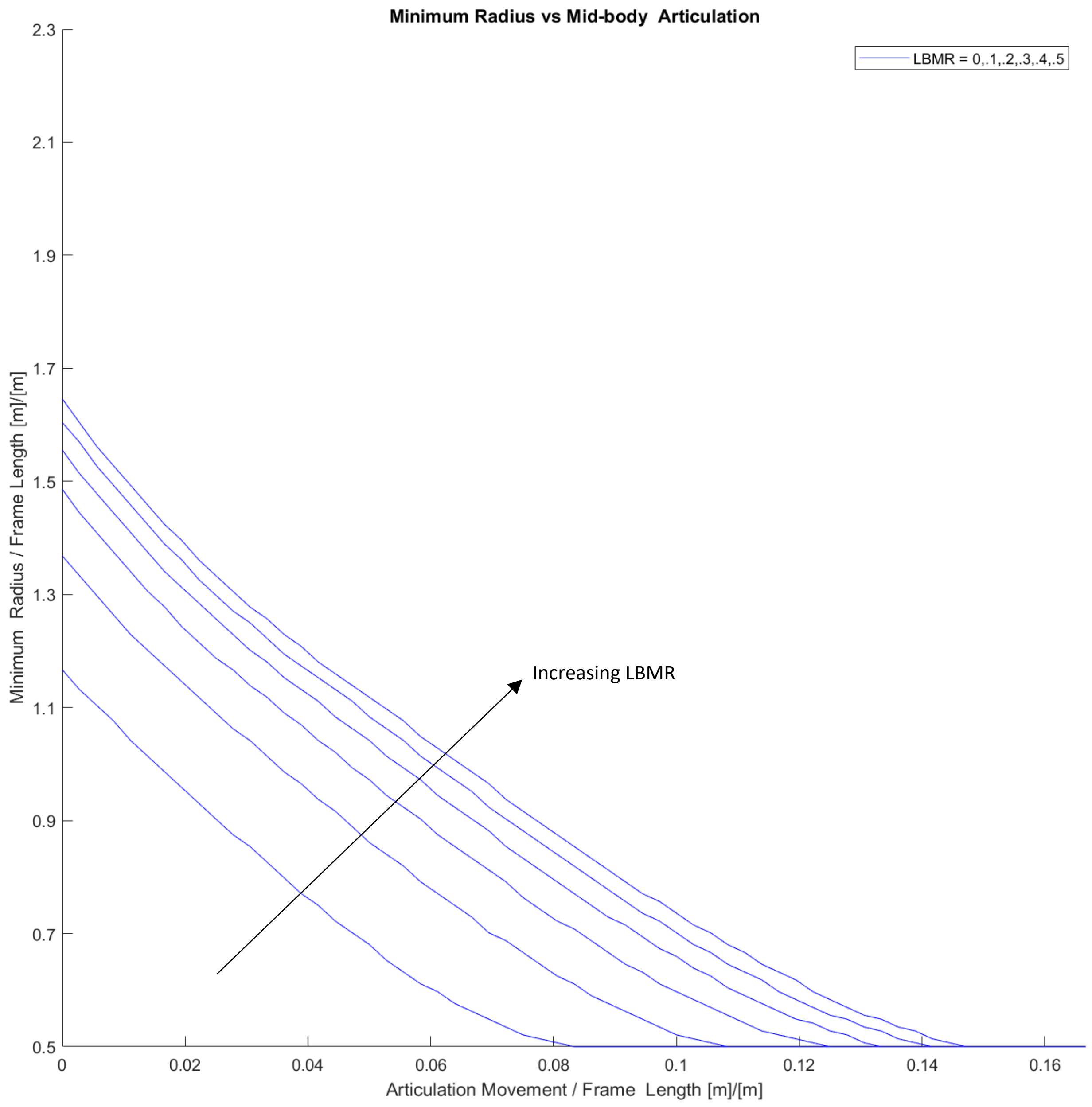

Figure 4-9: Minimum turning radius as a function of torso articulation for LWR = 2.

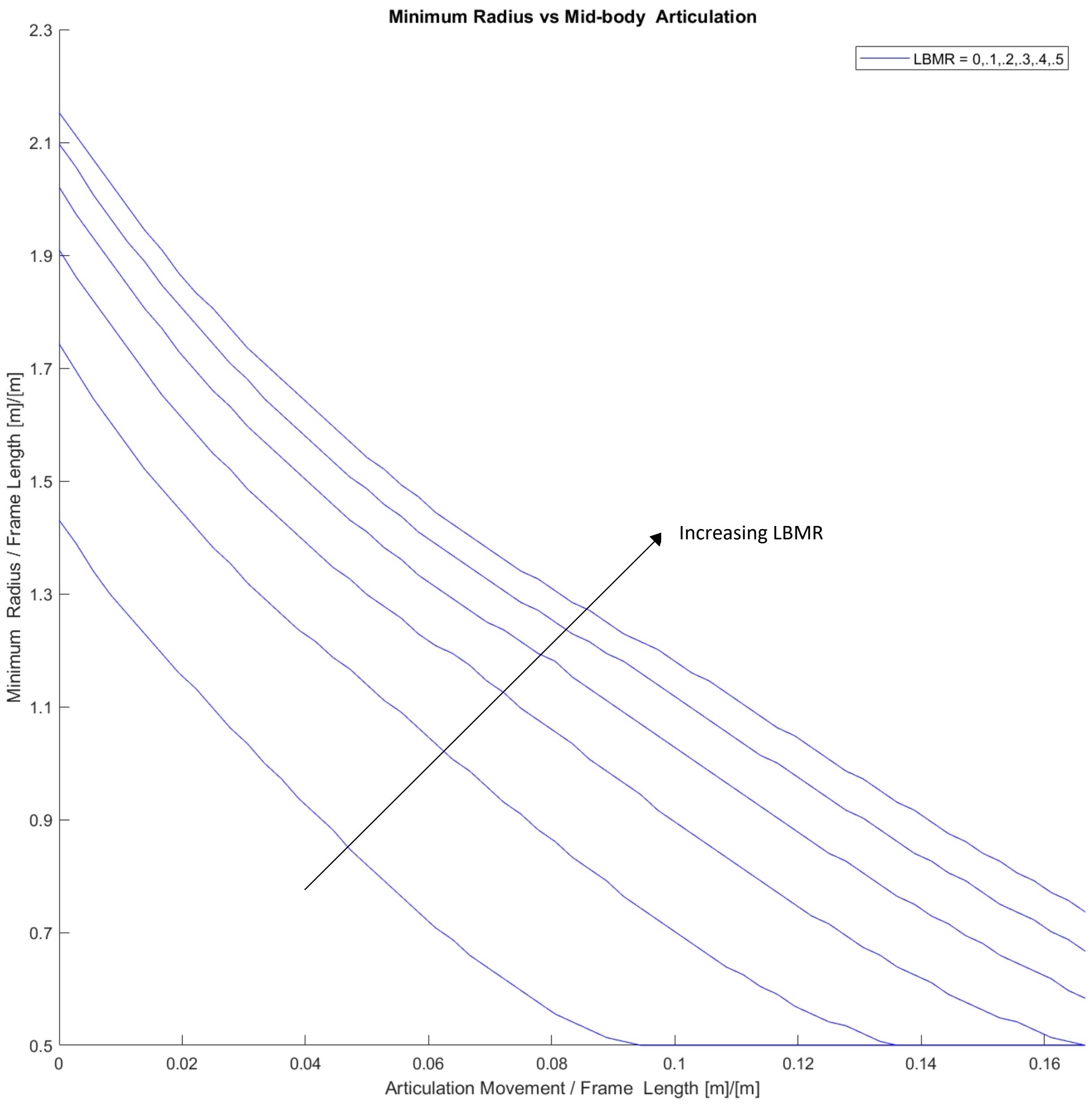

Figure 4-10: Minimum turning radius as a function of torso articulation for LWR = 3.

