Peer Reviewed

Title:
Simulating DuCTT and optimizing control for DuCTT with machine learning

Author:
Xydes, Alexander Lawrence

Acceptance Date:
2015

Series:
UC San Diego Electronic Theses and Dissertations

Degree:
M.S., UC San Diego

Permalink:
http://escholarship.org/uc/item/2f56s2b4

Local Identifier:
b8982731

Abstract:
Air duct inspection frequently requires mobile robots to visit areas inaccessible to humans. Tensegrity robots, with their small mass and cross-section are highly suited to inspecting air ducts without impeding the flow within the duct. One tensegrity robot designed at the UCSD Coordinated Robotics Lab for this task is the Duct Climbing Tetrahedral Tensegrity (DuCTT) robot. This robot consists of two tetrahedral sections connected by eight cables. This work presents a way to simulate this robot in the NASA Tensegrity Robotics Toolkit (NTRT). Once the robot is simulated, control strategies can be explored in a variety of different environments. These strategies can get the robot to climb and traverse air ducts. and ways to optimize controllers based on those strategies. The two different strategies are using sine waves to control the actuators, and a state-machine controller. Each controller is optimized using one stage of Monte-Carlo parameter estimation and a second stage genetic algorithm to improve the parameters found by the first stage. The state-machine controller ends up with better performance, 6.77 cm/s, compared to the sine wave controller's 0.38 cm/s

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Simulating DuCTT and optimizing control for DuCTT with machine learning

A Thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science

in

Computer Science

by

Alexander Lawrence Xydes

Committee in charge:

Professor Yoav Freund, Chair
Professor Thomas Bewley
Professor Ryan Kastner

2015

The Thesis of Alexander Lawrence Xydes is approved and is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Figure 1.3 has been reproduced with permission from B. T. Mirletz, I.-w. Park, T. E. Flemons, A. K. Agogino, R. D. Quinn, and V. Sun- spiral, Design and Control of Modular Spine-Like Tensegrity Structures, in 6WCSCM: Sixth World Conference on Structural Control and Monitoring, no. July, 2014.

I would like to acknowledge Jeffrey Friesen for his work on the Lagrangian dynamics model of the robot and his contributions to Section 2.5.

Section 3.1, in part, has been submitted for publication of the material as it may appear in The Second Generation Prototype of A Duct Climbing Tensegrity Robot, DuCTTv2, in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, 2015, J. Friesen, P. Glick, M. Fanton, P. Manovi, A. Xydes, and T. Bewley. The thesis author was a co-author of this paper.

VITA

2010	Bachelor of Science, California Polytechnic State University, San Luis Obispo
2015	Master of Science, University of California, San Diego

ABSTRACT OF THE THESIS

Simulating DuCTT and optimizing control for DuCTT with machine learning

by

Alexander Lawrence Xydes

Master of Science in Computer Science

University of California, San Diego, 2015

Professor Yoav Freund, Chair

Air duct inspection frequently requires mobile robots to visit areas inaccessible to humans. Tensegrity robots, with their small mass and cross-section are highly suited to inspecting air ducts without impeding the flow within the duct. One tensegrity robot designed at the UCSD Coordinated Robotics Lab for this task is the Duct Climbing Tetrahedral Tensegrity (DuCTT) robot. This robot consists of two tetrahedral sections connected by eight cables.

This work presents a way to simulate this robot in the NASA Tensegrity Robotics Toolkit (NTRT). Once the robot is simulated, control strategies can be explored in a

variety of different environments. These strategies can get the robot to climb and traverse air ducts. and ways to optimize controllers based on those strategies.

The two different strategies are using sine waves to control the actuators, and a state-machine controller. Each controller is optimized using one stage of Monte-Carlo parameter estimation and a second stage genetic algorithm to improve the parameters found by the first stage. The state-machine controller ends up with better performance, 6.77 $\frac{\text{cm}}{\text{s}}$, compared to the sine wave controller's 0.38 $\frac{\text{cm}}{\text{s}}$.

Introduction

Inspecting pipes can discover cracks before a spill of the pipe contents occur and many times the pipes are inaccessible to humans. Air ducts also need to be inspected to search for wiring shorts or other damage that can occur during years of use. Robots are increasingly being used to do this job and provide an eye inside the duct or pipe because they can reach the places inaccessible to humans. They can be designed specifically for the job at hand. Some designs use magnetic wheels, others use spring-loaded wheels to press against the walls, while still others use an inchworm motion. Tensegrity principles provide yet another design paradigm that can lead to lighter more efficient robots.

Tensegrity principles were first used in the art world during the 1960s to create standing structures. A tensegrity (tensile integrity) structure consists of members that are under compression or tension only [16, 32]. Without the bending or shear forces that come from being under both compression and tension, tensegrity structures can be made of lighter-weight materials without sacrificing robustness. The compression members (rods) are suspended between the members under tension (cables) leading to a web of tension and compression members. External forces are distributed along the web of tension members, reducing the amount of force each individual compression member undergoes. This light-weight and robust nature makes robotic tensegrities very suitable for missions like air duct-inspection where this is an advantage.

The University of California, San Diego (UCSD) Coordinated Robotics Lab has been developing a tensegrity-based robot for this task of duct-inspection. Their

robot, called the Duct Climbing Tetrahedral Tensegrity (DuCTT) robot, is a class three tensegrity where three compression members (rods) come together at a joint [19]. Class k tensegrities have k rods touching at each joint. This group has designed and implemented a force density based inverse kinematic control algorithm to get the Duct Climbing Tetrahedral Tensegrity (DuCTT) robot to climb a duct. This method is complex, involving a lot of parameters. It is also currently slow, with the robot climbing at a speed of 1.4 $\frac{\text{cm}}{\text{s}}$; although it has not yet been optimized for speed [18].

The tasks for this work include: simulating the DuCTT robot to provide a platform for controls development, validating that simulation. The last task is developing control algorithm(s) to make DuCTT climb a vertical air duct, traverse a horizontal duct and a horizontal plane. These three environments represent the three basic areas of a duct that this robot is likely to encounter in the real world. The horizontal plane is also a good proxy for air ducts that are substantially larger than the robot.

There are many different simulation environments for games and robotics to choose from, even a few designed specifically for tensegrities. However, there's only one modern simulation environment designed for tensegrity robotics: NASA's Tensegrity Robotics Toolkit (NTRT) [3]. This toolkit provides a lot of tensegrity specific constructions that make it easier to model a tensegrity robot and test it out in various environments.

To validate the simulation, an Euler-Lagrange model of the robot is constructed and then compared to the NASA Tensegrity Robotics Toolkit (NTRT) simulated robot. This involves controlling the NTRT simulation and Euler-Lagrange model in the same manner, and comparing the nodal positions of each model after the dynamics are calculated. The nodal positions of each model should match closely if the model's agree with each other.

Finally, machine learning is used to optimize three different controllers to find

the best control scheme for the DuCTT robot. Tensegrity robots include a large number of actuators, at least one per cable, which leads to a complex control problem. Machine learning is ideally suited to test out large numbers of controller parameter values and compare their performance. A combination of Monte-Carlo parameter estimation and genetic algorithm is used to search the parameter space and find the best set of parameter values. The Monte-Carlo parameter estimation finds a good basic set of parameters and then the genetic algorithm improves on those to get much better performance.

To compare the different controllers, average speed and distance traveled over a minute are the metrics used. While more complicated metrics are possible to develop, these two basic ones provide good performance without much computational cost. The metrics are also very intuitive and not ambiguous. Before exploring these methodologies in detail, it's worth looking at what else is available to provide some context.

Chapter 1

Literature Review

This work explores the use of different machine learning algorithms to a robot designed for pipe-inspection. The target robot is a hybrid of tensegrity and traditional mobile robots. Given that designing control policies for tensegrity robots is much harder than for traditional mobile robots because of the high degrees of freedom, it seems appropriate to use machine learning to optimize the control policy of this robot.

1.1 Duct-Inspection Robots

Inspecting pipelines, air ducts and similarly small enclosed spaces is hard or impossible for a human to do. Small mobile robots can successfully access these spaces and perform the desired tasks. There is a high degree of variability in the design of these robots as the geometry of the specific duct, pipeline or enclosed space combined with the desired task greatly affects the best design of a robot. Some robots built for inspecting pipelines use wheels, pressed against the sides of the enclosed space, to move. Others use magnetic wheels to stick to the sides of suitably ferrous enclosures [24]. Unfortunately, these wheeled robots can not handle irregularities such as sharp corners, changing duct shape or obstacles very well.

One approach to handling irregularities uses an inchworm motion to propel the robot along the duct. This motion involves securing one part of the robot to the duct,

pushing another part forward, securing the second part of the robot to the duct, then releasing the first part and pulling it forward. These robots take many forms, some using wheels [30], while others use complicated legs to press against the walls [23].

Another class of robot uses mechanical legs to handle irregularities like sharp corners and obstacles [34]. One such robot, whose design is based on a spider, has been shown in a 2D simulation to handle multiple types of such irregularities in ducts [27]. Mechanical legs are currently heavy and power-hungry, which limits their potential in an untethered mobile robot. One potential solution to this problem is the use of mobile robots based on tensegrity principles [31].

1.2 Tensegrity Robots

Tensegrity principles are used in a variety of manners to achieve different goals in regards to robotics. The elements of a tensegrity structure are only under compression or tension, not both; neither do the elements experience bending or shearing forces. Because of this, the materials making up such a structure can be more mass-efficient than in a regular robot. This means that more of the robot's weight can be taken up by power sources which should lead to a longer operating duration.

A good example of tensegrity principles allowing for lighter materials involves designing a robotic leg. The authors design a robotic leg and foot modeled after the human leg and foot connection [7]. The MIT Robotic Cheetah leg is redesigned to include cables and then tested for performance and stress induced on the leg materials. A finite element analysis indicates that the tensegrity cable reduces the stress on the leg and foot materials [7]. This means that the materials making up the leg and foot can be more lightweight as they do not have to withstand as much force while in use.

One robot based on the principles of tensegrity is designed to absorb the impact from landing on a planet and then be able to navigate the planet's surface. This SU-

PERBall robot is a 6 rod icosahedron tensegrity robot with 24 cables connecting the rods and allowing it to deform and move [11]. The control algorithm is evolved via co-evolutionary algorithms to optimize the motion of a single flop when applied in series to form a rolling motion [11].

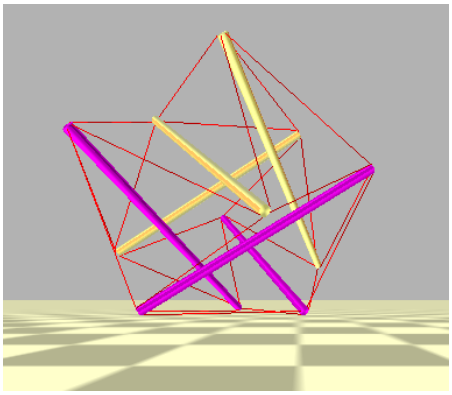


Figure 1.1. SUPERBall rendered in the NTRT toolkit.

Another robot uses tensegrity principles to create a tail structure used to propel the robot. This structure consists of two sets of 6 bars in a pair-wise "X" configuration, where one set is stacked vertically on top the other [10]. Cables connect the bars forming a square. Tensegrity principles allow the tail structure to be light-weight and to retain its structure under plane loading and it's own weight when oriented perpendicular to gravity.

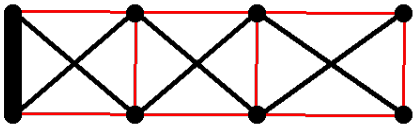


Figure 1.2. Representation of one set of bars of the tail structure for the CPG swimmer. Bars are in black, cables in red.

Another group explored the control of spine-like tensegrity structures. These structures consist of repeated rigid substructures connected by cables, where the rigid substructures could be shaped like tetrahedrons (a.k.a. Tetraspine, Fig. 1.3), ribs, or other structures [25]. In the tetrahedron case six cables connected each segment, in the rib

case 7 active cables and 4 passive (non-actuated cables). In all morphologies, the authors used twelve segments total to balance the need for behavioral flexibility and limiting computational complexity. Only the tetrahedron segment was prototyped in hardware (Fig. 1.3) as it provides lots of space for circuitry, motors and sensors [25].

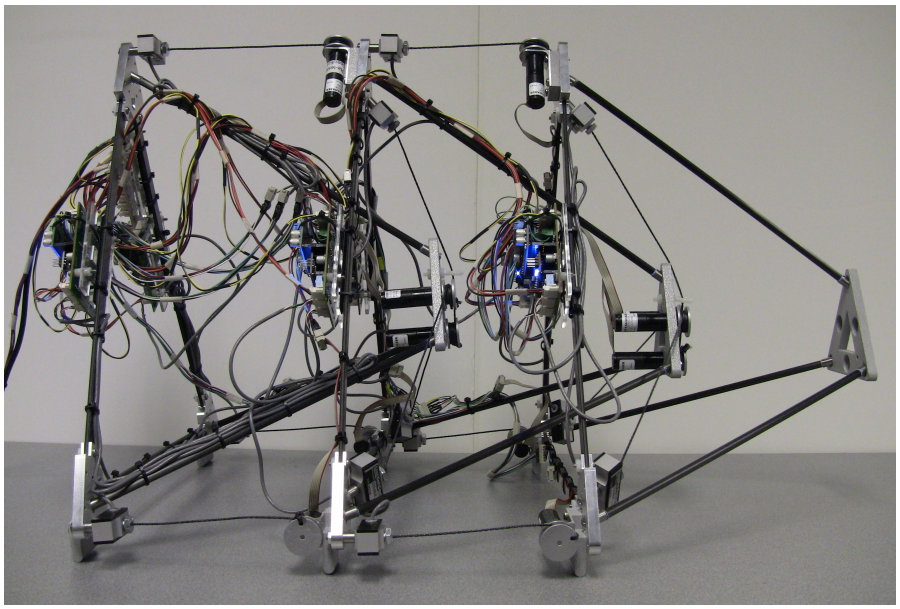


Figure 1.3. TetraSpine3 built by In Won Park and Vytas SunSpiral in the Intelligent Robotics Group at NASA Ames Research Center. [25]

1.3 Robotic and Tensegrity Simulation

Robotic simulations provide great environments to test and validate a robotic design. They also allow machine learning algorithms to test a design thousands of times without the difficulties of setting up a physical robot.

One way of simulating tensegrity-based robots and structures is to build a custom simulator, which is sometimes done in Matlab [13]. However, this strategy usually does not include collision physics and therefore cannot be used to investigate how a tensegrity robot interacts with an environment.

Some standard simulation environments include the Bullet Physics Engine, Havok,

ODE, and PhysX [15]. None of these were designed with tensegrity simulation in mind. They are either game engines that have been re-purposed for robotic simulation or they are designed for rigid robots. When they do include soft-body dynamics (Bullet, Havok Cloth, PhysX), they are either designed for game elements, like clothing, or are not quite accurate enough for tensegrity simulation [12].

Tensegrity specific simulators (Springie [6]) and simulators that support tensegrity structures do exist (Push-Me-Pull-Me [5]). Unfortunately, both of these are oriented towards static structures and not mobile robotics, which is necessary for this work. As a result there is no support for motors nor for programmatically moving the tensegrity structure.

