5-2019

Motion Planning for a Continuum Robotic Mobile Lamp: Navigating the Configuration Space to Assist with Aging in Place

Zachary Hawks
Clemson University, zachary.j.hawks@gmail.com

MOTION PLANNING FOR A CONTINUUM ROBOTIC MOBILE LAMP: NAVIGATING THE CONFIGURATION SPACE TO ASSIST WITH AGING IN PLACE

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Zachary Hawks
May 2019

Accepted by:
Dr. Ian Walker, Committee Chair
Dr. Ioannis Karamouzas
Dr. Adam Hoover

Abstract

For a robot to operate autonomously, it must have a method of planning its motion through its environment without the explicit guiding control of a human operator. In this thesis, a new approach was implemented to plan a collision-path for a mobile robot featuring a novel continuum arm.

We consider motion planning in the configuration spaces of robots containing continuum elements. The configuration space structure of extensible continuum sections was first analyzed, with practical constraints unique to continuum elements identified. The results were applied to generate the configuration space of a hybrid continuum lamp/mobile base robot developed as a part of a wider project aimed at robots in the home to assist aging-in-place. A conventional motion planning (Rapidly-exploring Random Tree search, RRT/A*) approach was subsequently applied for the robot in the aging-in-place application scenario.

The RRT generated complete paths through various environments and was successfully able to connect the start configuration to the goal configuration using the robot's specific configuration space. Once the RRT completed, an A* search algorithm was run on the graph and the optimal path was found. This path, consisting of series of actions necessary for the robot to move from configuration to configuration, was then communicated to two generations of robot hardware using a local wireless network. The robots then executed the actions and moved through the environment.

Dedication

I'd like to dedicate this thesis to Karen: You always believed in me even when I stopped believing in myself. Your support and encouragement got me through the hardest moments, and I will always love you for that.

Acknowledgments

I'd like to thank my adviser Dr. Ian Walker for all his support and guidance in this work. I'd also like to acknowledge my committee members Dr. Ioannis Karamouzas and Dr. Adam Hoover for their contributions.

In addition, I'd like to thank the other members of our lab for all their help and camaraderie. Specifically, I want to thank Chase Frazelle, whose help got me through the toughest problems and whose friendship made grad school more than just a job. I couldn't have done it without you, Chase.

Table of Contents

List of Figures

Chapter 1

Introduction

This thesis addresses the problem of motion planning for mobile robots featuring novel continuum sections. We discuss the nature of the configuration space of, and its use in motion planning for, continuum robots.

Continuum robots are composed of one or more continuum sections. A continuum section is kinematically described by continuous and smooth curvature [3, 4]. Continuum robots theoretically possess infinite degrees of freedom (DoF), unlike standard rigid-link robots which have finite DoF. Continuum sections are most often tendon or pneumatically driven, or composed of concentric tubes. These robots are often inspired by elements in biology, such as plant tendrils, an elephant trunk, or octopus tentacles. Because of their underlying curvature, continuum robots are often compliant in nature and are used to explore hard-to-reach areas [5, 6].

Generating control algorithms for robots, including continuum robots, is a well-studied problem, and multiple solutions are widely accepted and used [4, 7]. Often, these robots are tele-operated and a lot of effort has been focused on intuitive control methods for different robots [8, 9]. In contrast, autonomous motion requires motion planning.

Motion planning is the process of determining a path between two configurations

of the robot using its kinematics. A robot's configuration can be described as vector of the current value(s) of the independent kinematic variables of the robot. The set of all possible configurations is the configuration space of the robot.

For conventional, non-continuum robots, classical motion planning techniques using configuration space have been well studied [10]. For example, rapidly exploring random tree (RRT and RRT*) algorithms have been shown to successfully span the configuration space for mobile robots and rigid-link robots [11, 12]. The A* algorithm, given a graph and proper heuristic function, will guarantee the optimal path between any two nodes if one exists [13].

In the past, a variety of motion planning techniques have been proposed for continuum robots. The motion planning problem for active cannulas (concentric tube robots in medical applications) within tubular environments is formulated as an constrained optimization problem in [14].