As seen above, the use of articulation creates a potential for non-trivial improvements in turning radius across the range of build configurations considered. Considering the case where LWR = 1, allowing for articulation of 0.8 torso lengths reduces the minimum theoretical radius from about 1.5 torso lengths to 0.5 (spinning radius). With a LWR of 2, a similar result is seen with articulation of about 0.10-0.14 torso lengths, depending on LBMR. These results represent a significant reduction in the minimum theoretical radii for a relatively small articulation requirement. As this theoretical minimum is based on the point at which stability margin reaches zero, these results do not directly predict real performance but rather indicate a change in the limit of what is theoretically possible. As is the case for any analysis based on stability margin, it can be assumed that the relative trends will provide insight into real performance.

The generated gaits were tested on the experimental quadruped, and the turning radius performance was assessed for mid-body articulation distances between zero and 0.3 torso lengths. Without sway or articulation, turning became unstable as radii dropped below about 10 body lengths. With articulation, stable turning could be completed for radii above 2.8 torso lengths. Figure 4-11 shows these results. While turning performance requires a larger radius than was predicted by both theoretical limits, it does confirm the functionality of the gaits used and the ability of torso articulation to reduce minimum turning radius.

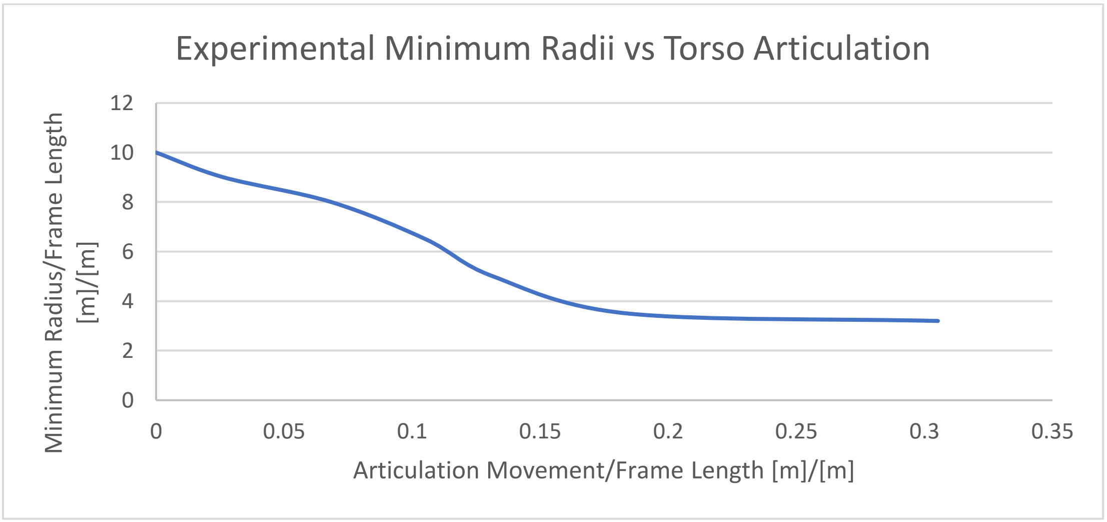

Figure 4-11: Minimum radius plot for experimental quadruped

Chapter 5: Summary

For straight gait, the value of adding mid-body articulation decreases quickly with increasing leg to body mass ratio. The impact is non-trivial, however, particularly at what would likely be the most realistic range of values (between 0.1 and 0.4). In this range, the stability margin improvement for mid-body displacement of one-eighth the frame length was between 24 percent and 73 percent, dependent also on the length to width ratio. The larger benefit was seen for builds with higher LWR, with significant improvement for the experimental build (LWR = 3) ranging between 42 percent and 62 percent as shown in Figure 4-3. Given the structural and energy storage requirements, the likelihood of an individual leg weighing even half as much as the body (LBMR = 0.5) is low. An important consideration is the possibility that LBMR will vary based on use if the quadruped is designed with the intention of carrying passengers and/or supplies. In this case, design should be done with consideration for the highest (least stable) LBMR.

The importance of turning performance would also be a key factor in evaluating the value of mid-body articulation. The decision to explore articulation would be of interest in applications where turning performance is of higher importance, such as if intended use involves ability to navigate urban landscapes and or any other environment where turning may be constrained by obstacles. The potential improvements that may come as a result of a mid-body articulated torso are not without cost, however, and the validity will depend heavily on the relative value of the benefits.

The results of this work indicate that such benefits are possible. For future work, it would be beneficial to apply this general framework to more specific build cases. This would allow for the costs to be better understood and quantified relative to the intended functionality of the quadruped. When attempting to maximize transport capacity, the weight of and space required for an articulated torso will work directly against that objective. Determining an effective way to quantify both the costs and gains associated with torso articulation will greatly extend the usefulness of the results presented here.

In addition to application-specific explorations, additional experimental work on scale models would be useful both to validate the results and to determine the extent of the relationship between the predicted and actual benefits. While the experimental build utilized in this work provided some insight in that regard, the primary purpose was to provide a rough validation of the gaits used in the generation of the theoretical data. Existing small-scale quadrupeds without articulation at the mid-body could be modified to provide a better experimental environment. In addition to more capable hardware, this would allow for the incorporation of extensively tested and optimized gait algorithms. Additionally, incorporating dynamic stability measures and additional factors such as quadruped height and type of terrain would extend the usefulness of the data.

When considering alternative experimental builds, it would also be advantageous to explore the different possible configurations for achieving torso articulation and turning in general. When feet can be approximated as a point-mass, coordination in a turning gait is appropriately less complicated. Likewise, torso articulation can be accomplished simply by addition of a joint at the middle of the torso. Such an approximation would not be possible for large scale vehicles due to the need for significant surface area at the foot.

With non-trivial foot surface area, consideration must be given to the relative rotation and orientations of the foot, shoulder, and torso. This is required both for turning and for incorporating torso articulation. Internal and external rotation of each shoulder, combined with rotation of the mid-body joint, permit both regular turning and full gait functionality of an articulated torso. This was the method used in the experimental build. Alternatively, internal/external rotation could occur at the connection of each pair of legs and the torso. In this configuration, the orientation of both legs in each pair is coupled, but fewer actuators are required and articulation at the torso can occur without rotating each shoulder pair. Given the previous emphasis on feet which are able to be approximated by point mass, there is potential for additional investigation into feet with deliberate orientation. The consideration of potentially incorporating passive joints would also be a worthwhile extension, both for general turning and in the context of articulated torso gait.