Fortunately, a good candidate exists in the form of the NASA Tensegrity Robotics Toolkit (NTRT) [3]. This library adds tensegrity specific features to the Bullet Physics Engine (Bullet) [1] to enable fast prototyping and accurate simulation of mobile tensegrity robots. It is described in more detail in Section 2.3.1.1.

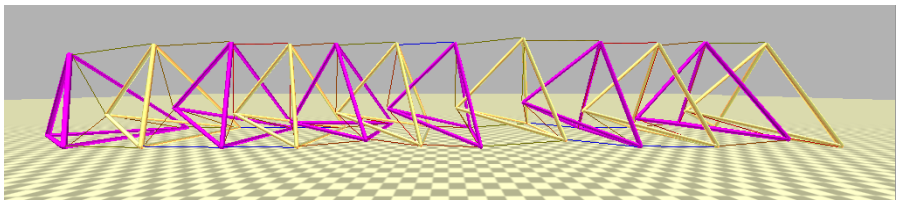


Figure 1.4. Tetraspine rendered in the NTRT toolkit.

1.4 Tensegrity Control

Controlling mobile tensegrity robots is more difficult than traditional skid-steer or ackermann drive robots. This comes from the fact that the tensegrity based robots contain more degrees of freedom and more actuators than the traditional robots. A tensegrity robot could potentially have an actuator for each cable in the robot, and some of these contain a lot of cables (SUPERBall has 24 cables for instance [11]).

1.4.1 Central Pattern Generators

One way of coordinating all these cables takes inspiration from the animal neurons that produce rhythmic outputs. These central pattern generators (CPGs) produce their patterns on their own, without any sensory input. Several of the central pattern generator (CPG) properties are useful for mobile robots including that they allow for distributed control with fast control loops [12]. CPGs are also resistant to disruptions [12].

The SUPERBall bot started with just CPGs for control and with their parameters learned via evolutionary algorithms [21]. It then moved to a hybrid CPG and inverse kinematic (IK) control algorithm that performed better in terms of the quality of the path than with just CPGs alone [12]. The inverse kinematic (IK) algorithm provides feedback that help correct the cable rest-lengths as well as provide a way to follow a desired trajectory. The CPG equation (Eq.1.1) that governs the rest lengths of the cables for this robot only contains four parameters [21]:

$$y(t) = C + A * sin(\omega t + \phi) \tag{1.1}$$

where C is the center position of the sine wave, A is its amplitude, ω is the angular frequency and ϕ is the phase.

The swimmer with a tensegrity tail uses CPGs with closed loop control to develop target propulsive gaits. The author starts with open-loop control of the CPGs to gather characteristics and metrics about the structure and control [10]. Then a closed loop control method with some time delay is developed which demonstrated robustness and the ability to target different gaits.

The tetraspine robot also uses CPG nodes to control the crawling motion of the robot. The authors group the CPGs in sets of three with one set per robot segment, leading to 11 sets of 3 CPGs for a total of 33 CPGs [33]. The equations driving these

CPGs are based on the equations from the salamander robot in [20]:

$$\dot{\theta}_i = 2\pi v_i + \sum_j r_j w_{ij} sin\left(\theta_j - \theta_i - \phi_{ij}\right) \tag{1.2}$$

$$\ddot{r}_i = a_i\left(\frac{a_i}{4}\left(R_i - r_i\right) - \dot{r}_i\right) \tag{1.3}$$

$$V_i = r_i\left(cos\left(\theta_i\right)\right) \tag{1.4}$$

In the equations above, v is a frequency term, r is the amplitude of a node, θ is the phase of a node, w is the coupling weight between two nodes and ϕ is a phase offset. The calculated amplitude (Eq. 1.3) reaches a constant value after the setpoint R_i, and a_i is a positive constant. The CPGs provide target velocities (V_i) based on the calculated phase ($\dot{\theta}_i$) and amplitude ($\ddot{r}_i$) for the low-level impedance controllers attached to each outside cable. These impedance controllers output a target tension for the cable they control according to the following formula [33]:

$$T = T_0 + K\left(L - L_0\right) + B\left(V - V_0\right) \tag{1.5}$$

In Equation 1.5, the authors define T_0 as an offset tension, K as a stiffness gain, L as the actual length of the cable, and L_0 as the rest length of the cable. They also define B as the velocity gain, with V being the actual velocity and V_0 being the target velocity. With this architecture, the robot achieves a speed of about 6.6 cm/s on flat terrain.

CPGs have been used extensively in the past to control tensegrity based robots [9, 11, 10, 25] and therefore were investigated as a good approach to distributed control for this work.

1.4.2 Learning the Controller Parameters

A common approach for mobile tensegrity robots is to learn the parameters via Monte Carlo parameter estimation [25] and/or some sort of evolutionary algorithm [21, 29, 22]. This is done because of the large number of parameters in control policies for typical tensegrity morphologies, which makes hand-tuning the control policy difficult.

One exploration of the SUPERBall robot compares the results of control policies developed by hand, with a centralized evolution and a decentralized evolution algorithm [21]. The fitness function for all algorithms is the distance moved by the robot in the given time. For this robot, with 24 different actuated cables and 96 parameters, developing a control policy by hand is extremely difficult. Their solution performed very poorly compared to the two evolutionary algorithm approaches, moving just under 100 meters in 60 seconds compared to 450 meters and 850 meters for the two evolutionary algorithms [21]. The centralized evolutionary algorithm learns a single control policy which sets all 96 parameters for their CPG controllers. The worst k policies are removed at the end of each generation and replaced by mutated versions of the best k policies. In the decentralized evolutionary algorithm, each agent only searches through the four parameters of its own CPG equation (Eq. 1.1). However, an agent's choice of parameters also affects the value of the choices of the other agents. The authors handle this by taking a fixed number of samples of the population of control policies for each agent at each generation. Then, to evaluate the samples they explore using the generational average, the max value and an historical average of the evaluations of the fitness function for a sample. The historical average strategy reaches the best score fastest and is more consistent than the other two strategies [21].

An earlier approach to controlling a simple three or four rod tensegrity uses a fixed length genetic algorithm. This algorithm optimizes the controllers over their

parameter space (10 parameters for the 3 rod, 13 for the 4 rod) [29]. The algorithm is run for 200 generations with a population size of 200 and keeping the 100 best population members after each generation. Those 100 best members are used to produce 50 mutated versions of those members and 50 new members using a pairwise one-point crossing. The fitness function is again the distance traveled over the time period, which in this case is 10 seconds. Different runs of the evolutionary algorithm produce different gaits of the robot, some are like the inchworm motion described in Sec. 1.1, some are closer to a bounding gait. This bounding gait of the three rod robot achieves a speed of 0.45 m/s which is about double that of the 0.26 m/s from the inchworm gait [29].

An alternative to genetic algorithms involves the use of Monte Carlo parameter estimation. One example applies this procedure to the Tetraspine robot introduced in Sec. 1.2 [25]. The authors explore multiple different spine morphologies in this paper, with the highest number of parameters being 202 for the tetrahedral complex morphology. Again, the fitness function is the distance moved by the robot under test. The Monte Carlo parameter estimation is run for between 10 thousand and 20 thousand trials. Afterward, the authors use Gaussian sampling with a standard deviation of 0.005 on the highest performing parameter sets. This is done to discover possibly better sets of parameters and allows the procedure to do limited hill-climbing without actually calculating the gradient [25].

For this work, both Monte Carlo parameter estimation and genetic algorithms seem to be the most relevant. These two procedures should provide everything needed to find reasonable control policies.

1.5 Acknowledgments

Figure 1.3 has been reproduced with permission from B. T. Mirletz, I.-w. Park, T. E. Flemons, A. K. Agogino, R. D. Quinn, and V. Sun- spiral, Design and Control of

Modular Spine-Like Tensegrity Structures, in 6WCSCM: Sixth World Conference on Structural Control and Monitoring, no. July, 2014.

Chapter 2

Approach

2.1 Overview

The goal of this thesis is to develop controllers for navigating the Duct Climbing Tetrahedral Tensegrity (DuCTT) robot within multiple types of ducts or pipes. These environments include:

- a vertical duct
- a horizontal duct
- a horizontal plane (no duct, or duct with walls bigger than robot)

The controller developed and optimized here allows the simulated model of the robot to climb a duct with a speed of about 6.7 $\frac{\text{cm}}{s}$. This is about 4x faster than the physical robot has ever climbed. This controller is a state-machine based controller with a small number of parameters. To do this, we first construct a physically and mechanically accurate simulation of the DuCTT robot (Fig. 2.1). Then we design control algorithms to get the simulation to climb or traverse a duct or plane. Finally we use machine learning to optimize the control algorithms we designed.

The robot platform is described in more detail in Section 2.2. The steps needed to simulate this robot and the desired environments are explained in Section 2.3. Section

2.4 describes the various controllers designed and tested for this work. Section 2.5 details how the validation of this simulation's mechanics and dynamics is carried out. Finally, Section 2.6 details the learning algorithms and cost function used to optimize the controllers.

2.2 Target Platform

The chosen platform for this work is the Duct Climbing Tetrahedral Tensegrity (DuCTT) robotic platform. This platform has been prototyped and developed over the past couple of years by the UCSD Coordinated Robotics Lab [19, 18]. With three rods touching at each joint, it is considered a class 3 tensegrity structure. It consists of two tetrahedral frames connected via eight actuated cables along with a linear actuator on the topmost and bottommost rods (Fig. 2.1). End-caps are placed on the ends of the linear actuators to provide greater friction between the platform and the walls of the environment as well as to protect the linear actuators.

The four cables going between two parallel rods are labeled vertical cables, and the four connecting two perpendicular rods are labeled saddle cables (Fig. 2.1b). The two linear actuators and the eight cable actuators are the only means of locomotion for this platform. In the past these were used to develop an open-loop controller using inverse kinematics [19] that was capable of traversing a vertical duct.

2.3 Simulation

While a physical platform is technically available to work with, it makes using machine learning techniques to discover an optimal controller harder. Without simulation, one would have to gather data from lots of different test runs and play it back during the machine learning. Simulation is also much safer than testing algorithms on a physical robot. This limits any possible damage to the physical robot while bugs in the algorithm

(a) DuCTTv2 prototype hardware

(b) Render of DuCTTv2 Prototype highlighting the two types of cables

Figure 2.1. DuCTT robot: hardware and render

are ironed out. It also increases the speed of iteration between algorithm versions, since it's easier to test the algorithm in simulation, make changes then test again, all without needing to repair or maintain a physical robot. Another benefit of working with simulation is that it decouples the algorithm development from the hardware development. In other words, the algorithm development is not dependent on having the physical robot in working order.

2.3.1 Libraries

The NASA Tensegrity Robotics Toolkit (NTRT) performs the simulation duties for this thesis. This toolkit was chosen because it provides good tools to build models of tensegrity structures as well as a machine learning framework that can be leveraged for running the learning algorithms.

2.3.1.1 NASA Tensegrity Robotics Toolkit (NTRT)

The NASA Tensegrity Robotics Toolkit (NTRT) [3] is an open-source library developed by the Intelligent Robotics group at NASA Ames to facilitate simulation of robots based on tensegrity principles. It is built to run on top of the open-source Bullet Physics Engine (Bullet) (Sec. 2.3.1.2). The developers have added tensegrity specific physics to the engine related to cables that make it easier to simulate the cables that make up a tensegrity structure. Since NTRT is a wrapper around the Bullet Physics Engine (Bullet) not all of Bullet's features are natively supported by the NTRT tools (have wrappers). However, one can always use Bullet's features in raw form if needed.

NTRT contains two main libraries and some peripheral libraries at this time. The main library "core" provides all the physical modeling and simulation capabilities. In the "core" library the default Bullet softbodies for cables are not used because they are not physically accurate enough for the needs of NTRT. Instead NTRT developers added their own two point linear cable model that uses Hooke's law forces and a linear damping term.

The other main library "tgcreator" provides tools to make building tensegrity structures easier. These tools allow the specification of rods and cables as sets of points in Cartesian space. These rods and cables can be linked together to form substructures, and these substructures are linked in a tree to form the complete structure.

The peripheral library "learning" contains tools to run learning algorithms on tensegrity structures. These tools include Monte Carlo parameter estimation, Gaussian sampling, co-evolution and genetic algorithms. These tools also provide the ability for mutation of a set of parameters, controller elitism and varying the number of children a controller has. The tools relevant to this work are detailed in Section 2.6.

The peripheral library "controllers" contains low-level controllers useful for

directly controlling cables. These controllers include an impedance controller, a PID controller and a tension controller. The impedance controller is the only controller used in this work and is detailed in Section 2.4.3.

2.3.1.2 Bullet

The Bullet Physics Library is an open-source real-time physics simulator [1]. The simulator is written in C++ and is mostly used by game developers and movie studios. It provides the underlying physics for collision detection, and rigid body modeling. It also contains a simple graphical viewer for debugging simulations in real-time.

2.3.2 Robot

A simulated robot was built up using the nodes, pairs and tags syntax of the NTRT toolkit (Fig. 2.2). This involves specifying nodes as $< x, y, z >$ positions in space and then connecting two nodes as a pair and assigning that pair a tag or series of tags. Each tag is a string, and the tags allow one to assign a single rigid body builder to a series of pairs without creating a new builder for each pair of nodes. Each builder represents a single type of Bullet object, whether that be a rigid body cylinder, or a slider constraint.

One tetrahedron of the robot is completely specified, then the other is created by copying the first specification and moving it some vertical distance away. Then the cables between tetrahedrons are specified using the same nodes already created for the tetrahedrons, adding four vertical cables and four saddle cables between the tetrahedrons. The caps on the ends of the linear actuator rods are simulated via sphere rigid bodies half-embedded in the ends of the simulated linear actuator rods. This way the simulated robot ends up with half-spheres on the ends of those rods just like the physical robot. In order to properly simulate the target platform in NTRT, features had to be added to the toolkit. These included two Bullet constraints, and touch sensors.

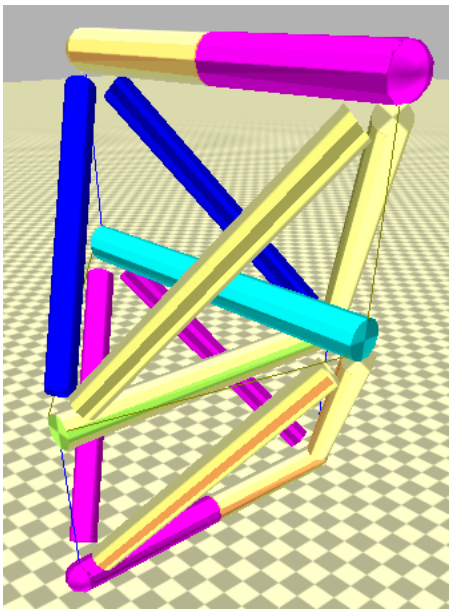


Figure 2.2. Simulation of DuCTT Prototype in NTRT

2.3.2.1 Constraints

Bullet Hinge and Slider constraints were added to the toolkit in a generic manner so that other developers could leverage the work done for this thesis. The hinge constraints are more limited than those provided directly by Bullet, this was done to simplify adding them using the NTRT builder tools. The Bullet hinges can be specified using arbitrary axes of rotation, however that proved too complicated to correctly port into the NTRT builder tools. Instead, the hinge constraints added for this thesis can only have X,Y, or Z axes of rotation. This limitation does not affect the quality of the simulated robot, and in fact improves it. Because the horizontal rods of the robot are specified parallel to the X and Z axes, these limited constraints better represent the axes of rotation on the joints of the physical robot.