Constrained optimization is also used in [15] to formulate and solve the motion planning problem for a soft planar continuum manipulator. Grasp planning for continuum robots using a bounding circle technique was investigated in [16] and [17]. A follow the leader approach for tendon-driven continuum robots is introduced in [18]. Researchers have used sampling based approaches based on the techniques of Rapidly-Exploring Roadmaps (RRM) [19], Rapidly-Exploring Random Graphs (RRG) [20, 21], and Rapidly-Exploring Random Trees (RRT and RRT*) [22, 23] to plan motions for concentric tube continuum robots in tubular environments for medical applications. RRTs are also used by [24] for steering bevel-tip needles in 3D (medical) environments.

However, it appears that the principles of the classical motion planning techniques such as RRT and A* have not yet been applied to tendon-actuated continuum robots in general non-tubular environments. This is in part due to a lack of formal analysis of the nature of the configuration space of tendon-actuated continuum robot elements. In this thesis, we

define and discuss the configuration space of single section extensible continuum robots and use the configuration space to plan paths for hybrid mobile/continuum robots using RRT. We select RRT for its ability to operate well in dynamic environments as an anytime-approach [11]. As we will detail further, both the environment and current objective might change rapidly, and the RRT approach will help to account for this.

The thesis is organized as follows: first in Chapter 2 we discuss the history of *home*+, our collection of robotic home furnishing elements designed to assist in the home with aging-in-place, and introduce the continuum robotic mobile lamp in Fig. 1.1(a-b) [1]. In that Chapter we also discuss the key motivation —to envision autonomous control for the h+lamp —behind our investigation of classical motion planning techniques applied to the configuration space of a hybrid mobile/continuum robot. In Chapter 3, we then discuss the hardware upgrades that we have made to the h+lamp that have led to the development of CuRLE, the third generation hybrid mobile/continuum lamp, shown in Fig. 1.1(c). The research in this thesis reported in Chapter 4 analyzes the configuration space for tendon-driven continuum sections for the first time [25]. The analysis is then applied to the specific example of CuRLE and further detailed in Chapter 4. In the process, we illustrate and highlight several previously unconsidered structural constraints imposed by the tendon-actuated continuum geometry.

The insight gained, and the configuration space models constructed, in Chapter 4 are exploited in Chapter 5 to develop motion planning algorithms for CuRLE. This work represents the first motion planning for hybrid mobile robot/tendon-driven continuum robots. The efficacy and potential of the results are demonstrated via a series of demonstrations using CuRLE reported in Chapter 6. Conclusions and suggestions for future work are presented in Chapter 7.