Chapter 6: Conclusions

The primary emphasis of this thesis examined the practicality and potential benefits of mid-body articulation in statically stable quadruped gait. Stability analysis was performed across a range of build configurations to determine generalized benefit predictions as a function of these variations in design, both for straight and for turning gait. The gaits developed were verified on a quadruped build with a relatively low natural stability in rigid torso configuration, validating both the gaits that were generated and the potential for predicting stability performance.

While stability for straight gait is improved by the addition of mid-body articulation, the value of these improvements would depend heavily on how stable the specific quadruped build is without articulation as well as the cost of implementation. When turning performance and precise path following ability are important, however, the benefits of including mid-body articulation are very clear in that the minimum radius required for turning can be improved significantly. This work confirms that the use of an articulated torso can benefit both straight and turning gait performance, and provides a base for additional application-specific work as well as more broad articulation-based gait research.

Appendix: System Overview

The development for this project was completed in MATLAB and ROS, with purpose-built modules coded in C++. MATLAB was used to generate the gait patterns used, as well as to test their theoretical stability and generate plots of results. Gaits were generated across a range of parameters and the results were collected for analysis. From these, the gaits selected to test experimentally were exported. The MATLAB scripts and functions used to generate the gait patterns are included at the end of this section. For experimental simulations, specific gait patterns were exported to ROS and executed on the experimental quadruped. As discussed in previous sections, the quadruped was built using ROBOTIS Dynamixel servos, each of which containing a microprocessor for the purpose of processing and executing joint commands. On the robot, an Arobtix-M microcontroller was used to coordinate each of the individual servos based on gait commands.

The Arbotix-M microcontroller is designed specifically to interface with Dynamixel servos, and is compatible with the Arduino IDE. As such, it is also compatible with the ROSSERIAL ROS module (10), allowing for communication between the Arbotix-M and the linux-based ROS system used to generate commands for the quadruped based on the desired gait pattern. The Arduino and C++ ROS module codes written for this project are also included below. The Dynamixel library for Arduino as well as the Arbotix-M documentation were provided by Trossen Robotics (11).