The Bullet slider constraints faced a similar axis problem when porting to NTRT. The slider constraint axis of rotation is also limited to the X, Y, and Z axes because it simplifies the port and allows for easier specification of the constraint using the NTRT

builder tools. While these NTRT constraints could be modified to take in any valid vector representing an axis, a better improvement would be to automatically configure the axis of rotation for both constraints based on the location of the nodes that they are connected to. At the same time developers should be able to provide an axis via configuration that would override this automatically calculated axis. That improvement is in the development road map for the NTRT toolkit.

2.3.2.2 Touch Sensors

While touch sensors are not currently present on the physical DuCTT robot, they are an improvement that would help it actually navigate HVAC ducts. Touch sensors will provide valuable information about when the linear actuators actually come in contact with the walls in a HVAC duct. Currently, the physical robot estimates contact with a wall via the current driving a linear actuator. When the linear actuator starts stalling, the physical robot decides that it has hit a wall and stops expanding the linear actuator. For this reason, they are added to the simulated version of the DuCTT robot.

In Bullet, there are a few ways of doing collision detection, which is necessary for implementing a touch sensor. These include btGhostObjects, iterating over all contact manifolds, and ray tracing [2]. Ray tracing is a good approximation of real world LIght Detection And Ranging (LIDAR) systems, but does not represent the type of sensor desired for this simulation. Iterating over all the contact manifolds would be done at multiple times during a single simulation step and is therefore not as efficient as the btGhostObject method.

The method that's closest to real world touch sensors and can be efficiently implemented is the btGhostObject method. This method involves adding btGhostObjects to the Bullet dynamics world and attaching them to the rigid bodies that are desired to be touch sensors. The btGhostObjects should have a collision shape that represents the 3D

area desired to be a touch sensor, the easiest way to do that is to make the ghost shape the same as the collision shape of the rigid body that the btGhostObject is attached to. The btGhostObject keeps track of only the other objects that overlap with itself. This greatly limits the number of contact manifolds that need to be iterated over and provides an efficiency that the other contact manifold method does not posses.

The btGhostObject method was chosen as the desired method of adding touch sensors to the simulation. Therefore, btGhostObjects are attached to the spheres at the end of the prismatic joints and given sphere collision shapes to match the physically simulated spheres. In addition, each touch sensor contains a list of objects to ignore when activating. This is done so that the sensors would not activate when they are touching the ground object.

2.3.2.3 Robot Simulation Parameters

The simulated DuCTT robot contains a good amount of parameters that can be varied to explore how they would affect the physical structure of a hardware version or the locomotion capabilities of the robot. These include: how far apart the tetrahedrons are from each other, the height of one tetrahedron, rod radius, rod length, rod density and more. All of these parameters are set to values that are as close to the values for the physical prototype (DuCTTv2, Fig. 2.1a) as possible. Table 2.1 gives a listing of the parameters and their values for this work.

2.3.3 Environment

Three different environments are used for testing. All of them contain a planar ground, and for the third environment, the horizontal plane is all that exists. The second environment contains a simulation of an horizontal air duct created by combining four thin long boxes at their corners (Fig. 2.3b). These boxes extend along one of the two

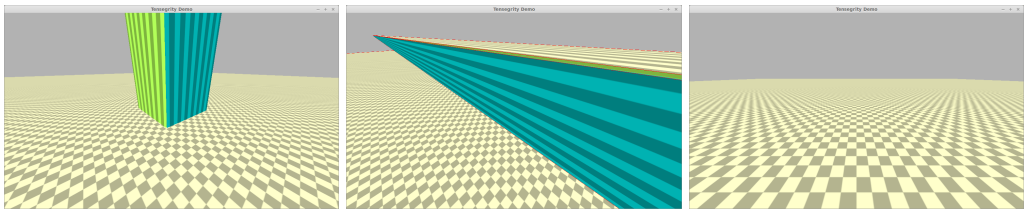
Table 2.1. Robot simulation parameters and their values for this work.

Parameter	Value
triangle_length	32 cm
duct_distance	15 cm
duct_height	23 cm
linActRadius	1.524 cm
linActExtent	10.16 cm
linActDensity	0.001943 $\frac{kg}{cm^3}$
vertRodRadius	1.27 cm
vertRodDensity	0.000895 $\frac{kg}{cm^3}$
innerRodRadius	2.0955 cm
innerRodDensity	0.001359 $\frac{kg}{cm^3}$
tipRadius	1.524 cm
tipDensity	0.00001943 $\frac{kg}{cm^3}$
cableStiffness	5000 $\frac{kg}{s^2}$
cableDamping	25 $\frac{kg}{s}$
maxVertCableVel	25.4 $\frac{m}{s}$
maxSaddleCableVel	8.5 $\frac{m}{s}$
maxCableForce	5000 $\frac{kg\,cm}{s^2}$
minCableRestLength	1.2 cm

horizontal axes (X or Z axes) of the simulation environment. This forms a rectangular duct that can be used to explore traversal strategies for a robot. All of the boxes are static objects in Bullet, meaning they cannot move but they can be interacted with. The first environment contains a vertical duct simulated in the same manner as the horizontal duct, but extending along the vertical axis of the simulation (Y axis).

2.4 Control

All controls development done for this thesis assumes perfect state information is available to the algorithm. This limits the scope of the problem to be solved to just the controls problem and leaves the state estimation work to be done separately. In addition, some state estimation work has already been done by the team at UCSD [18].



(a) Vertical duct **(b)** Horizontal duct **(c)** Horizontal plane

Figure 2.3. Three different environments used for testing control strategies.

There are many contemporary methods for controlling a robot, as well as some developed specifically for tensegrity-based robots [12, 21, 22, 19]. Two options are open-loop and closed-loop controllers. The inputs and outputs available on the robotic system can also help guide the choice of type of controller. Table 2.2 details the inputs and outputs available from the simulated robot of this thesis.

Table 2.2. The inputs and outputs available from the simulated parts of the robot.

Simulated Part	Number Used	Inputs	Outputs
Cable	8	tension rest length	start length rest length current length tension velocity
linear actuator	2	length max velocity of motor max force of motor	length
Touch sensor	4	N/A	contact (yes/no)

Two different strategies are explored for controlling the DuCTT robot. One involves using sine waves to control the linear actuators and groups of cables, the other uses a state machine. Two slightly different controllers are implemented for the first strategy, and one for the second, leading to a total of three different controllers.

2.4.1 Control Strategies

The two strategies for control have two different parameter spaces that determine the degrees of freedom of each controller. This also impacts the space that the machine learning algorithms have to search through for an optimal solution.

2.4.1.1 Strategy 1

The first strategy uses simple sine waves as the control method. This limits the amount of control variables per cable that need to be set compared to CPGs and also simplifies the equations used for control. This scheme has also been used to successfully control the SuperBall tensegrity robot [21]. The output of Equation 2.1 can be used to control the rest length, or velocity of a cable; as well as the desired length of the linear actuator or any other actuator input as desired. This strategy uses the output as the desired velocity of a cable and the desired length of a linear actuator.

$$y(t) = C + A * \sin(\omega t + \phi) \tag{2.1}$$

The sine wave parameters of Equation 2.1 are:

- C: center of sine wave
- A: amplitude
- ω: angular frequency
- ϕ: phase, i.e. where in the cycle the oscillation begins (at t=0)

Once the state space has been defined, there are three main options for how exactly to control the different parts of this robot (Tbl 2.2). The linear actuators are never grouped up because they need to be actuated separately to provide the desired inchworm motion for climbing. If they were in the same group and therefore used the same sine wave, then they would always be at the same length and the robot would not climb

anywhere. Option 1 provides the most stability, as all the vertical cables are grouped together and all the saddle cables are grouped together. This should should prevent the saddle cables from oscillating against each other and destabilizing the robot. It will also keep the vertical cables at the same length, which should provide the most distance per wave period. Option 3 provides the best flexibility with a sine wave for each cable and linear actuator. Options 1 and 3 are both explored in this thesis.

Table 2.3. Control groupings considered.

Option	Cables	Linear Actuators	# Sin Waves	# Parameters
1	2 groups of 4	1 group of 2	4	16
2	4 groups of 2	1 group of 2	6	24
3	8 groups of 1	1 group of 2	10	40

There are two options for how to make the touch sensors affect the sine waves controlling the robot. Either pause the sine wave when the touch sensor is triggered, or saturate the sine wave when the touch sensors are triggered. The second option seems like it might result in better performance so that is the option implemented. In this second option, when both touch sensors connected to the bottom tetrahedron trigger, the controller saturates the sine wave associated with the linear actuator on the bottom tetrahedron. This means that all control input for that sine wave is ignored, but the wave still oscillates. Then, when both touch sensors connected to the top tetrahedron trigger, the controller unsaturates the sine wave on the bottom linear actuator and saturates the sine wave on the top linear actuator.

In addition, some hysteresis is added to the touch sensors in the form a small amount of lag between when they first activate and when they actually saturate the sine wave. This takes the form of another parameter (τ) that indicates how many seconds to lag. Once that time period has passed, then the sine wave is actually saturated.

2.4.1.2 Strategy 2

The second strategy was developed because the first one was not having much success in getting the robot to climb. It would climb with the first strategy, but it was very slow and the movement was vibration-like. This strategy towards controlling the DuCTT robot also takes advantage of the fact that its climbing behavior is very periodic. However, instead of using sine waves to recreate that periodicity, it uses a state machine and cycles through a set of six states:

1. EXPAND_BOTTOM
2. RETRACT_TOP
3. PUSH_TOP
4. EXPAND_TOP
5. RETRACT_BOTTOM
6. PULL_BOTTOM

Each state actuates either the cables or the linear actuators, but not both. When moving the linear actuators, the same hysteresis affect on the touch sensors used in strategy one is used in this strategy. However, the desired length for a linear actuator at each time step is not the output of Equation 2.1. Instead, this strategy sets it to the length that can be reached by moving the linear actuator at maximum velocity.

When actuating the cables, there are a total of four parameters for this strategy. This includes the τ parameter for determining how many seconds to lag saturation behind the touch sensor activation. One (μ) determines the minimum length in centimeters cables should attempt to retract to when attempting to pull the tetrahedrons closer together. Another (η) determines the maximum length (cm) cables should attempt to reach when pushing the tetrahedrons away from each other. The last (ε) determines the maximum difference (cm) between actual length and goal length that will be considered reaching

the goal for cables.

State 1 expands the linear actuator on the bottom tetrahedron all the way or until it hits a duct wall. This will lock the bottom tetrahedron to the duct walls. State 2 retracts the top linear actuator to its minimum size, thereby releasing the top tetrahedron from the duct. Then state 3 pushes the top tetrahedron away from the bottom until the cables reach their maximum length goal, at which point state 4 expands the top linear actuator the same way that state 1 does the bottom linear actuator. This will lock the top tetrahedron to the duct, after which state 5 retracts the bottom linear actuator in the same manner as the top one is retracted in state 2. During state 6, the cables are actuated so that the bottom tetrahedron is pulled up towards the top. This occurs until the cables reach their minimum goal length. The state machine then repeats the sequence starting at state 1.

2.4.2 Controllers

Controllers one and two use the first strategy, and controller three uses the second. In controller one, each grouping contains a separate set of the sine wave parameters from Equation 2.1. The number of parameters for controller one is the same as in Table 2.3. In controller two, the angular frequency (ω) for all sine waves is a single parameter which helps keep the cables and linear actuators from canceling each other's movements out. Controller two has 14 total parameters for option 1 (12 for sine waves, 1 for hysteresis, 1 for angular frequency), and 32 total parameters for option 3 (30 for sine waves, 1 for hysteresis, 1 for angular frequency). Controller three uses the second strategy with no modifications.

2.4.3 Low-level Controllers

Along with this, all the cables are given an impedance controller by default so that they maintain their tension. This low-level reflexive control helps maintain the tension in

the cables and prevents cables from going slack often. This impedance controller uses the same equation as the TetraSpine robot does (Eq. 1.5, reproduced here for convenience) to determine what tension to set the cable to.

$$T = T_0 + K(L - L_0) + B(V - V_0) \tag{2.2}$$

2.5 Mechanical Test and Validation

The aim of this experiment is to test and validate the mechanics and dynamics of the model of the Duct Climbing Tetrahedral Tensegrity (DuCTT) robotic platform built in the NTRT Library. It does this by comparing the control of the NTRT model against an Euler-Lagrangian (Lagrangian) model over a series of test cases. Lagrangian dynamics are often used to model the dynamics of robots and robotic manipulators ([28, 35, 14]). The Euler-Lagrangian (Lagrangian) model will provide proof that the NTRT model is mechanically sound and can be used to study control of the Duct Climbing Tetrahedral Tensegrity (DuCTT) robot.

2.5.1 Lagrangian Model

This simulation model uses Lagrangian mechanics (a specific use of the Euler-Lagrange equation) to obtain equations of motions from the calculated potential and kinetic energy. The are 5 point masses for each tetrahedron with offsets from the center

of the tetrahedron defined as follows:

$$\mathbf{t}_1 = (0,0,0)^T$$
$$\mathbf{t}_2 = (0,-l/2,-h/2)^T$$
$$\mathbf{t}_3 = (0,l/2,-h/2)^T$$
$$\mathbf{t}_4 = (-l/2,0,h/2)^T$$
$$\mathbf{t}_5 = (l/2,0,h/2)^T$$

Euler angles determine each tetrahedron's orientation in space; the angles are defined as θ, γ, and ϕ for rotations about the x, y, and z axes respectively. The rotations are applied with rotation matrices in the order defined as:

$$R(\theta,\gamma,\phi) = R(\phi) * R(\gamma) * R(\theta)$$
$$\mathbf{r}_1 = (x,y,z)^T$$
$$\mathbf{r}_2 = \mathbf{r}_1 + R(\theta,\gamma,\phi)\,\mathbf{t}_2$$
$$\mathbf{r}_3 = \mathbf{r}_1 + R(\theta,\gamma,\phi)\,\mathbf{t}_2$$
$$\mathbf{r}_4 = \mathbf{r}_1 + R(\theta,\gamma,\phi)\,\mathbf{t}_2$$
$$\mathbf{r}_5 = \mathbf{r}_1 + R(\theta,\gamma,\phi)\,\mathbf{t}_2$$

The rotation matrices are applied before translation occurs, so $R(\theta,\gamma,\phi) * \mathbf{t}_1 = 0$. If each node is defined to have equal mass and spacing, $\mathbf{r}[1]$ is located at the center of mass; this causes decoupling between rotational and kinetic energy, which is highly desirable for simplicity of equations. This occurs because the linear actuator is closed for all trials. Directly computing rotational inertias or angular velocities is not necessary

because the model is a point mass model, and a rotation of angle will cause a shift in the nodal velocities.