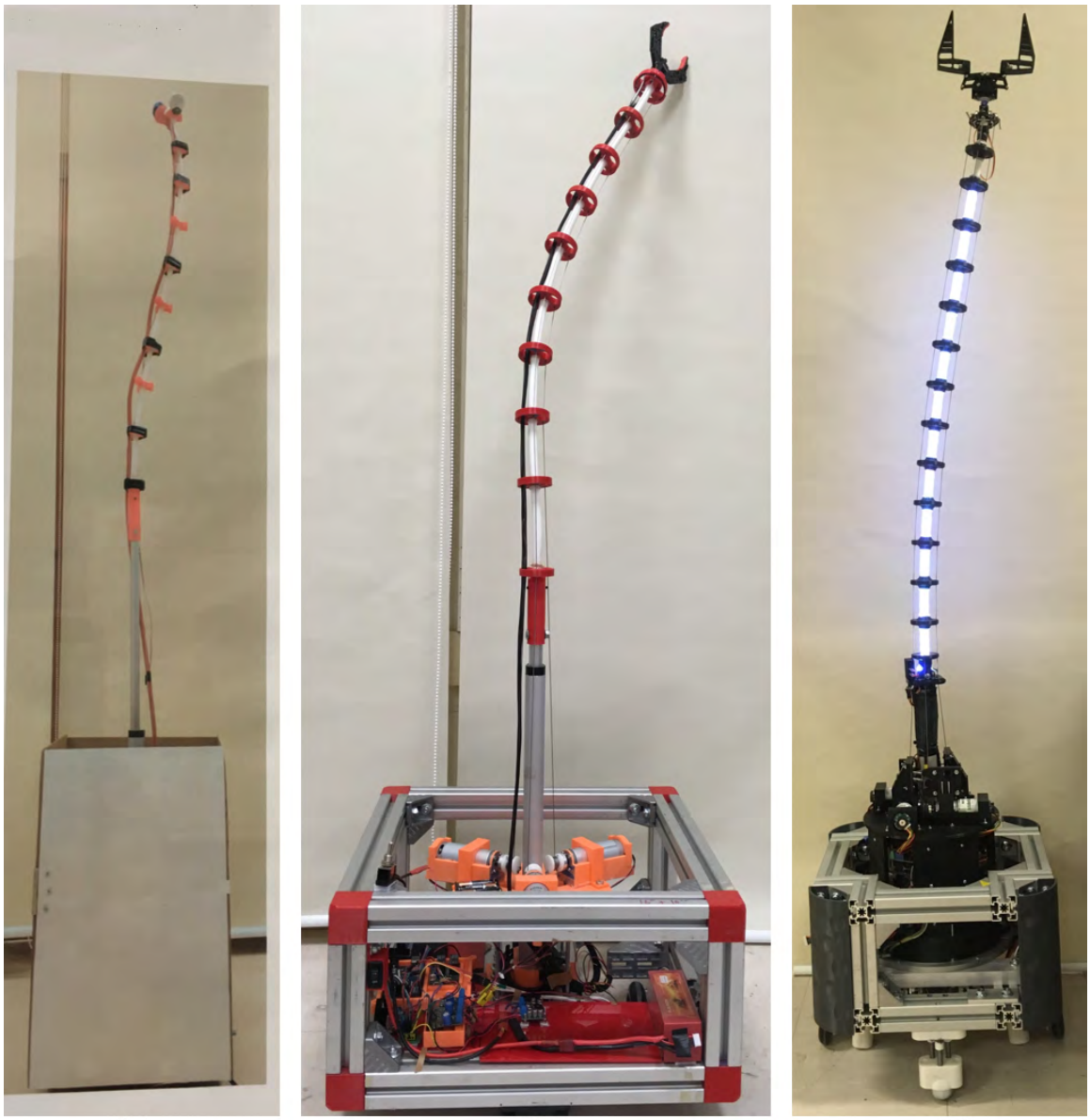


Figure 1.1: The evolution of the continuum robotic mobile lamp element of the *home+* suite. (a) The original first generation lamp reported in [1]. (b) The second generation, called h+lamp, also described in [1]. (c) The third generation, called CuRLE (**C**ontin**u**um **R**obotic **L**amp **E**lement) that forms the basis for this thesis.