ROS C++ Code for generating leg and torso movement commands from gait:

```
// Kenneth Swidwa kss5251
// This program publishes desired XYZ positions for each quadruped leg
// Input feedback-dependent routine function or well as fixed gait phase tables

#include <ros/ros.h> //ROS library
```

```
#include <quad_driver/joints.h> //joints message file
#include "xyz2j.h" //robot specific constants and joint calculator function
#include <quad_driver/Telequad.h> //service header file (ROS message type definition)
#include <cmath> //standard C++ math library, used for angle calculations
#include <iostream> // I-O libaries
#include <fstream>
using namespace std;

int jMBD = 0;
double th2j = 195.55;

// Class definition for storing LEG INFO, BASIC LEG FUNCTIONS:
class Leg {
       public:
              double x, y, z, xdes, ydes, zdes;
              int jA, jR;
              void move_up (double lift){
                     xdes = xdes - lift;
                     x = xdes;
              };
              void move_down (double lift){
                     xdes = xdes + lift;
                     x = xdes;
              };
              void move_foward (double dist){
                     ydes = ydes + dist;
                     y = ydes;
              };
              void move_y (double dist){
                     ydes = ydes + dist;
                     y = ydes;
              };
              void move_backward (double dist){
                     ydes = ydes - dist;
                     y = ydes;
              };
              void move_z (double dist){
                     zdes = zdes + dist;
                     z = zdes;
              }
              void set_y (double yPos){
                     ydes = yPos;
                     y = ydes;

              };
              void rotate_shoulder(double rot){
                     jR = jR + rot;
              };
              void adduct(int adductVal){
                     jA = jA - adductVal;
              };
              void abduct(int abductVal){
                     jA = jA + abductVal;
              };


              void init(){
                     xdes = strideHeight;
                     ydes = 0;
                     zdes = gaitSpread;
                     x = xdes;
```

```
                        y = ydes;
                        z = zdes;
                };
} leg[4];


//ROS Message Function
quad_driver::Telequad::Request request_constructor();

//multi-function movement commands:
void move_body(double Y, double Z);
void stand_straight();
void swing(double D);
void turnfront(double theta);
void turnback(double theta);
void swingfront(double alpha);


int main(int argc, char **argv){

        ros::init(argc, argv, "straight"); //initialize with ROS
        ros::NodeHandle nh; // set ros node handle
        ros::ServiceClient spawnClient =
nh.serviceClient<quad_driver::Telequad>("move_request"); //Register node as a client
        quad_driver::Telequad::Request req;//create request and response objects
        quad_driver::Telequad::Response resp;//response object
        bool success; //tests for successful message receipt

        //Constants used for gait code:
        int sleepTime = 5000;

        stand_straight();
        int cl1 = 1; int cl2 = 0; int cl3 = 2; int cl4 = 3; // corrected leg numbers (legacy
code)
        int phaseMoveLeg[8] = {cl4, cl2, 0, 0, cl3, cl1, 0, 0};    // set the move leg for each
phase
        int readyCheck = 1;   // used as input to complete additional runs

        // straight gait:
        while(readyCheck){

                //Constants for importing gait
                const int nCommands = 13;
                const int nPhases = 8;
                double gaitCommands[nCommands][nPhases];

                cout << "Initializing gait sequence. Proceed? ";

                cin >> readyCheck;

                //IMPORT GAIT FILE
                ifstream inFile("gait.txt");
                int cmdN = 0;
                while (!inFile.eof()) {
                        for (int phase = 0; phase < nPhases; phase++) {
                                inFile >> gaitCommands[cmdN][phase];
                        }
                        cmdN = cmdN + 1;
                }

                //For each phase, assign YZ and jR command info
```

```
		for (int phase = 0; phase < nPhases; phase++){
			leg[cl1].y = gaitCommands[0][phase];
			leg[cl2].y = gaitCommands[3][phase];
			leg[cl3].y = gaitCommands[6][phase];
			leg[cl4].y = gaitCommands[9][phase];
			leg[cl1].z = -gaitCommands[1][phase];
			leg[cl2].z = -gaitCommands[4][phase];
			leg[cl3].z = -gaitCommands[7][phase];
			leg[cl4].z = -gaitCommands[10][phase];
			leg[cl1].jR = int(1.1*th2j*gaitCommands[2][phase]);
			leg[cl2].jR = int(1.1*th2j*gaitCommands[5][phase]);
			leg[cl3].jR = int(th2j*gaitCommands[8][phase]);
			leg[cl4].jR = int(th2j*gaitCommands[11][phase]);
			jMBD = -int(2*th2j*gaitCommands[12][phase]);

			cout << "PHASE #" << phase+1 << endl;

			// Print leg commands
			int legsVect[4] = {cl1, cl2, cl3, cl4};
			for (int legN = 0; legN < 4; legN++){
				int current_leg = legsVect[legN];
				cout << "X: " << leg[current_leg].x << "  _Y: " <<
leg[current_leg].y << "  _Z: " << leg[current_leg].z << "  _jR: " << leg[current_leg].jR <<
endl;

			}
			cout << "MBD joint: " << jMBD << endl;

			//Execute First Phase
			req = request_constructor(); //build the request (uses current leg
object data)
			success = spawnClient.call(req,resp); //publish request to quadruped
driver code

			//IF PREVIOUS PHASE LEFT LEG IN THE AIR:
			if (phase == 1 || phase == 2 || phase == 5 || phase == 6) {
				cout << "PHASE #" << phase+1 << " LOWER LEG, Confirm Proceed: ";
				cin >> readyCheck;

				//Correct for joint stiffness
				leg[phaseMoveLeg[phase-1]].y = leg[phaseMoveLeg[phase-1]].y +0.5;
				leg[phaseMoveLeg[phase-1]].abduct(10);
				req = request_constructor();
				success = spawnClient.call(req,resp);

				cout << endl << "CONFIRM ADJUST MOVELEG ";
				cin >> readyCheck;

				//Lower Leg
				leg[phaseMoveLeg[phase-1]].y = leg[phaseMoveLeg[phase-1]].y + 1;
				leg[phaseMoveLeg[phase-1]].move_down(strideRise); //lower
previous phase' moveleg

				req = request_constructor();
				success = spawnClient.call(req,resp);

				cout << endl << "CONFIRM ADJUST MOVELEG ";
				cin >> readyCheck;

				//Remove corrections from lowering
```

```
                              leg[phaseMoveLeg[phase-1]].y = leg[phaseMoveLeg[phase-1]].y -
1.5;
                              leg[phaseMoveLeg[phase-1]].adduct(10);
                              req = request_constructor();
                              success = spawnClient.call(req,resp);

                       }

                       //IF this is a lift phase, execute lift process:
                       if (phase == 0 || phase == 1 || phase == 4 || phase == 5) {
                              cout << endl << "PHASE #" << phase+1 << " LIFT LEG, Confirm
Proceed: ";
                              cin >> readyCheck;

                              //Raise leg to stride transfer height:
                              leg[phaseMoveLeg[phase]].move_up(strideRise);
                              req = request_constructor();
                              success = spawnClient.call(req,resp);

                       }
                       cout << endl << "PHASE #" << phase+1 << " COMPLETE, Confirm Proceed: ";
                       cin >> readyCheck;
                       cout << endl;

              }

              //Execute final move request:
              req = request_constructor();
              success = spawnClient.call(req,resp);

       }

       usleep(sleepTime);
}



//Output ROS message detailing desired XYZ, shoulder angles, and MBD angle
quad_driver::Telequad::Request request_constructor(){

       quad_driver::Telequad::Request req;

       for (int i=0; i<4; i=i+1){
              req.x[i] = leg[i].x;
              req.y[i] = leg[i].y;
              req.z[i] = leg[i].z;
              req.jA[i] = leg[i].jA;
              req.jR[i] = leg[i].jR;
              req.jMBD = jMBD;
       }

       //ROS_INFO_STREAM(req.x[0] << " " << req.x[1] << " " << req.x[2] << " " << req.x[3]);

       return req;
}

//General movement functions
void move_body(double Y, double Z){
       double theta[4];
```

```
	for (int i = 0; i<4; i = i+1){
		theta[i] = leg[i].jR/th2j;
		//ROS_INFO_STREAM(i << "old theta: " << theta[i] << "old Y: " << leg[i].y << 
"old Z: " << leg[i].z );
		leg[i].move_y(-Y*cos(theta[i])-Z*sin(theta[i]));
		leg[i].move_z(Y*sin(theta[i])-Z*cos(theta[i]));
		//ROS_INFO_STREAM(i << " theta: " << theta[i] << " Y: " << leg[i].y << " Z: " << 
leg[i].z );
	}

}

// Tell all legs to stand balanced
void stand_straight(){
	for (int i = 0; i<4; i = i+1){
		leg[i].init();
	}
	leg[leg2].move_z(-2*gaitSpread);
	leg[leg3].move_z(-2*gaitSpread);
	leg[leg1].move_y(-strideLength/2);
	leg[leg4].move_y(-strideLength/2);
	for (int i = 0; i<4; i = i+1){
		//leg[i].move_z(zswing);
	}
	//leg[leg3].move_up(0.2*strideRise);
	//leg[leg4].move_up(0.2*strideRise);
}

// general swing function
void swing(double D){

	int W = 3; // geom param build
	int L1 = 6;
	double thetaMid;
	thetaMid = asin(-D/L1);
	ROS_INFO_STREAM("ThetaMid:");
	ROS_INFO_STREAM(thetaMid);
	jMBD = int(2*thetaMid*th2j);
	leg[leg1].jR = jMBD/2;
	leg[leg2].jR = jMBD/2;
	leg[leg3].jR = -jMBD/2;
	leg[leg4].jR = -jMBD/2;
	leg[leg1].y = leg[leg1].y + sin(thetaMid)*W/2;
	leg[leg2].y = leg[leg2].y - sin(thetaMid)*W/2;
	leg[leg3].y = leg[leg3].y - sin(thetaMid)*W/2;
	leg[leg4].y = leg[leg4].y + sin(thetaMid)*W/2;
	leg[leg1].z = leg[leg1].z + (cos(thetaMid)*W/2 - W/2);
	leg[leg2].z = leg[leg2].z - (cos(thetaMid)*W/2 - W/2);
	leg[leg3].z = leg[leg3].z - (cos(thetaMid)*W/2 - W/2);
	leg[leg4].z = leg[leg4].z + (cos(thetaMid)*W/2 - W/2);
}

void turnfront(double theta){
	theta = .01745 * theta; //convert to RAD
	int jJoints= theta*th2j;

	int W = 3; // geom param build
	leg[leg1].jR = jJoints;
	leg[leg2].jR = jJoints;
	leg[leg1].y = leg[leg1].y + sin(theta)*W/2;
	leg[leg2].y = leg[leg2].y - sin(theta)*W/2;
```

```
        leg[leg1].z = leg[leg1].z + (cos(theta)*W/2 - W/2);
        leg[leg2].z = leg[leg2].z - (cos(theta)*W/2 - W/2);
}



void turnback(double theta){
        theta = -.01745 * theta; //convert to RAD
        int jJoints= theta*th2j;

        int W = 3; // geom param build
        leg[leg3].jR = jJoints;
        leg[leg4].jR = jJoints;
        leg[leg3].y = leg[leg3].y - sin(theta)*W/2;
        leg[leg4].y = leg[leg4].y + sin(theta)*W/2;
        leg[leg3].z = leg[leg3].z - (cos(theta)*W/2 - W/2);
        leg[leg4].z = leg[leg4].z + (cos(theta)*W/2 - W/2);
}

void swingfront(double alpha){
        turnback(0);
        alpha  = 0.01745*alpha;
        double Y = (cos(alpha)-1)*12;
        double Z = sin(alpha)*12;
        double theta[2];
        for (int i = 0; i<2; i=i+1){
                leg[i].jR = leg[leg1].jR - alpha*0.01745*th2j;

                theta[i] = leg[i].jR/th2j;
                ROS_INFO_STREAM(i << "old theta: " << theta[i] << "old Y: " << Y << "old Z: " <<
Z);

                leg[i].move_y(-Y*cos(theta[i])-Z*sin(theta[i]));
                leg[i].move_z(Y*sin(theta[i])-Z*cos(theta[i]));
                }

}
```