Kinetic (T) energy, mechanical potential (V) energy, and the Lagrangian (L) are defined as follows:

$$T = \frac{1}{2} m \sum_{i=1}^{5} \left(\frac{d}{dt} (\mathbf{r}_i)^T * \frac{d}{dt} (\mathbf{r}_i) \right) \tag{2.3}$$

$$V = mg(0,0,1) \sum_{i=1}^{5} (\mathbf{r}_i) \tag{2.4}$$

$$L = T - V \tag{2.5}$$

Then L is plugged into the Euler-Lagrange equation to get the equations of motion. Note the spring potential energy is not included in the V equation. Instead, the spring tension forces will be applied using the concept of generalized forces, so that the uni-directional forces (strings cannot push) can be handled and cable damping can be added.

The string anchor points are defined as follows, (where 0.015 corresponds to the 1.5 cm radius of the linear actuator and 0.02 to the 2 cm radius of the midbar). $\alpha = \frac{\pi}{6}$ is the angle from the vertical axis of the cross section of the midbar rod to the mounting point.

$$anch\_bot[1] = t2 + [0;0;0.015]$$

$$anch\_top[1] = r1 + R * (t2 + [0;0;-0.02])$$

$$anch\_bot[2] = t3 + [0;0;0.015]$$

$$anch\_top[2] = r1 + R * (t3 + [0;0;-0.02])$$

$$anch\_bot[3] = t4 + [0;0;0.02]$$

$$anch\_top[3] = r1 + R * (t4 + [0;0;-0.015])$$

$$anch\_bot[4] = t5 + [0;0;0.02]$$

$$anch\_top[4] = r1 + R * (t5 + [0;0;-0.015])$$

$$anch\_bot[5] = t4 + [0;-\sin(\alpha);-\cos(\alpha)] * 0.02$$

$$anch\_top[5] = r[1] + R * (t2 + [-\sin(\alpha);0;\cos(\alpha)] * 0.02)$$

$$anch\_bot[6] = t5 + [0;-\sin(\alpha);-\cos(\alpha)] * 0.02$$

$$anch\_top[6] = r[1] + R * (t2 + [\sin(\alpha);0;\cos(\alpha)] * 0.02)$$

$$anch\_bot[7] = t4 + [0;\sin(\alpha);-\cos(\alpha)] * 0.02$$

$$anch\_top[7] = r[1] + R * (t3 + [-\sin(\alpha);0;\cos(\alpha)] * 0.02)$$

$$anch\_bot[8] = t5 + [0;\sin(\alpha);-\cos(\alpha)] * 0.02$$

$$anch\_top[8] = r[1] + R * (t3 + [\sin(\alpha);0;\cos(\alpha)] * 0.02)$$

For all bottom anchor points the bottom tetrahedron is fixed so rotation matrices etc. are not necessary. The string lengths, tensions and vector forces are calculated as in Algorithm 1.

Algorithm 1. String lengths, tensions and forces

for i=1:8 **do**
    String_vector[i] = anch_bot[i] - anch_top[i]
    length[i] = norm(String_vector[i])
    if Tension[i] ¿ 0 && length[i] ¿ restLength[i] **then**
        Tension[i] = $K * (length[i] - restLength[i]) - c * \frac{d}{dt}(length[i])$
    else
        Tension[i] = 0
    end if
    F[i] = Tension[i] * (String_vector[i])
end for

The string lengths, tensions and vector forces are used to compute the generalized forces for the coordinate system:

$$Q_j = \sum_{i=1}^{8} \left(\frac{\mathrm{d}}{\mathrm{d}q_j} (anch\_top[i])^T * F[i] \right) \tag{2.6}$$

The subindex j represents cycling through the chosen coordinates, in this case $x, y, z, \theta, \gamma, \phi$.

Generalized equations of motion:

$$Q_j = \frac{\mathrm{d}}{\mathrm{d}t} \left(\frac{\partial T}{\partial \dot{q}_j} \right) - \frac{\partial T}{\partial q_j} \tag{2.7}$$

Then Eq. 2.6 is substituted into Eq. 2.7 which results in a set of six equations. This system of six equations is solved to get equations for each coordinate's second derivative and then these equations are used to step the simulation forward using the predictor-corrector method twice. Once per time step to get the first derivative of the

generalized coordinates, then a second time to step forward the actual coordinates.

2.5.1.1 Matching Lagrangian to NTRT

Upon initial testing of the Lagrangian model, several discrepancies were discovered which were preventing the two models from being in agreement. After extensive tuning and trial and error, the greatest contributor to model disagreement was found to be a difference in cable mounting locations. The NTRT model assumed all cables were mounted along the perimeter of the ends of the rod elements while the Lagrangian model had them fixed at the rod center points. This equated to a discrepancy of only roughly a centimeter error, but upon moving the cable mount points in the Lagrangian model to the proper locations a dramatic increase in model agreement occurred.

A second smaller contributor to model disagreement was the method for checking the slack cable event. Instead of just checking for negative tension, the Lagrangian model had to check to see if the rest length was greater than the length between nodes which would more accurately describe a slack string.

2.5.2 Control

The model agreement tests focus on inducing dynamic excitations with the cable elements and not the linear actuator, which is fixed at a given length. This decision was made because the actuation bandwidth of the linear actuator within the physical prototype is low when compared with the bandwidth of dynamics of the robot, and its mechanical stiffness is orders of magnitude larger than the cable elements. As such, an accurate dynamic model of the linear actuator is not necessary to produce a reasonably accurate model of the system dynamics. Therefore, the effectiveness of this test is not greatly reduced by simply locking the linear actuator into a single position during the dynamics comparison testing, as it generally behaves as a rigid member regardless.

Sine waves control the rest lengths of the cables with the following parameters:

- l: offset length (cm)
- C: center of sine wave (cm)
- A: amplitude (cm)
- Ω: angular frequency (Hz)
- p: phase, i.e. where in the cycle the oscillation begins (at t=0) (rad)

This control is all done to the NTRT model using the following equations, with target being the target rest length of the cable:

$$cycle = \sin\left((timestep) * \Omega + 2 * C * \pi + p\right) \tag{2.8}$$

$$target = l + cycle * A \tag{2.9}$$

$$if\,(target < l)\, target = l \tag{2.10}$$

At every timestep, the NTRT controller output the rest lengths of each cable as well as the $< x, y, z >$ positions of each node. These outputs were then given to the Lagrangian model.

In the Lagrangian model, at each time step the rest lengths provided from the NTRT data are used to set the rest length variables in the Lagrangian simulation. The rest lengths are the only input to the actual Lagrangian model.

The Lagrangian model uses the nodal coordinates to compare the state of the NTRT model to the state of its own model. It uses these nodal coordinates to back out the generalized coordinates described above. First, it calculates 8 lengths corresponding to where the 8 strings would be if they stemmed directly from the nodes; it then uses a process similar to trilateration.

This process is used because a tool already exists to use this method, and it works

quite well. It was built for backing out the robot's state given the absolute string lengths. The problem with trilateration is the solution is often ambiguous between two points and exactly 3 lengths are used to determine a location. Instead this process uses a non-linear least squares optimization to do the job. Essentially, it computes a function that describes the 8 string lengths in terms of the 6 generalized coordinates $(x, y, z, \theta, \gamma, \phi)$; then it also computes the Jacobian of this function. After that it just uses this function to fit a state variable to each time step of the NTRT simulation using a non-linear least squares optimization.

2.5.3 Tests

During all tests, the bottom tetrahedron of the DuCTT model is artificially constrained to a static position. This is done to simplify the testing and the Lagrangian model. Six tests are conducted on the models to validate the dynamics of the NTRT model (2.4). All tests lasted for 30 seconds, with no actuation during the first 5 seconds to allow the NTRT model to settle into a stable position. During all tests, if the target rest length went below an offset length, then the target rest length was set to the offset length. This prevents the rest lengths from going negative, as well as prevents collisions from occurring between rigid bodies, since the Lagrangian model was not set up to handle those.

The first test consists of actuating all the cables using the same sine wave. The second test consists of actuating the top two vertical cables of the DuCTT model with the same sine wave, but at different offsets ($p-1$ and -2 in 2.4). This leads to an oscillation motion in the top tetrahedron that tests more of the model dynamics than the first test. The third test is a lot like the second, only it actuates the bottom two vertical cables. The last three tests are duplicates of the first three, except instead of using a single frequency for the sine wave, the tests sweep over a range of angular frequencies. These frequencies

started at 10 and increased to 50 in steps of 5, with each frequency running for 3 seconds.

Table 2.4. The sine wave parameters used for each test case.

Parameter	Case 1	Case 2	Case 3
l	3	3	3
C	1	1	1
A	5	15	10
Ω	50	50	50
$p-1$	$\frac{\pi}{4}$	0	0
$p-2$	-	$\frac{\pi}{2}$	$\frac{\pi}{2}$

2.6 Learning Algorithms

The NTRT library has support for an evolution scheme which can use Monte-Carlo, Gaussian sampling, co-evolutionary and genetic algorithm machine learning methods [4]. This work uses the Monte-Carlo parameter estimation to find the first set of good parameters. In past works, Gaussian sampling was then used to build on the first set and find better sets without needing to compute a gradient [25]. However, seeding a genetic algorithm with the set of parameters generated by the Monte-Carlo method and running that produced better performing parameters for this work.

Both methods use algorithms included in the NTRT library and are available online [3]. For all methods, each run produces a final set of parameters after a number of trials, where the performance of a trial is defined by the cost function (Sec. 2.7). Each trial gives the robot a specific number of seconds to do what it can in the environment, then measures the performance with the provided cost function.

The parameters in both methods vary between 0 and 1, but this is just a standardized range for the algorithm to search through. The controller then takes that value and transforms it into a range defined for that controller. Each parameter has a different

Table 2.5. Value ranges for each parameter of the two controller strategies.

Strategy	Parameter	Min Value	Max Value
1	C	0	40
1	A	0	40
1	ω	0.01	20
1	ϕ	$-\pi$	π
1,2	τ	0	2
2	μ	0	10
2	η	5	20
2	ε	0	5

range of values that are hand picked to allow the best possible movement (Tbl. 2.5). The parameters for strategy one will be replicated, with different values, for each cable based on the grouping option picked.

2.6.1 Monte-Carlo

The Monte-Carlo algorithm is multi-level and brute-forces the best parameters for the controller given the cost function, robot and environment over many trials. To initialize, it selects the value of each parameter for a controller from a uniform distribution between zero and one. It also mutates all controllers in the population after each generation. A controller mutates by drawing a new value for parameters in the same way they are initially selected. The number of parameters that are changed when a controller mutates is defined by the number of controllers to mutate hyperparameter.

2.6.2 Genetic Algorithm

Genetic algorithms are based on the natural selection process, and are used to find solutions to optimization problems. They involve evaluating a population of potential solutions and mutating or breeding the most promising ones together. Each member is evaluated based on the desired fitness function. In addition, each member of the

population is defined by a genome; this genome can be a series of bits, a series of numbers, or even a series of letters. The fitness function is the main driver of complexity for genetic algorithms, and having a simple function helps limit the time required to complete the algorithm. Genetic algorithms have been used to order sets of DNA fragments, predict protein structure [26], and to evolve controllers for tensegrity robots [21]. They have also been used to evolve the network architecture of a neural network [26].

The NTRT genetic algorithm run to establish a second set of better controller parameters based on the set generated by Monte-Carlo is a standard genetic algorithm [26]. This algorithm evolves, via an iterative process, a population of possible solutions to an optimization problem into better solutions. Each member of the population contains a set of doubles between 0 and 1 that make up its genome. If the population is seeded, the genome of one member of the population takes on the values of the seed genome. This genome can be mutated at the end of a generation, where a generation consists of the population of one iteration. Before any mutations occur, the fitness of each member is evaluated using the provided cost function (Sec. 2.7). This evaluation determines which individuals are the most fit and should be used for recombination and mutation. For this work, each trial of a run tests a single member of a population which means that the maximum number of generations is determined by the number of trials to run and population size.

At the end of a generation, two processes occur depending on the values of the hyperparameters number of children, and number of controllers to mutate. The number of children hyperparameter determines how many population members of the next generation to generate via uniform crossover. Uniform crossover uses two population members of the last generation as parents to make a single population member (the child) for the new generation. These parents are chosen from the last generation randomly, and

the child's genome is made from half the parameters of each parent. In addition, the child has a 10% chance to mutate in the normal manner (when combined with the 50% chance of mutation below this leads to a total 5% chance to mutate).

The number of controllers to mutate hyperparameter determines how many members of the last generation to mutate in the normal manner. Each controller has a 50% chance to actually mutate, and each gene of that controller's genome has a 50% chance to mutate. If a gene is actually mutating, then the mutation amount is drawn from a normal distribution with a mean of 0 and a standard deviation of 0.03. This mutation amount is then added to the value of the gene to form the new gene.

2.6.3 Other learning approaches

There are a variety of modern learning approaches and derivative-free optimization algorithms not considered in the present work, including boosting methods and generalized pattern searches, that would be worthy to consider in future work, and ultimately to include in NTRT itself.

Boosting methods search an ensemble of algorithms for a heuristic that can solve an optimization problem sufficiently. They involve combining weak learners iteratively into a final strong learner. When combining, weights are used to take into account each weak learner's accuracy. AdaBoost is a boosting algorithm that can adapt to the weak learners as they are iterated through [17]. In other words, successive weak learners are formulated to concentrate more on instances misclassified by previous weak learners.

Another derivative-free optimization algorithm is Generalized Pattern Search (GPS). This is a type of pattern search that involves exploratory movements in parameter space [8]. These movements take place on a lattice of points in parameter space, commonly a Cartesian grid. The search begins by using a guess in parameter space to identify a local minimum of a function. Typically a model of the actual function of interest is

then constructed that is inexpensive to compute and differentiable to help identify the function's global minimum. This model is used to identify the trends in the available data over the feasible region in parameter space.

2.7 Cost Functions

Two different cost functions are compared to determine which gives the best performance. The first cost function is the total distance that the robot's center of mass moved (Eq. 2.11). The center of mass is calculated by averaging the center of mass of all the rods that make up the DuCTT robot (from both bottom and top tetrahedrons). This is a basic cost function used in many machine learning algorithms for tensegrity robots [22, 25, 29]. For the two duct navigation tasks it is advantageous to only consider movement along the axis of the duct being explored. However, for the open plane traversal true distance in all three dimensions is used so as to not limit the algorithm needlessly.

$$\text{dist} = \begin{cases} \Delta y, & \text{if duct-axis} = y \\ |\Delta x|, & \text{if duct-axis} = x \\ |\Delta z|, & \text{if duct-axis} = z \\ \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}, & \text{if horizontal plane} \end{cases} \tag{2.11}$$

$$\Delta x = x_t - x_{t_0}$$

$$\Delta y = y_t - y_{t_0}$$

$$\Delta z = |z_t - z_{t_0}|$$

The second cost function attempts to smooth out the movement of the robot while maintaining a high speed. This cost function is only used to try to improve the performance of a controller that can already successfully make the DuCTT robot climb. It does this by taking into account both the average speed of the robot, along with the maximum and minimum instantaneous speed of the trial (Eq. 2.12):

$$\text{CoIS} = \bar{s} + \left(\frac{1}{\max(s_t) - \min(s_t)} \right) \tag{2.12}$$

$$\bar{s} = \text{dist}/\Delta t \tag{2.13}$$

where s is instantaneous speed, $\bar{s}$ is average speed calculated by dividing total displacement by total time. The second term comparing the maximum and minimum instantaneous speeds will be close to zero when the maximum is much greater than the minimum. It will then be maximized when the max and min speeds are as close to each other as possible. The chances of the two being exactly equal are are not significant, especially when the main component of this cost function is the average speed.