ROS C++ position command interpolation code and joint command publisher

```
// Kenneth Swidwa kss5251
// This program receives position commands and sends
// interpolated joint commands via ROSSERIAL
#include <ros/ros.h>
#include <quad_driver/joints.h>
#include <quad_driver/Telequad.h>
#include "xyz2j.h"
#include <std_srvs/Empty.h>
#define publishmsg(msg) for (int i=0; i<2500; i=i+1){ pub.publish(msg); }
#define publ ROS_INFO_STREAM(pub.getNumSubscribers(); )

//Initialize parameters used
int j[31]; int jDes[31];
double x[4]; double y[4]; double z[4];
double xdes[4]; double ydes[4]; double zdes[4];
int jA[4]; int jR[4]; int jMBD = 0;
int jAdes[4]; int jRdes[4]; int jMBDdes = 0;
int speedPCT = 100; //joint speed configuration parameter (servo-level)
int initialized = 0; //keeps track of whether or not commands have been sent since power up
int newCmd = 0; //used for interpolation tracking

bool moveRequest(quad_driver::Telequad::Request &req, quad_driver::Telequad::Response &resp){

        //Read desired J from action call, set as desired end position
        for (int i =1; i<31; i = i+1){
                jDes[i] = req.j[i];
        }

        for (int i=0; i<4; i=i+1){
                xdes[i] = req.x[i];
                ydes[i] = req.y[i];
                zdes[i] = req.z[i];

                jAdes[i] = req.jA[i];
                jRdes[i] = req.jR[i];
                jMBDdes = req.jMBD;
        }
        speedPCT = 100;

        if (initialized == 0){
                speedPCT = 50;
                for (int i=1; i<31; i=i+1){
                        j[i] = jDes[i];
                initialized = 1;
                }
                for (int i=0; i<4; i=i+1){
                        x[i]=xdes[i];
                        y[i]=ydes[i];
                        z[i]=zdes[i];
                        jA[i]=jAdes[i];
                        jR[i]=jRdes[i];
                }
                jMBD = jMBDdes;
        }

        newCmd = 1;

        return true;}


int main(int argc, char **argv){
        /*******************************************
        Initialize the ROS System and become A NODE
        *******************************************/
        ros::init(argc, argv, "driver"); //create as ros node
        ros::NodeHandle nh;  //initialize ROS handle
        ros::Publisher pub = nh.advertise<quad_driver::joints>("bot1/cmd_pos",1000);
//initialize as publisher for ROSSERIAL
        ros::ServiceServer server = nh.advertiseService("move_request", &moveRequest);
//initialize as server for gait code (client)
        ros::Rate rate(30); //set message rate

        quad_driver::joints msg;     // create message to send

        //Containers for interpolation
        double deltax[4];     double deltay[4];     double deltaz[4];
        double deltajMBD;     double deltajR[4];    double deltajA[4];
```

```
	double deltaMax = 0;

	//Interpolation Parameters
	int interpoN = 0;
	int numSteps = 15;

	while (ros::ok()){

		if (initialized == 1){

			//IF a new command was received, calculate interpolation deltas
			if (newCmd == 1){
				interpoN = 0;
				for (int i=0; i<4; i=i+1){
					deltax[i] = xdes[i] - x[i];
					deltay[i] = ydes[i] - y[i];
					deltaz[i] = zdes[i] - z[i];
					deltajR[i] = jRdes[i] - jR[i];
					deltajA[i] = jAdes[i] - jA[i];
				}
				newCmd=0;
				deltajMBD = jMBDdes - jMBD;
			}
			if (interpoN < (numSteps)){
				for (int i=0; i<4; i=i+1){

					x[i] = x[i] + deltax[i]/(numSteps+1);
					y[i] = y[i] + deltay[i]/(numSteps+1);
					z[i] = z[i] + deltaz[i]/(numSteps+1);
					jR[i] = jR[i] + deltajR[i]/(numSteps+1);
					jA[i] = jA[i] + deltajA[i]/(numSteps+1);
				}
				jMBD = jMBD + deltajMBD/(numSteps+1);
				interpoN = interpoN +1;

			}
			else{
				for (int i=0; i<4; i=i+1){

					x[i] = xdes[i];
					y[i] = ydes[i];
					z[i] = zdes[i];
				}
			}

			msg = xyz2j(x, y, z, jA, jR, jMBD, speedPCT); //send message request to 
converter, creates msg with joints

		//Publish the constructed message with joint commands (sent over ROSSERIAL)
			pub.publish(msg);
		}
		ros::spinOnce(); //processor subscriber callbacks
		rate.sleep();
	}

}
```