2.8 Acknowledgments

I would like to acknowledge Jeffrey Friesen for his work on the Lagrangian dynamics model of the robot and his contributions to Section 2.5.

Chapter 3

Experimental Results

Here we detail the results of the mechanical validation experiment as well as the learning algorithm optimization of the DuCTT controllers. The mechanical validation experiment shows that the NTRT simulation of the DuCTT robot is robust when compared to an Euler-Lagrange model. It also shows that some details that were originally not thought to have a big impact on the results of this validation actually do. The results of the learning algorithm optimization show that controller strategy 1 does not perform as well as strategy 2 (the state-machine based controller).

3.1 Mechanical Validation

After the first test was run, a high discrepancy between the models was found (Fig. 3.1, 2cm mounting error). After investigating the issue, the mounting location of the cables were discovered to have caused the discrepancy. The NTRT model of DuCTT located the cable mount points on the circumference of the rods, while the Euler-Lagrange model located them exactly at the node of the rod. This translated into a 2cm mounting error between the two models. Before the test was run, this inconsistency was not thought to have this much of an impact on the results; however, these results (Fig. 3.1) disprove that notion. Once this inconsistency was fixed by moving the Euler-Lagrange cable mount locations to the circumference of the bars, the discrepancies mostly disappear and

the state variables of each model are brought into close agreement.

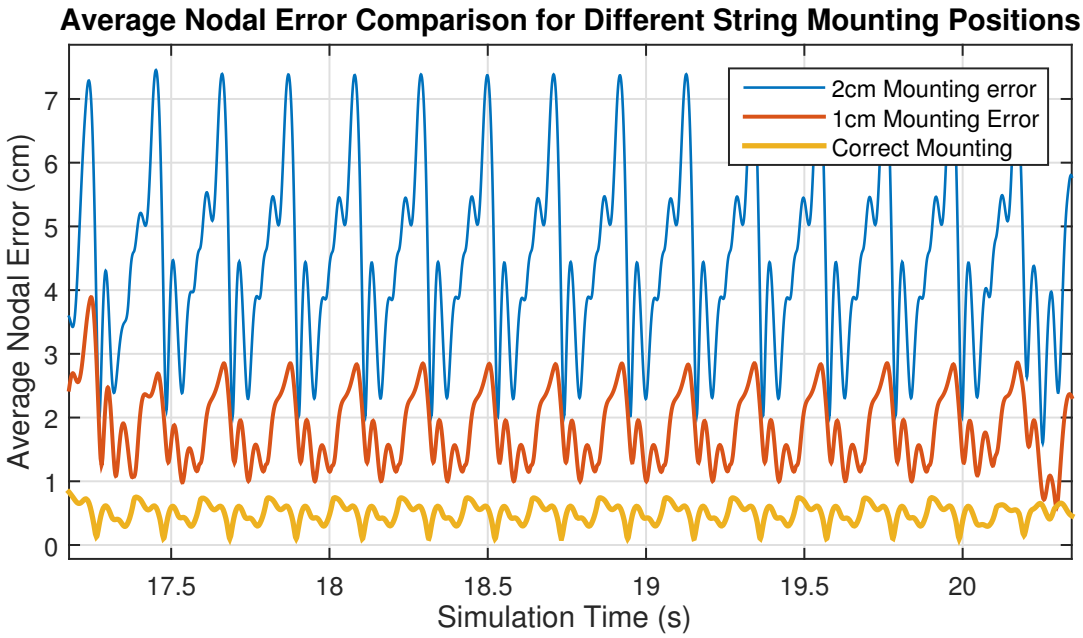


Figure 3.1. Average nodal error between the models. Calculated by averaging the distance between all the nodes of the two models.

After fixing the inconsistencies between the models, average nodal error is mostly < 0.005 m, but reaches as high as 0.01 m in test cases 4 & 5 (Fig. A.10 & A.13) both of which involve the frequency sweep. The change between frequencies may explain the relatively high average nodal error in those test cases. Whenever the controller changes frequencies, there's a higher than normal deviation in the robot's motion that may be the cause of the high nodal errors.

In test case 2, the y, θ, and ϕ state variables from the NTRT model are essentially flat (Fig. A.5), whereas the Lagrangian model exhibits oscillations in value. However, looking at the comparison of the ϕ state variable portraits between the two models shows very similar looking tracks (Fig. A.6). So even though on first glance it doesn't look like the state variables are tracking closely, the two models are actually similar in behavior.

In addition, the state variables from both models track each other closely most

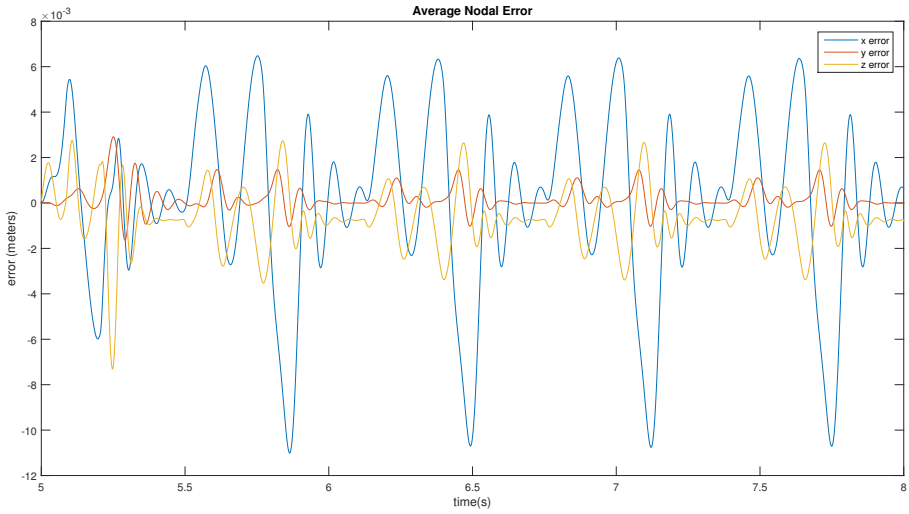


Figure 3.2. Test case 5: average nodal error

of the time (ex. Fig. 3.3) with the most consistent exception being the ϕ variable. Even that variable is within 4×10^{-5} radians of each other in Fig. 3.3. And for most of the test cases, the two models have similar tracks for the ϕ state variable (Apdx. A.1).

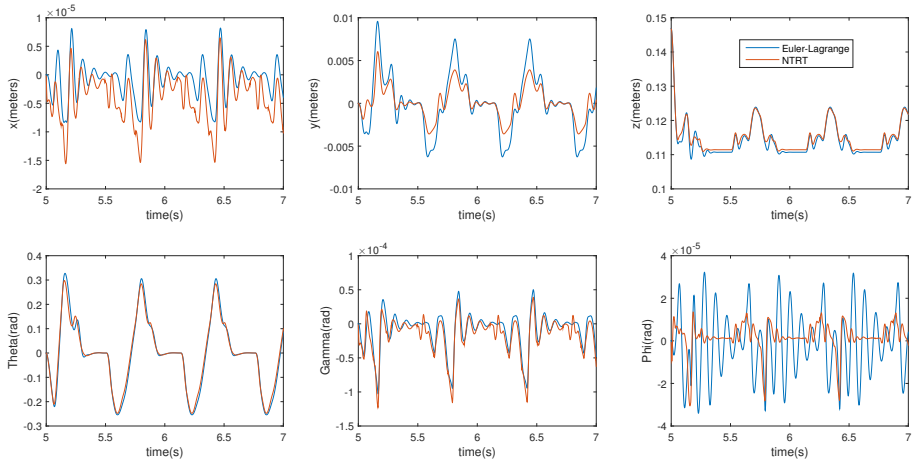


Figure 3.3. Test case 6: state variable comparison

Most of the differences in the Lagrangian model results occur in the same frequency range as the NTRT model results or are insignificant mechanically. These results

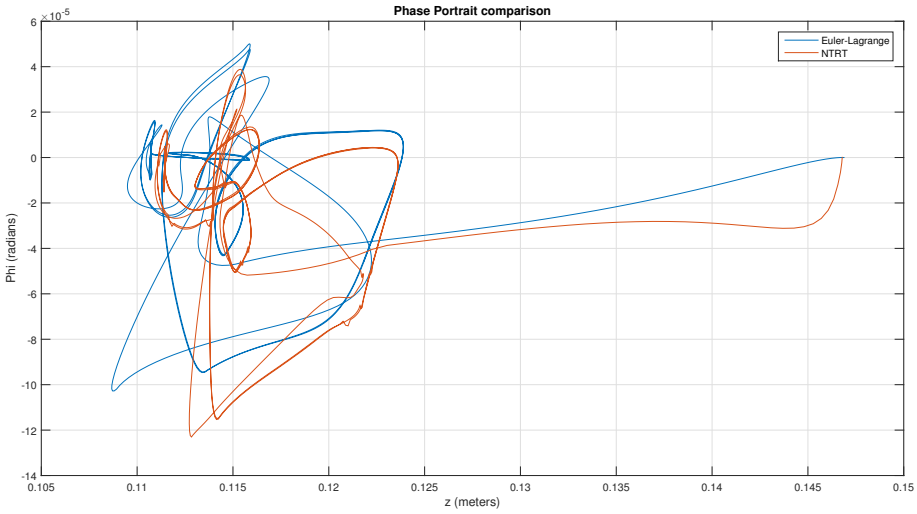


Figure 3.4. Test case 6: Comparison of ϕ state variable

indicate that the two models are mathematically similar, which tells us that the NTRT model is mechanically sound. For the full set of results see Appendix A.1.

Overall, the Lagrangian model matched the NTRT model quite well. This tells us that the NTRT model is mechanically sound and accurately simulates the robot. While there were some small differences in the models, these are mostly attributable to the differences in setup or control of the models. Therefore, using the NTRT model for research and development of control methods is mechanically sound. Any methods developed on the NTRT model should translate to the actual physical robot accurately.

3.2 Learning Results

To optimize the parameters for the three controllers, two stages of learning are performed. During both stages, each trial lasts 60 seconds. The first three seconds of each trial are used to let the simulated robot reach a stable starting position, and the last 57 seconds are movement time. Stage 1 consists of the Monte Carlo parameter search for 20,000 trials and Stage 2 consists of the genetic algorithm seeded with the

best performing parameters from stage 1. The Stage 2 hyperparameter population size is set to 10, number of children is set to 2, and number of controllers to mutate is set to 1. The first environment (vertical duct) is used to compare the performance of the three controllers. The two other environments (horizontal duct and horizontal plane) are used to explore the performance of the best controller. For both environments featuring an air duct, the robot is rotated such that the end-caps of the linear actuators are stuck in the corners of the duct (Fig. 3.5). This provides the friction required to climb, and better performance in the horizontal duct. In addition, for these two environments, the duct size is 33×33 cm unless otherwise stated.

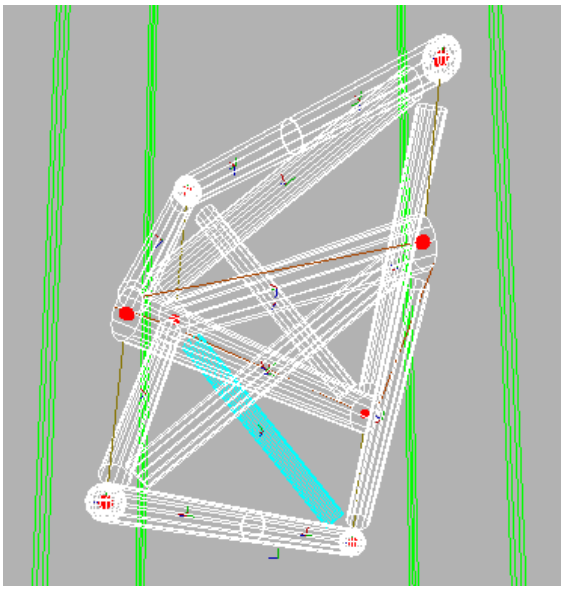


Figure 3.5. Starting position of simulated DuCTT. Green lines are the corners of the duct.

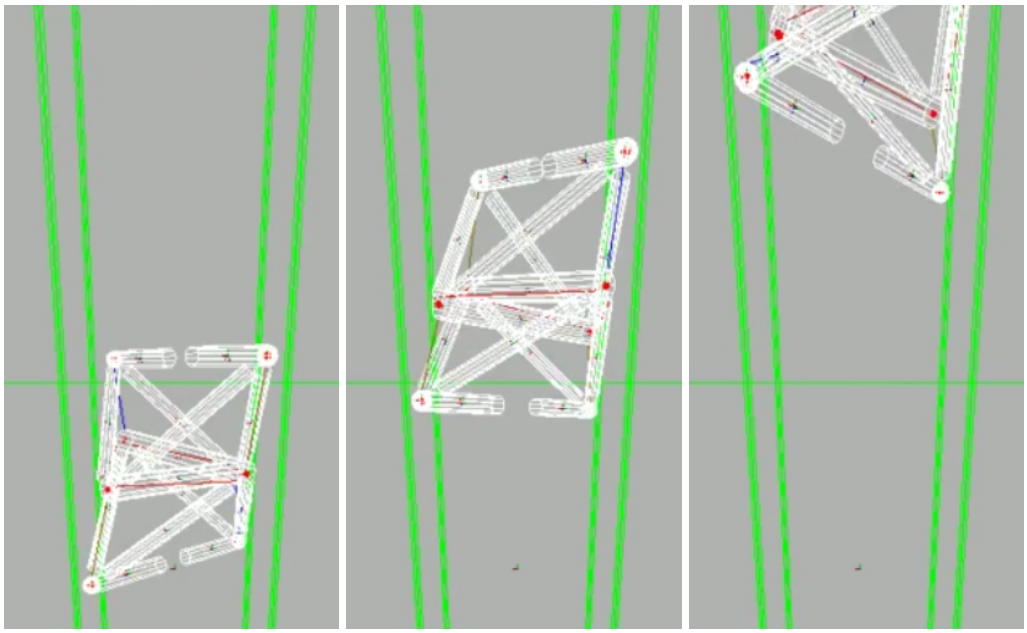
3.2.1 Vertical Duct Climbing

For the vertical duct climbing task, all three controllers are explored. Controller 1 with both options 1 and 3, controller 2 with option 3 and controller 3 with both cost functions. Controllers 1 and 2 are only learned using the first cost function. Controller

Table 3.1. Performance of each controller at climbing vertical duct after learning.

Controller	Speed ($\frac{cm}{s}$)	Distance per minute (m)
1, Option 1	0.38	0.228
1, Option 3	0.55	.318
2, Option 3	0.0089	0.497
3, cost 1	6.1	3.66
3, cost 2	6.77	4.06
DuCTTv2	1.4	.84

3 performs the best out of the three providing a maximum of 17x the performance of the worst performing controller (Tbl. 3.1). In addition, it gets about a 4x speedup over the best speed the physical robot can currently climb at using it's Inverse Kinematic based controller. The learned parameters for controller 3 have the following values: $\tau = 0.002391$ seconds, $\mu = 3.77$ cm, $\eta = 18.34$ cm, and $\varepsilon = 4.06$ cm.