ROS (C++) position to joint converter function

```
#include <quad_driver/joints.h>

//************************************//
//Set up STRIDE PARAMETERS (test gait)//
//************************************//
double L = 9.5;
double strideHeight = 9.6;//height to hips when standing
double strideRise = 2.0; //desired leg lift height
double strideLength = 3.8;
double gaitSpread = .95; //(z shift from hips)
double frontShift = -2; //(yshift of front legs from hips) previously -2
double rearShift = 0.4; //(yshift of rear legs from hips) previously 0.4
double zswing = 1.25;
int shoulderAdjust =10;
double gaitPosY[4] = {-strideLength/2, -strideLength/6, strideLength/6, strideLength/2};

struct posinfo{
	double x[4]; double y[4];  double z[4]; double zinit[4]; int jM[31]; int
setSpeed;//INITIALIZE POSITION VECTORS
} ;

//**********************//
//Set up LEG PARAMETERS//
//********************//
double L1 = 7.5;
double L2 = 3.0;
double Lmax = 12;
double Lmin = 4;

//FIXED PARAMETERS&CALCULATIONS//
int leg1 = 0; int leg2 = 1; int leg3 = 2; int leg4 = 3; // leg4(B), leg1(F), leg3(B), leg2(F)
int shoulderJoint[4] = {2, 9, 16, 23};

quad_driver::joints xyz2j(double x[4], double y[4], double z[4], int jA[4], int jR[4], int
jMBD, int maxJointSpeed){

	quad_driver::joints msg;

	//XYZ2JOINT PARAMETERS//
	double R[4];  double theta[4];  double theta1[4];  double theta2[4];  double theta3[4];
double phi[4];
	double zfix[4];
	double L1 = 7.5;
	double L2 = 3.0;
	double Lmax = 10.45; double Lmin = 6
	int leg1 = 0; int leg2 = 1; int leg3 = 2; int leg4 = 3;

	//adjust z for sign convention of joints
	zfix[leg1]=z[leg1];   zfix[leg2]=z[leg2];   zfix[leg3]=-z[leg3];  zfix[leg4]=-z[leg4];
	for (int i = 0; i<4 ; i = i + 1){
		//find distance between shoulder and foot command, limit by constraints to avoid
bad joint requests
		R[i] = max(Lmin,min(Lmax,sqrt((x[i]*x[i])+(y[i]*y[i])+(zfix[i]*zfix[i]))));

		//find joint angles for legs based on desired position in space
		if (R[i] <= Lmax && R[i] >= Lmin){
			theta[i] = atan(-y[i]/x[i]);
			phi[i] = 3.14/2-acos(zfix[i]/R[i]);
			theta1[i] = acos(((R[i]*R[i])+(L1*L1)-(L2*L2))/(2*R[i]*L1));
			theta2[i] = acos(((L1*L1)+(L2*L2)-(R[i]*R[i]))/(2*L1*L2));
			theta3[i] = 3.1416 - theta1[i] - theta2[i];
			}
		}

		//Assign each joint based on desired leg length and orientation, simplified for flat
terrain
		msg.j3 = (int)((theta[leg1]+theta1[leg1])*195.55+512);
		msg.j4 = (int)(-(3.1416-theta2[leg1])*195.55+473);
		msg.j5 = (int)((-theta3[leg1]+theta[leg1])*195.55+780);
		msg.j2 = jA[leg1]+(int)((phi[leg1])*195.55+520);
		msg.j7 = msg.j2; // Jdesired[2]; /l/mirroring Jdesired2 adjust ankle to sit flat

		msg.j10 = (int)(-(theta[leg2]+theta1[leg2])*195.55+512);
		msg.j11 = (int)((3.1416-theta2[leg2])*195.55+551);
		msg.j12 = (int)(-(-theta3[leg2]+theta[leg2])*195.55+244);
		msg.j9  =-jA[leg2]+(int)((phi[leg2])*195.55+504);//jM (-) to give (+) ABDUCTION
		msg.j14 = msg.j9; // Jdesired[9]; //mirroring Jdesired2 adjust ankle to sit flat

		msg.j17 = (int)((-theta[leg3]+theta1[leg3])*195.55+512);
		msg.j18 = (int)(-(3.1416-theta2[leg3])*195.55+473);
		msg.j19 = (int)((-theta3[leg3]-theta[leg3])*195.55+780);
		msg.j16 = jA[leg3]+(int)((phi[leg3])*195.55+520);
		msg.j21 = 1024-msg.j16 ;//Jdesired[16]; //mirroring J2 adjust ankle to sit flat

		msg.j24 = (int)(-(-theta[leg4]+theta1[leg4])*195.55+512);
		msg.j25 = (int)((3.1416-theta2[leg4])*195.55+551);
		msg.j26 = (int)(-(-theta3[leg4]-theta[leg4])*195.55+244);
		msg.j23 = -jA[leg4]+(int)((phi[leg4])*195.55+504);//jM (-) to give (+) ABDUCTION
		msg.j28 = 1024-msg.j23 ;//Jdesired[23]; //mirroring Jdesired2 adjust ankle to sit flat

		//set shoulder angle (zero for alignment with y axis)
		msg.j1 = jR[leg1]+666;		msg.j8 = jR[leg2]+358;
		msg.j15 = jR[leg3]+666;		msg.j22 = jR[leg4]+358;

		//fixed ankle joints
		msg.j6 = 512;  msg.j13 = 512;		msg.j20 = 512; msg.j27 = 512;

		//adjust mid-body angle
		msg.j29 = 512+jMBD;

		//control max servo speed for testing
		msg.d1=maxJointSpeed;

		//print shoulder angles
		ROS_INFO_STREAM(jR[0] << " " << jR[1] << " " << jR[2] << " " << jR[3]);

		return msg;

}
```

Arduino code for Arbotix-M:

```
// Publisher code to receive joint request from ROS and send to each individual servo

#include <ax12.h>
#include <ros.h>
#include <quad_driver/joints.h>
#include <std_msgs/String.h>

ros::NodeHandle nh;
```

```
//Initialize joint arrays
int Jcurrent[31]; //31 so index matches joint id
int Jdesired[31];
int Jcmd[31];
int speed_max = 1024;
int speed_pct = 10;
int interpolateBool = 0;


//Callback function for subscriber. Triggered anytime message received
void pos_req_cb(const quad_driver::joints& msg) {

        /*for (int i = 1; i < 31; i=i+1){
        Jdesired[i]={msg.j1};
        }*/ //Replace below with a for loop
        Jdesired[1] = { msg.j1 };  Jdesired[2] = { msg.j2 };  Jdesired[3] = { msg.j3 };
        Jdesired[4] = { msg.j4 };  Jdesired[5] = { msg.j5 };  Jdesired[6] = { msg.j6 };
        Jdesired[7] = { msg.j7 }; // end Leg 1
        Jdesired[8] = { msg.j8 };  Jdesired[9] = { msg.j9 };  Jdesired[10] = { msg.j10 };
        Jdesired[11] = { msg.j11 };  Jdesired[12] = { msg.j12 };  Jdesired[13] = { msg.j13 };
        Jdesired[14] = { msg.j14 }; //end Leg 2
        Jdesired[15] = { msg.j15 };  Jdesired[16] = { msg.j16 };  Jdesired[17] = { msg.j17 };
        Jdesired[18] = { msg.j18 };  Jdesired[19] = { msg.j19 };  Jdesired[20] = { msg.j20 };
        Jdesired[21] = { msg.j21 }; //end Leg 3
        Jdesired[22] = { msg.j22 };  Jdesired[23] = { msg.j23 };  Jdesired[24] = { msg.j24 };
        Jdesired[25] = { msg.j25 };  Jdesired[26] = { msg.j26 };  Jdesired[27] = { msg.j27 };
        Jdesired[28] = { msg.j28 }; //end Leg 4
        Jdesired[29] = { msg.j29 };  //Midbody
        int speed_pct = { msg.d1 };

        // parameters for local interpolation (diagnostic)
        int maxDelta = 0;
        int Jdelta[31];

        //SET base speed from speed_pct and speed_max
        for (int i = 0; i < 31; i = i + 1) {
                dxlSetGoalSpeed(i, speed_max * speed_pct);
        }

        //SET UP JOINT INTERPOLATION(for diagnostics if not being  done in ROS)
        if (interpolateBool) {
                positionReader(); //Find current joint positions


                                                // FIND All Deltas AND max delta
                for (int i = 1; i < 31; i = i + 1) {
                        Jdelta[i] = Jdesired[i] - Jcurrent[i];
                        if (maxDelta < abs(Jdelta[i])) { //Find the max delta (used to calculate
N later)
                                maxDelta = abs(Jdelta[i]);
                        }
                }
                for (int i = 0; i < 31; i = i + 1) {
                        dxlSetGoalSpeed(i, speed_max * speed_pct * Jdelta[i] / maxDelta);
                }
        }

        for (int i = 1; i < 31; i = i + 1) {
                Jcmd[i] = Jdesired[i];
        }
        moveLegs(); //Send joint commands to servos
}

//SET UP ROS SUBSCRIBER
ros::Subscriber<quad_driver::joints> sub("bot1/cmd_pos", pos_req_cb);

//SETUP FUNCTION
void setup() {
        delay(100);
        nh.getHardware()->setBaud(115200);
        //Adjust servo speed based on request
        for (int i = 0; i < 31; i = i + 1) {
                dxlSetGoalSpeed(i, speed_max * speed_pct);
        }

        positionReader(); //Reads current joint positions from servos

                                        //INITIALIZE AS A ROS NODE
        nh.initNode();
        nh.subscribe(sub);
}

//KEEP RUNNING WAITING FOR A NEW MESSAGE
void loop() {
        nh.spinOnce();
        delay(1);
}

//FUNCTION TO UPDATE DESIRED POSITION OF EACH JOINT
void moveLegs() {
        for (int i = 1; i < 31; i = i + 1) {
                //if (i != i*6 || i != i*7){
                SetPosition(i, Jcmd[i]);
                //}
                delay(1);
        }
}

//FUNCTION TO GET CURRENT JOINT POSITIONS//
void positionReader() {
        for (int i = 1; i < 31; i = i + 1) {
                Jcurrent[i] = dxlGetPosition(i);
        }
}
```

MATLAB Code for generating straight gait:

```
%finds the pose associated with the phase before moving moveleg

%straight gait experimental
%this is the modified code for experimental outputs (everything needed
%to run on the robot

function [pFeet, pShoulders, pExpShoulders, mbd_angle] = straight_gait_experimental(moveLeg,
mbdShift)

global L W Rx Ry WS shoulderVector expShoulderVector

mbd_angle = asin(2*mbdShift/L) %cmath.h
```

```
pFront = [cos(mbd_angle)*L/2 0]; %accomodates for length change due to MBD shift
pRear = [-cos(mbd_angle)*L/2 0];

%Find location of shoulders and experimental shoulders based front and back
%of shoulder pair
pShoulders = [pFront + shoulderVector
              pFront - shoulderVector
              pRear + shoulderVector
              pRear - shoulderVector ];

pExpShoulders = [pFront + expShoulderVector*rotmatr(-mbd_angle)
              pFront - expShoulderVector*rotmatr(-mbd_angle)
              pRear + expShoulderVector*rotmatr(mbd_angle)
              pRear - expShoulderVector*rotmatr(mbd_angle) ];

%First Pose: LHS centered with COF, feet centered with shoulders

%If a true move phase, find the other leg on the same side as the moving
%leg, the leg opposite and adjacent to the moveleg, and the leg opposite
%and non-adjacent to the movelg
if moveLeg < 5

[moveSide, oppAdj, oppNonAdj] =  leg_assigner(moveLeg);

    pFeet = zeros(4,2);
    pFeet(oppAdj,:) = pShoulders(oppAdj,:);
    pFeet(oppNonAdj,:) = pShoulders(oppNonAdj,:);
    pFeet(moveLeg,:) = pShoulders(moveLeg,:) - [Rx/2 0];
    if pShoulders(moveSide,1) > 0
        pFeet(moveSide,:) = pShoulders(moveSide,:) - [Rx/2 0];
    else
        pFeet(moveSide,:) = pShoulders(moveSide,:) + [Rx/2 0];
    end


%These phases are used for experimental exporting of gait and are not
%needed for stability calculations due to symmetry: They control the two
%stages where both feet are on the ground and the body is moving
elseif moveLeg == 6

    pFeet = pShoulders;
    pFeet(2,:) = pFeet(2,:) + [Rx/2 0];
    pFeet(4,:) = pFeet(4,:) + [Rx/2 0];


elseif moveLeg == 5

    pFeet = pShoulders;
    pFeet(1,:) = pFeet(1,:) + [Rx/2 0];
    pFeet(3,:) = pFeet(3,:) + [Rx/2 0];


elseif moveLeg == 8
    pFront = [L/2 0]; %accomodates for length change due to MBD shift
    pRear = [-L/2 0];
    pFeet = pShoulders; %use the adjusted shoulder length to find previous step foot positions
    pShoulders = [pFront + shoulderVector
              pFront - shoulderVector
              pRear + shoulderVector
              pRear - shoulderVector ];

    pExpShoulders = [pFront + expShoulderVector*rotmatr(0)
              pFront - expShoulderVector*rotmatr(0)
              pRear + expShoulderVector*rotmatr(-0)
              pRear - expShoulderVector*rotmatr(-0) ];

    pFeet(2,:) = pFeet(2,:) + [Rx/2 0];
    pFeet(4,:) = pFeet(4,:) + [Rx/2 0];
    pFeet = pFeet - [Rx/4 0];


elseif moveLeg == 7
    pFront = [L/2 0]; %accomodates for length change due to MBD shift
    pRear = [-L/2 0];
    pFeet = pShoulders; %use the adjusted shoulder length to find previous step foot positions
    pShoulders = [pFront + shoulderVector
              pFront - shoulderVector
              pRear + shoulderVector
              pRear - shoulderVector ];

    pExpShoulders = [pFront + expShoulderVector*rotmatr(0)
              pFront - expShoulderVector*rotmatr(0)
              pRear + expShoulderVector*rotmatr(-0)
              pRear - expShoulderVector*rotmatr(-0) ];

    pFeet(1,:) = pFeet(1,:) + [Rx/2 0];
    pFeet(3,:) = pFeet(3,:) + [Rx/2 0];
    pFeet = pFeet - [Rx/4 0];

end

end
```