(a) t = 0 seconds **(b)** t = 5 seconds **(c)** t = 10 seconds

Figure 3.6. Controller 3 making DuCTT climb.

3.2.2 Horizontal Duct Traverse

For the horizontal duct traverse, two methods were tried for control. The first is setup much like the horizontal plane traverse, with the bottom tetrahedron resting on the ground and the linear actuators perpendicular to the duct walls. However, this setup did not perform very well, with performance similar to that of the horizontal plane (on the order of .8 $\frac{\text{cm}}{\text{s}}$). The next method rotates the robot much like in the vertical duct climbing task so that the ends of the linear actuators are located in the corners of the ducts. In this case, the weight of the robot would be resting on the two end-caps on the bottom corners of the duct. Using controller 3, a speed of 5.19 $\frac{\text{cm}}{\text{s}}$ is achieved after the two stages of learning resulting in a displacement of 3.11 meters over a minute. The learned parameters for controller 3 have the following values: $\tau = 0.00$ seconds, $\mu = 2.54$ cm, $\eta = 18.82$ cm, and $\varepsilon = 3.26$ cm.

3.2.3 Horizontal Plane Traverse

Neither controller 1 (option 1) nor controller 3 could get very good at traversing a horizontal plane. The performance seems to cap at around .7 to .8 $\frac{\text{cm}}{\text{s}}$. Controller 3 with cost function 2 performed the best, moving at a speed of .84 $\frac{\text{cm}}{\text{s}}$ and covering a distance of 0.5 m in a minute. The poor performance might be occurring because the geometry of the DuCTT robot doesn't provide many feet-like structures to push off the ground with. Without more friction between the robot and ground it's hard to get moving very fast. The learned parameters for controller 3 have the following values: $\tau = 0.97$ seconds, $\mu = 8.07$ cm, $\eta = 13.79$ cm, and $\varepsilon = 4.35$ cm.

3.2.4 Robustness Testing

All robustness testing is done in the vertical duct environment (Fig. 2.3a) using the final learned parameters for controller 3, using cost function 2.

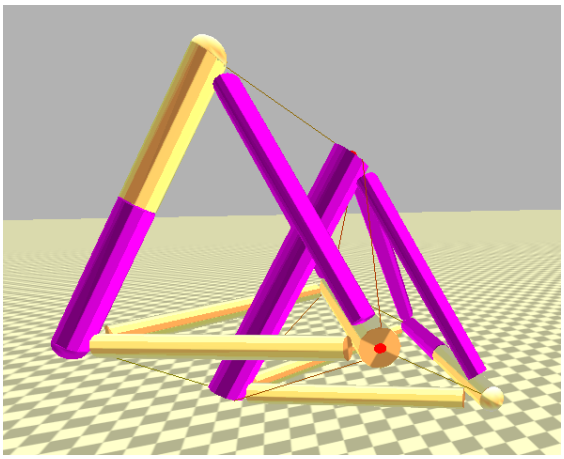


Figure 3.7. DuCTT on a horizontal plane.

3.2.4.1 Starting Position Robustness

This test explores the relationship between starting location in the duct and final performance. It does this by varying the starting location of the robot over the grid in Figure 3.8. Even though the duct itself is 33 cm on a side, because of the size of the robot the starting location could only vary 5 cm from the center of the duct. If the starting location was over 5 cm from the center, the robot would bounce off the walls of the duct towards the center and the starting location would not be deterministic.

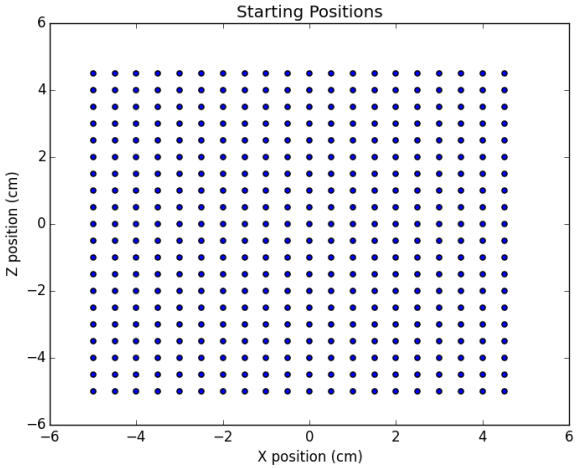


Figure 3.8. Starting positions of positional robustness test. (0,0) is the center of the duct.

The performance of controller 3 with its final learned parameters did not vary much based on starting location. The robot's speed was $6.28 \pm 0.34 \frac{\text{cm}}{\text{s}}$ over all locations and the distance covered was 3.58 ± 0.20 m. This shows that the performance of the controller does not depend on starting exactly in the center of the duct.

3.2.4.2 Duct Size Robustness

Another test of robustness is varying the size of a duct wall. For this test, the controller is tested on ducts of size 25 cm to 35 cm. Since the robot is rotated into the corners, that means that the diagonal of each duct varied from 35 cm to 49.5 cm. Given the compressed length of a linear actuator with touch sensors is 35 cm, and the extended length is 45 cm, this should span the widths of ducts that can be negotiated by DuCTT.

As expected, the robot is not able to traverse ducts where the diagonal is larger than the extent to which the linear actuators can extend (sizes 34cm and above, Fig. 3.9). This is because there is nothing to push against and lock a tetrahedron to the wall in the inchworm motion.

In addition, when the duct is smaller or exactly the same size as the robot (sizes 25 and 26 cm), the performance of the controller varies wildly since the collisions between the robot and the duct walls will occasionally overwhelm Bullet. At 27cm, the duct may still be too small for the robot to navigate as seen by the large standard deviation on both speed and distance. However, in the sweet spot (sizes 28cm to 33cm) the performance of the controller is generally good.

3.2.4.3 Parameter Robustness

The experiment tests the ability of the controller to withstand small variations in the values of its parameters. It is run for 100 iterations, with each iteration starting from the final controller 3 learned parameters and giving the controller 57 seconds to climb the robot as far as it can. In each iteration, all four controller parameters are varied

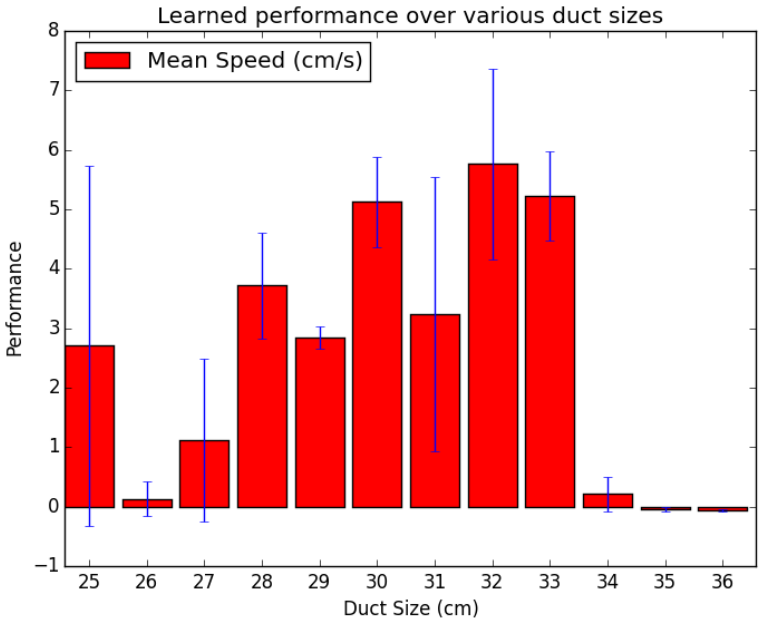


Figure 3.9. Results of duct size robustness test.

by adding to them an amount drawn from a normal distribution with a mean of 0 and standard deviation of 0.5%.

The results show that while these variations in parameters do affect the quality of the controller's performance (as expected), they do not prevent it from climbing altogether. The mean speed after 100 iterations is $3.34\frac{\text{cm}}{\text{s}}$ with a standard deviation of $1.17\frac{\text{cm}}{\text{s}}$. The mean distance traveled 57 seconds is 190 cm $\pm$ 66.9 cm. Therefore, while getting the right parameter values is important for maximizing performance, it does not affect the ability of the controller to do its job.

3.3 Acknowledgments

Section 3.1, in part, has been submitted for publication of the material as it may appear in The Second Generation Prototype of A Duct Climbing Tensegrity Robot, DuCTTv2, in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International

Conference on, 2015, J. Friesen, P. Glick, M. Fanton, P. Manovi, A. Xydes, and T. Bewley. The thesis author was a co-author of this paper.

Chapter 4

Conclusion

This work has shown that it is possible to get good climbing performance out of a well designed robotic controller tuned via machine learning, as well as the validity of simulating a tensegrity robot via an open-source physics engine designed for video games. The robustness of the learned controller parameters was then shown over air ducts of various size and different starting positions.

Controller 3 using strategy 2 (a state-machine) performs the best out of the controllers and strategies tried. While strategy 1 (sine waves) was thought to be good because of the previous literature using it or a similar strategy (CPGs), this work has shown that for the DuCTT robot a higher level control strategy will perform better. This might be due to the geometry of the robot, with more rods connected by joints than some of the other tensegrity robots. The state-machine might be able to account for that better than assigning sine waves to each cable. Or it might be that the desired inch-worm motion is easier for the state-machine controller to discover than the ones based on sine waves.

Now that basic climbing and horizontal traversal have been solved, further research could explore navigating an air duct right-angle or T-junction. This might require designing a new controller, but with the framework developed for this work most of the basic work is already done leaving just the actual controller design. Additionally, circular

air ducts could be modeled so that a more realistic simulation could occur. This would bring the simulation closer to what the physical robot has been tested on.

Also, the simulated robot can take advantage of some recently introduced features of the NTRT library. First, the cable model could be switched for one which includes an actual PID control loop inside it. This will provide more realistic cable movement based on the actual motor parameters for the physical robot. In addition, the cable models could be made to interact with the environment by using a cable model that collides with rigid bodies. While this particular shortcoming does not affect the results of this work as the cables never collide with anything, a future controller could possibly use the collision cables to help navigate a T-junction.

Bibliography

[1] Bullet physics library. [Online]. Available: bulletphysics.org

[2] Collision callbacks and triggers. [Online]. Available: http://www.bulletphysics.org/mediawiki-1.5.8/index.php/Collision_Callbacks_and_Triggers

[3] Nasa tensegrity robotics toolkit (ntrt). [Online]. Available: http://ti.arc.nasa.gov/tech/asr/intelligent-robotics/tensegrity/ntrt/

[4] Ntrt simulator: Learning. [Online]. Available: http://ntrt.perryb.ca/doxygen/learning.html

[5] Push me pull me. [Online]. Available: expeditionworkshed.org/workshed/push-me-pull-me/

[6] Springie: A tensegrity simulator using java, vrml, and pov ray. [Online]. Available: springie.com

[7] A. Ananthanarayanan, M. Azadi, and S. Kim, "Towards a bio-inspired leg design for high-speed running," in *Bioinspiration & Biomimetics*, vol. 7, no. 4. IOP Publishing, 2012, p. 046005.

[8] P. Beyhaghi, D. Cavaglieri, and T. Bewley, "Delaunay-based derivative-free optimization via global surrogates, part i: Linear constraints."

[9] T. Bliss, J. Werly, T. Iwasaki, and H. Bart-Smith, "Experimental validation of robust resonance entrainment for cpg-controlled tensegrity structures," *Control Systems Technology, IEEE Transactions on*, vol. 21, no. 3, pp. 666–678, May 2013.

[10] T. K. Bliss, "Central Pattern Generator Control Of A Tensegrity Based Swimmer," Ph.D. dissertation, University of Virgina, 2011.

[11] J. Bruce, K. Caluwaerts, A. Iscen, A. P. Sabelhaus, and V. Sunspiral, "Design and Evolution of a Modular Tensegrity Robot Platform," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 3483–3489.

[12] K. Caluwaerts, J. Despraz, A. Içen, A. P. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral, "Design and control of compliant tensegrity robots through simulation and hardware validation." *Journal of the Royal Society Interface*, vol. 11, no. 98, 2014. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24990292

[13] P. Corke, "A robotics toolbox for matlab," *Robotics Automation Magazine, IEEE*, vol. 3, no. 1, pp. 24–32, Mar 1996.

[14] A. De Luca and B. Siciliano, "Closed-form dynamic model of planar multilink lightweight robots," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21, no. 4, pp. 826–839, Jul 1991.

[15] T. Erez, Y. Tassa, and E. Todorov, "Simulation Tools for Model-Based Robotics : Comparison of Bullet , Havok , MuJoCo , ODE and PhysX," 2015.

[16] V. Ferkiss, R. B. Fuller, and E. J. Applewhite, "Synergetics: Explorations in the Geometry of Thinking," *Technology and Culture*, vol. 17, no. 1, p. 104, 1976. [Online]. Available: http://www.jstor.org/stable/3103256?origin=crossref

[17] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[18] J. Friesen, P. Glick, M. Fanton, P. Manovi, A. Xydes, and T. Bewley, "The Second Generation Prototype of A Duct Climbing Tensegrity Robot, DuCTTv2," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, submitted.

[19] J. Friesen, A. Pogue, T. Bewley, M. D. Oliveira, R. Skelton, and V. Sunspiral, "DuCTT : a Tensegrity Robot for Exploring Duct Systems," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 4222–4228.

[20] A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From swimming to walking with a salamander robot driven by a spinal cord model," *Science*, vol. 315, no. 5817, pp. 1416–1420, 2007. [Online]. Available: http://www.sciencemag.org/content/315/5817/1416.abstract

[21] A. Iscen, A. Agogino, V. Sunspiral, and K. Tumer, "Controlling Tensegrity Robots through Evolution," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 1293–1300.

[22] A. Iscen, A. Agogino, V. SunSpiral, and K. Tumer, "Flop and roll: Learning robust goal-directed locomotion for a tensegrity robot," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 2236–2243.

[23] W. Jeon, J. Park, and I. Kim, "Development of high mobility in-pipe inspection robot," in *System Integration (SII), 2011 IEEE/SICE International Symposium on*, 2011, pp. 479–484. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6147496

[24] Y. Kawaguchi, I. Yoshida, H. Kurumatani, T. Kikuta, and Y. Yamada, "Internal pipe inspection robot," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 1, 1995, pp. 857–862.

[25] B. T. Mirletz, I.-w. Park, T. E. Flemons, A. K. Agogino, R. D. Quinn, and V. Sunspiral, "Design and Control of Modular Spine-Like Tensegrity Structures," in *6WCSCM: Sixth World Conference on Structural Control and Monitoring*, no. July, 2014.

[26] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[27] W. Neubauer, "Locomotion with articulated legs in pipes or ducts," *Robotics and Autonomous Systems*, vol. 11, no. 3-4, pp. 163–169, 1993.

[28] S. Nicosia, P. Valigi, and L. Zaccarian, "Dynamic modelling of a two link flexible robot and experimental validation," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 3, Apr 1996, pp. 1953–1958 vol.3.