MATLAB functions for Turning Gait:

```
function [pShoulders, pShouldersMatrix, pShouldersNext, pShouldersPrev, radii_o, radii_i,
tBodyNext, tBodyPrev, mbd_angle, pShouldersExp] = set_frame(mbd_Shift, radii)

global L shoulderVector Rx Ry expShoulderVector


%find body layout from articulation:
mbd_angle = asin(mbd_Shift/(L/2));
artic_L = cos(mbd_angle)*L;

%find angle between pF and pR in circular coordinates aligned with
%center of turning Radius [figA]
tDeltaBody = 2*asin(artic_L/(2*radii));
%Angular Position of Front and Rear from the angle between them, center 0
tRear =-tDeltaBody/2;
tFront = tDeltaBody/2;
%Cartesian coords:
pFront = [radii*sin(tFront)  radii*cos(tFront)];
pRear = [radii*sin(tRear)  radii*cos(tRear)];

%Find the current shoulder positions based on mid-body articulation
pShoulders = [pFront + shoulderVector*rotmatr(-mbd_angle)
              pFront - shoulderVector*rotmatr(-mbd_angle)
              pRear + shoulderVector*rotmatr(mbd_angle)
              pRear - shoulderVector*rotmatr(mbd_angle)];

pShouldersExp = [pFront + expShoulderVector*rotmatr(-mbd_angle)
  pFront - expShoulderVector*rotmatr(-mbd_angle)
  pRear + expShoulderVector*rotmatr(mbd_angle)
  pRear - expShoulderVector*rotmatr(mbd_angle)];

%Find inner and outer radii:
radii_i = sqrt(pShoulders(4,1)^2+pShoulders(4,2)^2);
radii_o = sqrt(pShoulders(3,1)^2+pShoulders(3,2)^2);

% x^2+y^2 = radii_o  --- INTERSECTION OF circle and WorkSpace Line
% dont even need these points, just radii and xpos gives the angle
%because triangles. Hyp is radii, x is opposite
%use the half angle to find next and previous steps (for foot pos)
t2=asin((pShoulders(1,1)+Rx/2)/radii_o);
t1=asin((pShoulders(1,1)-Rx/2)/radii_o);
t3=acos((pShoulders(1,2)-Ry/2)/radii_o);
%tDelta=(t2-t1)/2.1%TODO check math above
tc=asin((pShoulders(1,1))/radii_o);
tDelta = min(min(abs(t2-tc),abs(t1-tc)),abs(t3-tc));

%Angular positions of next and prev front and rear
tRearPrev = tRear - tDelta;
tFrontPrev = tFront - tDelta;
tRearNext = tRear + tDelta;
tFrontNext = tFront + tDelta;

%Convert angular positions to cartesian:
pFrontPrev = [radii*sin(tFrontPrev)  radii*cos(tFrontPrev)];
pRearPrev = [radii*sin(tRearPrev)  radii*cos(tRearPrev)];

pFrontNext = [radii*sin(tFrontNext)  radii*cos(tFrontNext)];
pRearNext = [radii*sin(tRearNext)  radii*cos(tRearNext)];

bodyNextDir = (pFrontNext-pRearNext)/norm(pFrontNext-pRearNext);
bodyPrevDir = (pFrontPrev-pRearPrev)/norm(pFrontPrev-pRearPrev);

%atan to find vector because opposite/adj
tBodyNext = atan(bodyNextDir(2)/bodyNextDir(1));
tBodyPrev = atan(bodyPrevDir(2)/bodyPrevDir(1));
%vectors pointing in direction of BODY orientation. MBD Angle is relative
%to this body orientation

pShouldersNext = [pFrontNext + shoulderVector*rotmatr(tBodyNext-mbd_angle)
              pFrontNext - shoulderVector*rotmatr(tBodyNext-mbd_angle)
              pRearNext + shoulderVector*rotmatr(tBodyNext+mbd_angle)
              pRearNext - shoulderVector*rotmatr(tBodyNext+mbd_angle)];

pShouldersPrev = [pFrontPrev + shoulderVector*rotmatr(tBodyPrev-mbd_angle)
              pFrontPrev - shoulderVector*rotmatr(tBodyPrev-mbd_angle)
              pRearPrev + shoulderVector*rotmatr(tBodyPrev+mbd_angle)
              pRearPrev - shoulderVector*rotmatr(tBodyPrev+mbd_angle)];


end



function [margins,intersections,pShoulders,pShouldersNext,pShouldersPrev, ...
```

```
    tBodyNext,radii_i,radii_o,yOffset, tMBD, pShouldersExp]= ...
    turn_gait(mbd_Shift, sway, radii, LWR, LBMR_local)

%Container function for portability
%set global parameters: L, W, WS, Rx, Ry, ShoulderVector (local to pF/pR)
set_params;
%set_global(body_length, LWR, LBMR_local, WSLF, WSWF)

%Find Shoulders & body rotation for current, previous and Next Pose
[pShoulders,pShouldersMatrix, pShouldersNext, pShouldersPrev, radii_o,...
    radii_i, tBodyNext, tBodyPrev, tMBD, pShouldersExp] = set_frame_interpolation(mbd_Shift,
radii);


pFront = (pShoulders(1,:) - pShoulders(2,:))/2 + pShoulders(2,:);
pRear = (pShoulders(3,:) - pShoulders(4,:))/2 + pShoulders(4,:);


%bring to y = 0 to center the reference frame;
yOffset = -pFront(2);
pShoulders(:,2) = pShoulders(:,2) - pFront(2);
pShouldersNext(:,2) = pShouldersNext(:,2) - pFront(2);
pShouldersPrev(:,2) = pShouldersPrev(:,2) - pFront(2);
pShouldersExp(:,2) = pShouldersExp(:,2) - pFront(2);


for moveLeg = 1:4
    pFeet = turn_feet(pShoulders, pShouldersPrev, pShouldersNext, moveLeg);
    COM = find_COM(pFeet,pShoulders,mbd_Shift,sway);
    [margin_vect(moveLeg), inter_vect(:,moveLeg)] = stab_marg(pFeet,COM,moveLeg);

end

margins = margin_vect;
intersections = inter_vect;

end

function pFeet = turn_feet(pShoulders, pShouldersPrev, pShouldersNext, moveLeg)
%Assigns feet based on which leg is supposed to be moving and the next and previous frame
locations

%RUN for moveLeg and opposite moveLeg, find margin for both.

pFeet = pShoulders;


if (moveLeg <= 4)
    [moveSide, oppAdj, oppNonAdj] =   leg_assigner(moveLeg);



    if (moveLeg == 4)
        pFeet(moveLeg,:) = pShouldersPrev(moveLeg,:);
        pFeet(moveSide,:) = pShouldersPrev(moveSide,:);
    elseif (moveLeg == 3)
        pFeet(moveLeg,:) = pShouldersPrev(moveLeg,:);
        pFeet(moveSide,:) = pShouldersPrev(moveSide,:);
    elseif (moveLeg == 2)
        pFeet(moveLeg,:) = pShouldersPrev(moveLeg,:);
        pFeet(moveSide,:) = pShouldersNext(moveSide,:);
    elseif (moveLeg == 1)
        pFeet(moveLeg,:) = pShouldersPrev(moveLeg,:);
        pFeet(moveSide,:) = pShouldersNext(moveSide,:);
    end

end


    if (moveLeg == 6 || moveLeg == 8)
        pFeet(4,:) = pShouldersNext(4,:);
        pFeet(2,:) = pShouldersNext(2,:);
    elseif (moveLeg == 5 || moveLeg == 7)
        pFeet(3,:) = pShouldersNext(3,:);
        pFeet(1,:) = pShouldersNext(1,:);
    end
```

Bibliography

1. **Bekker, M. G.** *Off-the-Road Locomotion.* Ann Arbor : The University of Michigan Press, 1960.

2. **Santos, Pablo Gonzalez de, Garcia, Elana and Estremera, Joaquin.** *Quadrupedal Locomoation: An Introduction to the Control of Four-legged Robots.* Madrid, Spain : Springer, 2006.

3. **LS3 - Legged Squad Support Systems .** ***Boston Dynamics.*** **[Online] 2012. [Cited: Feb 6, 2016.] http://www.bostondynamics.com/robot_ls3.html.**

4. ***Gaits and Geometry of a Walking Chair for the Disabled.*** **Zhang, Chang-de and Song, Shin-Min. No. 3/4, Chicago : Journal of Terramechanics, 1989, Vol. 26, pp. 211-233.**

5. ***The Adaptive Suspension Vehicle.*** **Waldron, Kenneth J. and McGhee, Robert B. 1986, IEEE Control Systems Magazine, pp. 7-12.**

6. Sicliano, Bruno and Khatib, Oussama. ***Springer Handbook of Robotics.*** **Berlin Heidelberg : Spring-Verlag, 2008.**

7. Hardarson, Freyr. ***Stability analysis and synthesis of statically balanced walking for quadruped robots.*** **Stockholm : Royal Institute of Technology, KTH, 2002.**

8. Haueisen, Brooke M. ***Investigation of an Articulated Spine in a Qaudruped Robotic System.*** **Ann Arbor, Michigan : s.n., 2011.**

9. ***Quadruped Creeping Gaits.*** **Frank, R. B. McGhee' A. A. Los Angeles : Mathematical Biosciences, 1968, Vol. 3, pp. 331-351.**

10. Dohare, Shibhansh. rosserial_arduino/Tutorials/Arduino IDE Setup. *ROS.org.* [Online] Open Source Robotics Foundation, 06 16, 2016.

11. Learn.TrossenRobotics.com. *ArbotiX Introduction.* [Online] Trossen Robotics, 2012. http://learn.trossenrobotics.com/arbotix.