[29] C. Paul, F. J. Valero-Cuevas, and H. Lipson, "Design and control of tensegrity robots for locomotion," *IEEE Transactions on Robotics*, vol. 22, no. 5, pp. 944–957, 2006.

[30] R. Richardson, S. Whitehead, T. Ng, Z. Hawass, A. Pickering, S. Rhodes, R. Grieve, A. Hildred, A. Nagendran, J. Liu, W. Mayfield, M. Tayoubi, and R. Breitner, "The Djedi Robot Exploration of the Southern Shaft of the Queens Chamber in the Great Pyramid of Giza, Egypt," *Journal of Field Robotics*, vol. 30, no. 3, pp. 323–348, 2013.

[31] R. Skelton, R. Adhikari, J.-P. Pinaud, W. C. W. Chan, and J. Helton, "An introduction to the mechanics of tensegrity structures," in *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, vol. 5, 2001, pp. 4254–4259.

[32] K. Snelson, "Continuous tension, discontinuous compression structures," Feb. 16 1965, uS Patent 3,169,611. [Online]. Available: https://www.google.com/patents/US3169611

[33] B. R. Tietz, R. W. Carnahan, R. J. Bachmann, R. D. Quinn, and V. Sunspiral, "Tetraspine: Robust terrain handling on a tensegrity robot using central pattern generators," in *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, 2013, pp. 261–267.

[34] T. White, N. Hewer, B. L. LUK, and J. Hazel, "The design and operational performance of a climbing robot used for weld inspection in hazardous environments," in *Control Applications, 1998. Proceedings of the 1998 IEEE International Conference on*, September 1998, pp. 451–455.

[35] X. Zhang, J. Mills, and W. Cleghorn, "Dynamic modeling and experimental validation of a 3-prr parallel manipulator with flexible intermediate links," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 4, pp. 323–340, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10846-007-9167-4

Appendix A

Appendices to Section 3.1

A.1 Mechanical Test and Validation Results

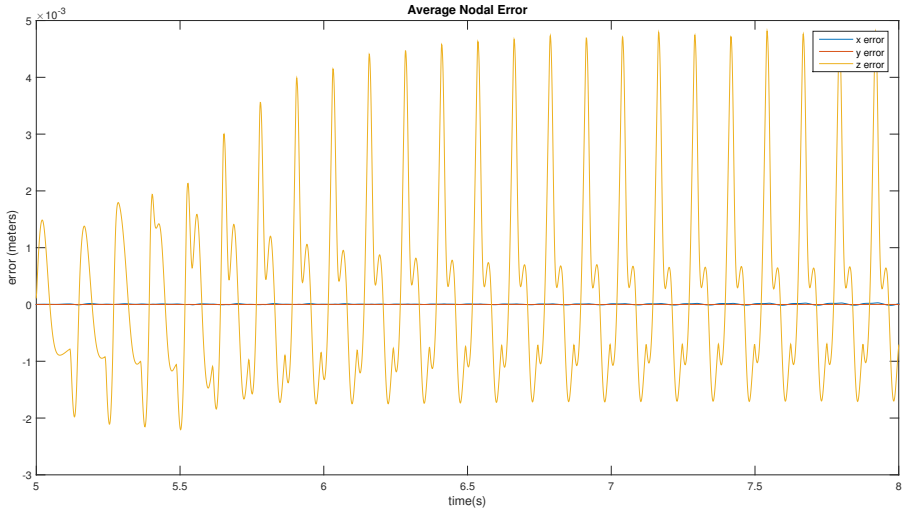


Figure A.1. Test case 1: average nodal error

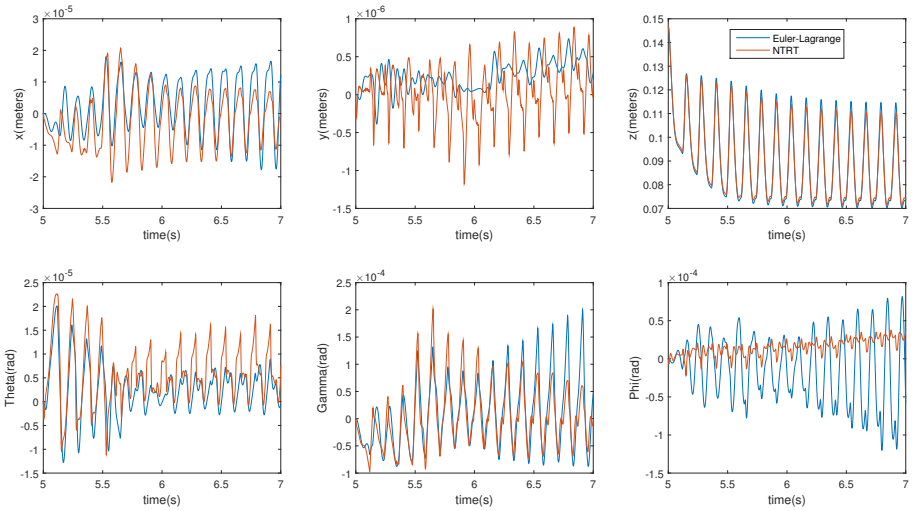


Figure A.2. Test case 1: state variable comparison

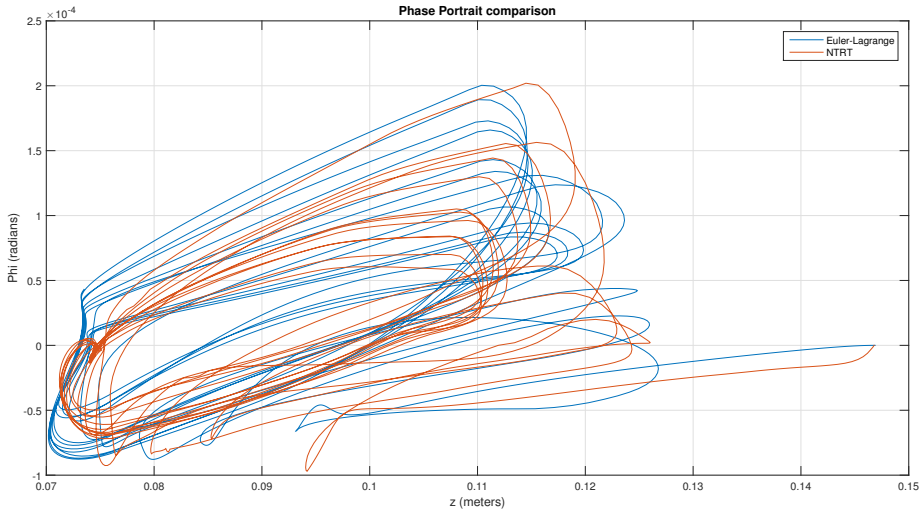


Figure A.3. Test case 1: Comparison of ϕ state variable

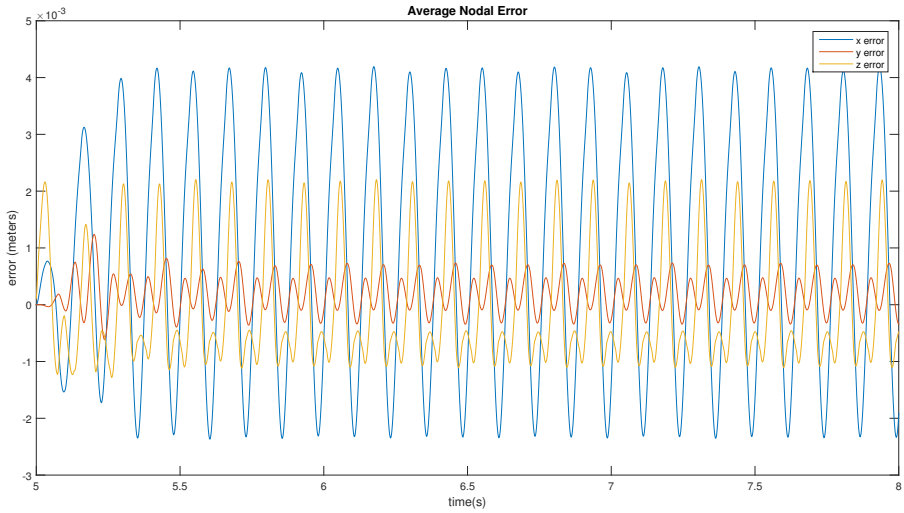


Figure A.4. Test case 2: average nodal error

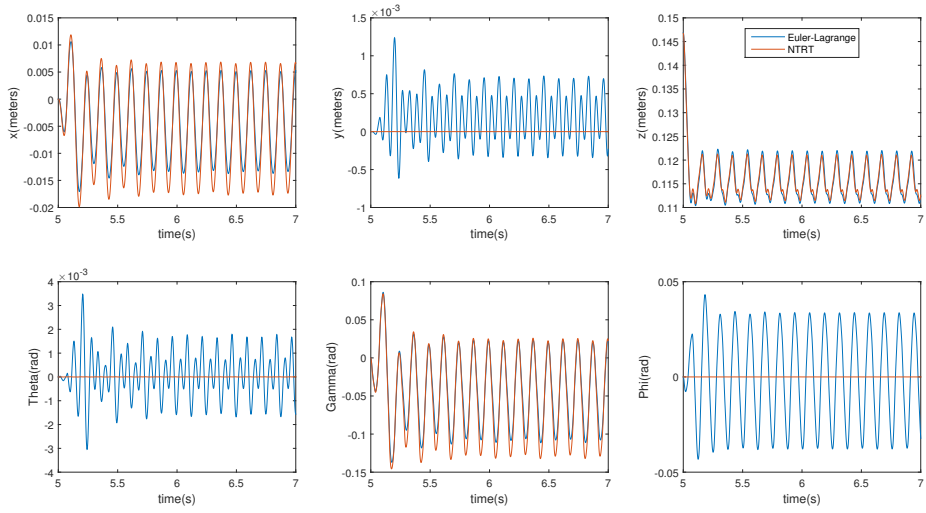


Figure A.5. Test case 2: state variable comparison

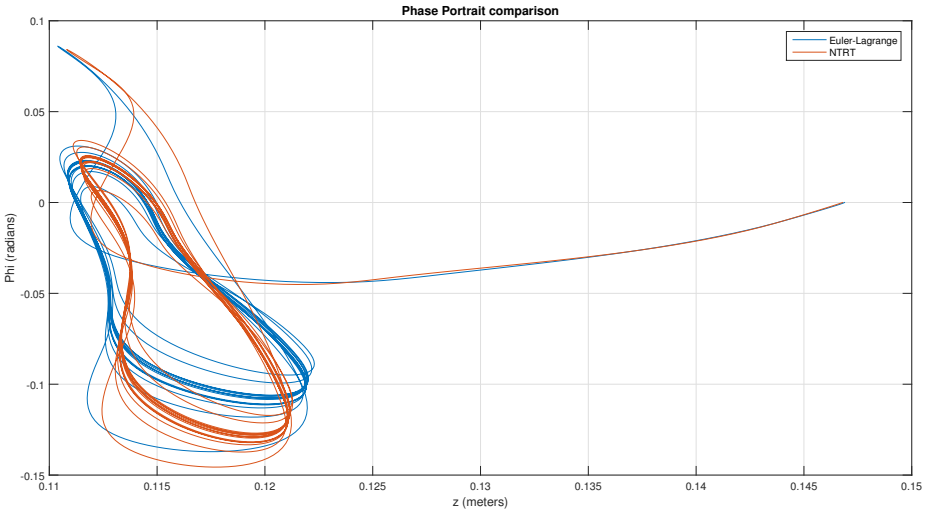


Figure A.6. Test case 2: Comparison of ϕ state variable

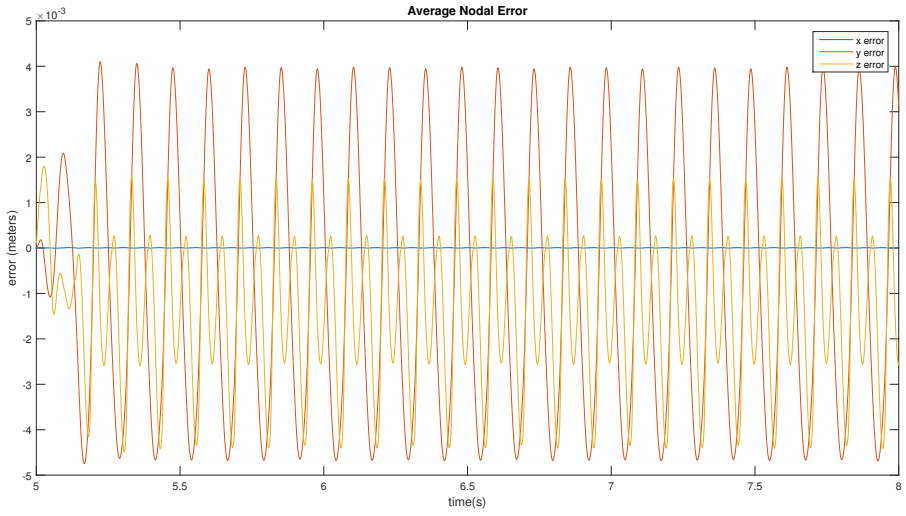


Figure A.7. Test case 3: average nodal error

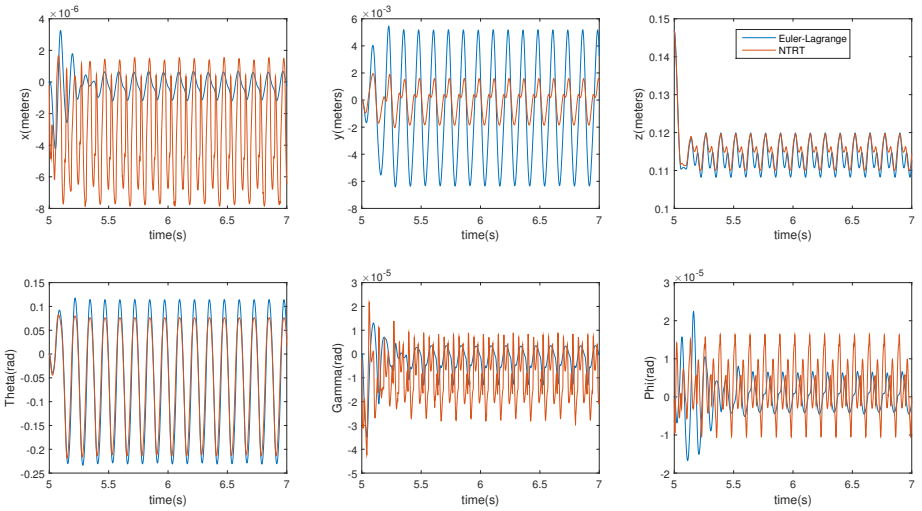


Figure A.8. Test case 3: state variable comparison

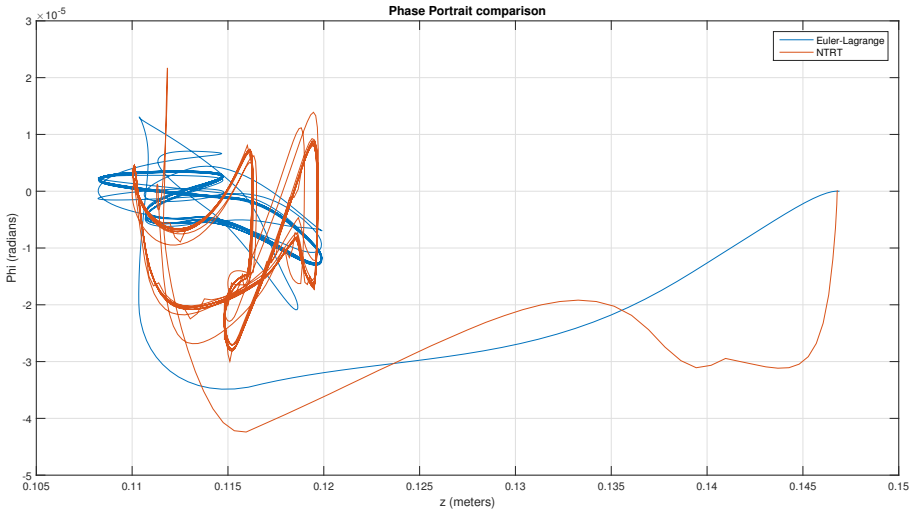


Figure A.9. Test case 3: Comparison of ϕ state variable

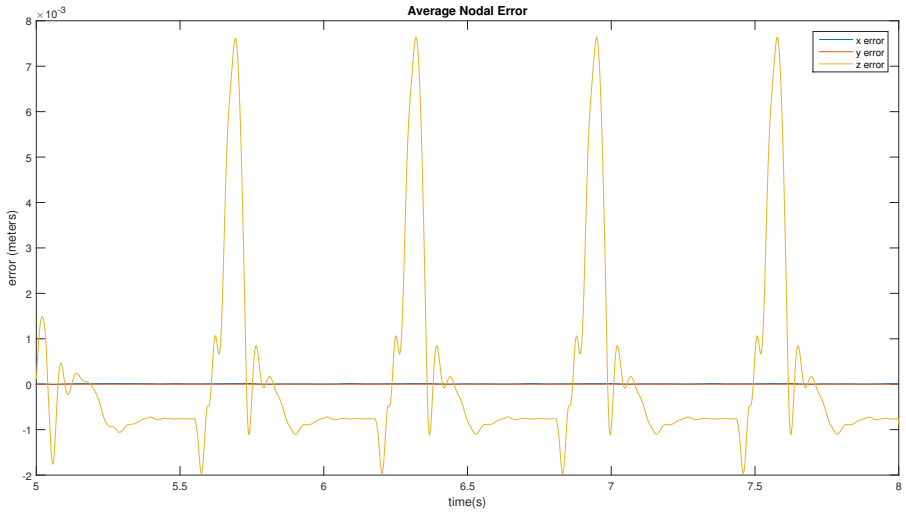


Figure A.10. Test case 4: average nodal error

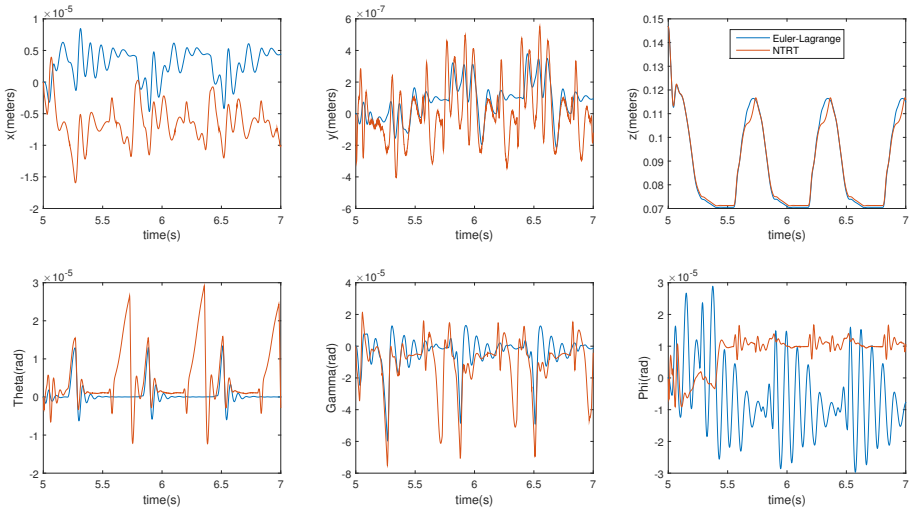


Figure A.11. Test case 4: state variable comparison

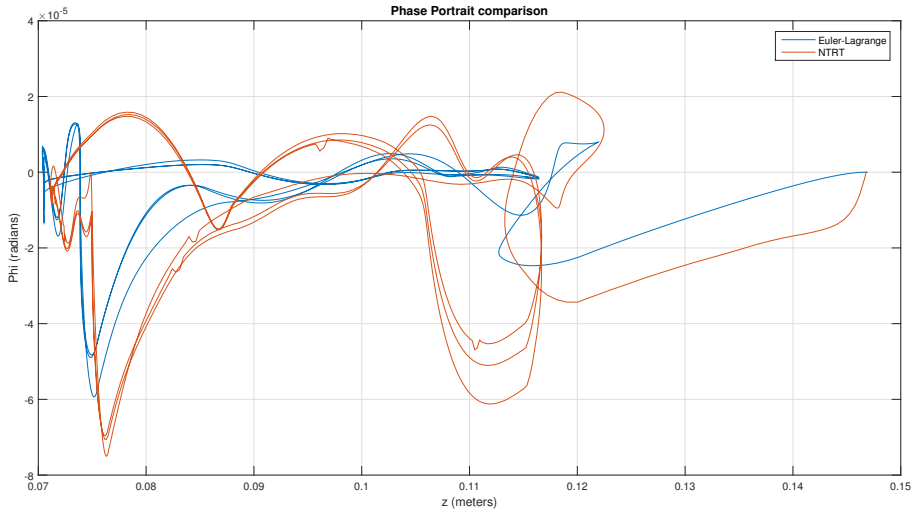


Figure A.12. Test case 4: Comparison of ϕ state variable

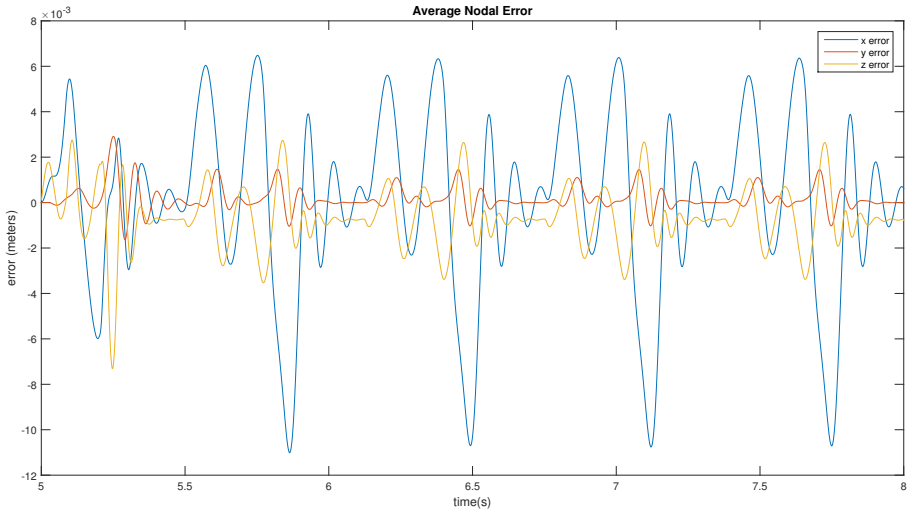


Figure A.13. Test case 5: average nodal error

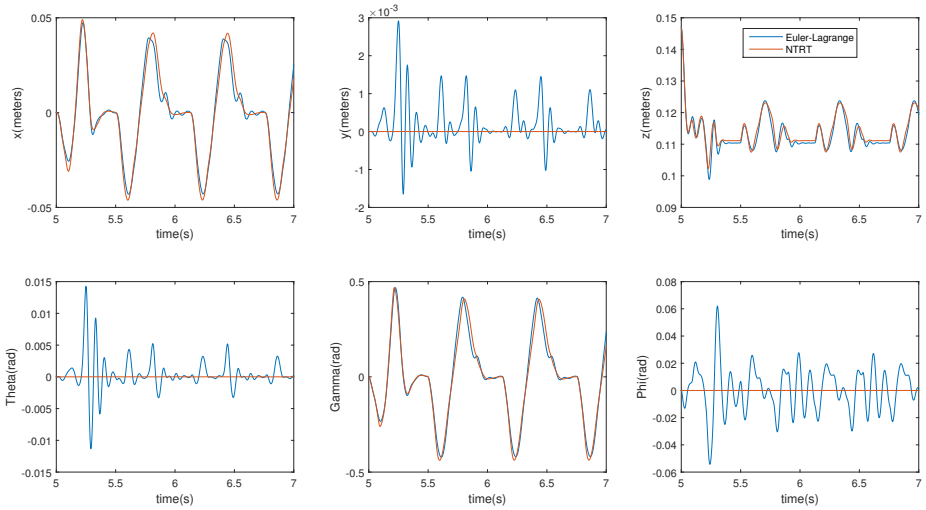


Figure A.14. Test case 5: state variable comparison

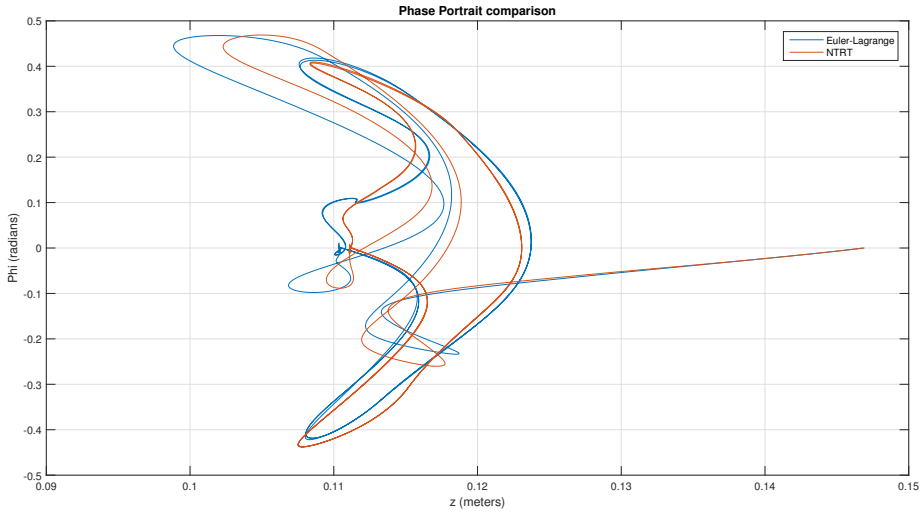


Figure A.15. Test case 5: Comparison of ϕ state variable

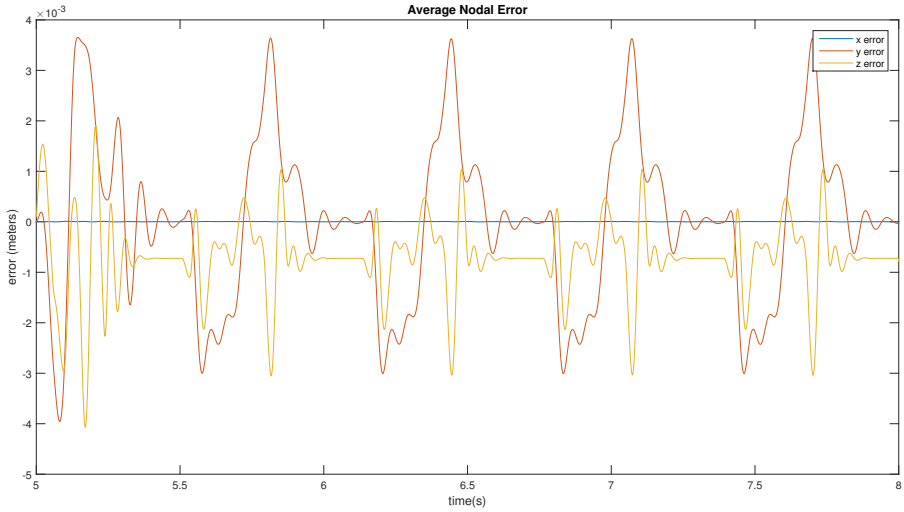


Figure A.16. Test case 6: average nodal error

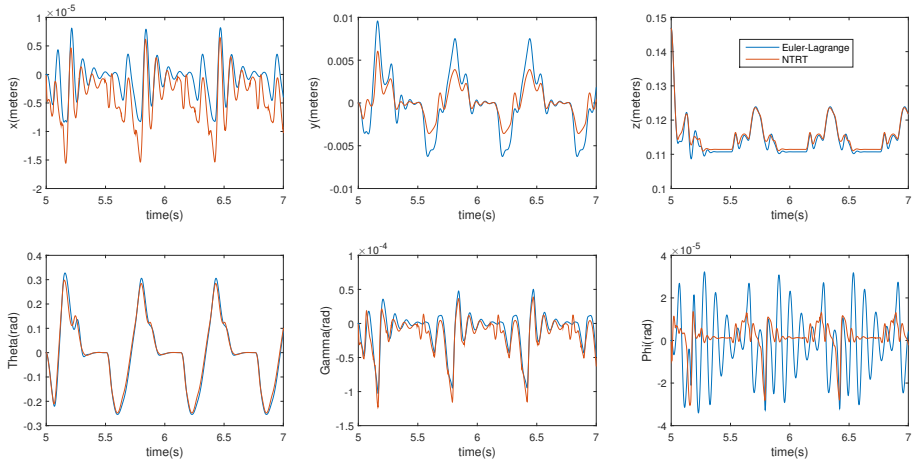


Figure A.17. Test case 6: state variable comparison

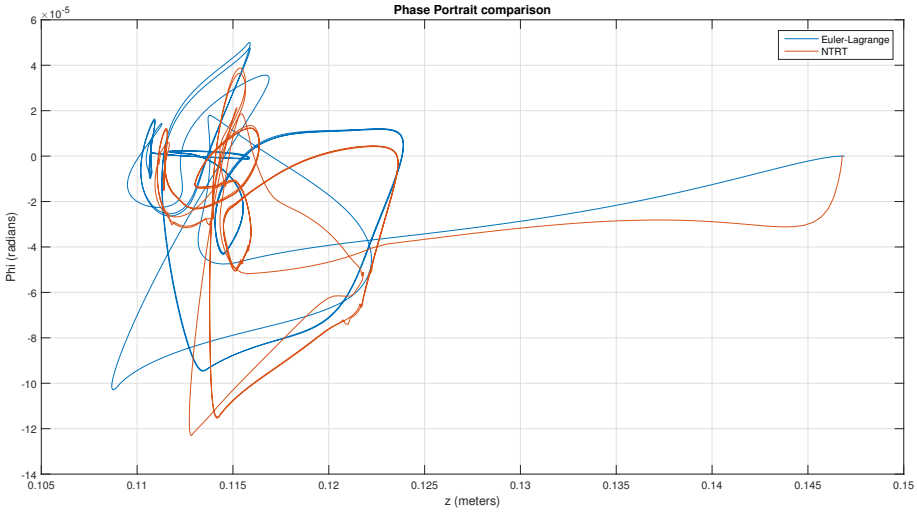


Figure A.18. Test case 6: Comparison of ϕ state variable