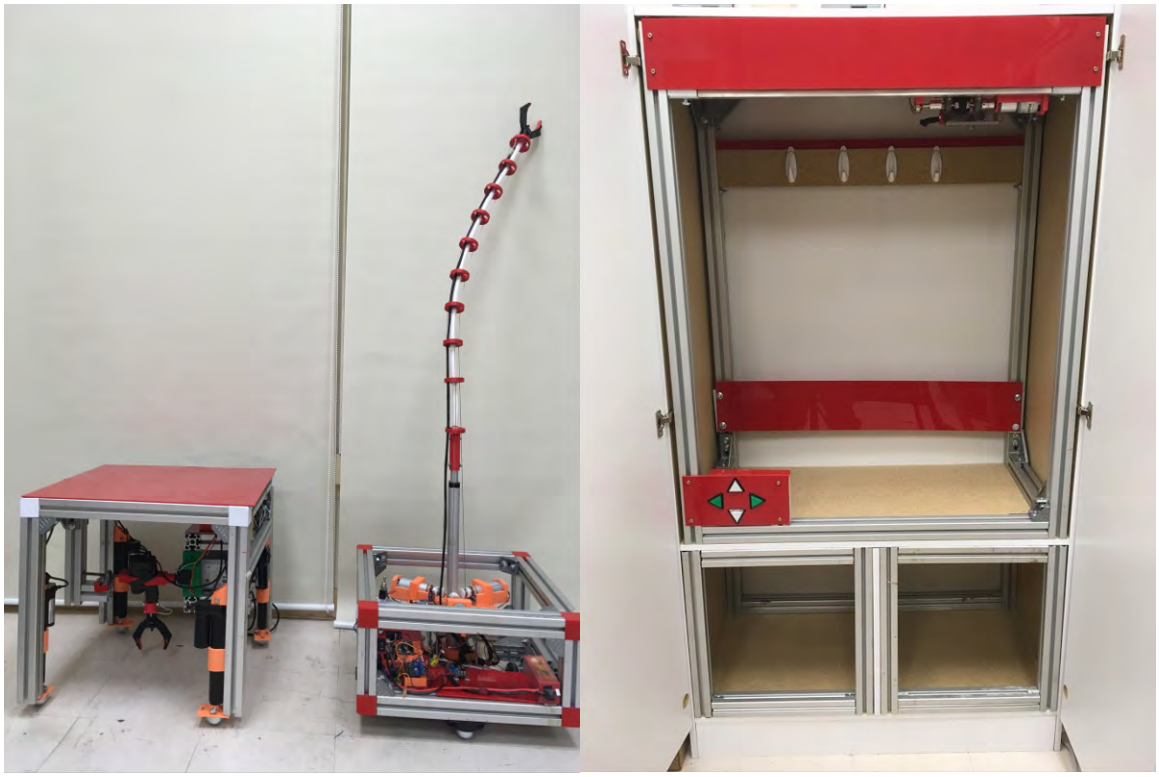
Chapter 2

Home+ and the Second Generation Continuum Robotic Mobile Lamp

This Chapter describes the first attempt at realizing the *home*+ continuum robotic mobile lamp as an autonomous system. In doing so, we first discuss the key motivation driving the *home*+ research project in more detail, and then describe how the work in this thesis supports that effort.

2.1 In-Home Scenario

With the current societal move towards smart devices in every home, we envision a collection of robotic furnishing elements that can provide at-home care and assistance. As we age, we often lose the ability to perform simple day-to-day tasks and eventually reach a point where we can no longer live without assistive care. Our suite of robots, collectively called *home*+, are intended to collaborate with individuals over time in the home to help with these day-to-day tasks and prolong the time that the individual can live independently [1].



(a) (b) (c)

Figure 2.1: The *home+* suite of robotic furnishing elements. (a) h+cube (b) h+lamp (c) h+armoire

We often envision at-home robotic care to be administered by fully functional, android-esque robots such as those seen in cinema and dreamed of in science fiction. That technology, however, has yet to be created, while many of the tasks that individuals need assistance with can be accomplished by robotic technology that currently exists. For instance, people need help retrieving objects from high shelves, so a continuum robotic mobile lamp that includes the ability to do such tasks (in addition to functioning as a lamp) was developed prior to the work reported in this thesis. This first generation robot, shown in Fig. 2.2, was added to the *home+* suite. The other elements of *home+* are a robotic end-table, or "cube", which is detailed in [1], and a robot "armoire" (Fig. 2.1). These robots coordinate together with the user to accomplish every-day tasks and assist the user with aging-in-place.

One of the key motivations driving the *home+* effort has been to evaluate various levels of user interaction with the suite of robots. For the work reported in this thesis, we selected the continuum robotic mobile lamp experiment with the spectrum of user control. To this end, we made extensive upgrades to allow for tele-operation and autonomous motion, as seen in Fig. 2.3 and 2.4. With this second generation robot, referred to as h+lamp hereafter, we were able to conduct preliminary experiments to evaluate motion planning for the mobile base.

2.1.1 Tele-Operation of h+lamp

On one end of the spectrum of control, the user is fully in command of the robot via tele-operation, in which the user provides input to the robot via some form of user-interface. The first generation lamp was controlled by a series of switches directly wired between the power supply and the actuators [1]. Our initial upgrades moved this tethered, analog control method to be wireless and digital. A hardware controller, shown in Fig. 2.5, received input from the user and communicated the commands with wireless Bluetooth technology to the robot. This was subsequently replaced with an Xbox360®controller communicating over a wireless LAN connection by installing Wi-Fi enabling hardware into the h+lamp, shown in Fig. 2.6. A central computer decoded input on the controller via C++ code which then sent the commands to the robot over Wi-Fi.

2.1.2 Autonomous Motion of the h+lamp

The other end of the control spectrum is full autonomy of the robot. In this mode, the user would give verbal, high-level commands, such as ”Bring me my cup.” and the h+lamp would autonomously navigate (avoiding obstacles) to where the cup is stored, grab the cup, and bring it back to the user. This is the mode the work in this thesis was intended to

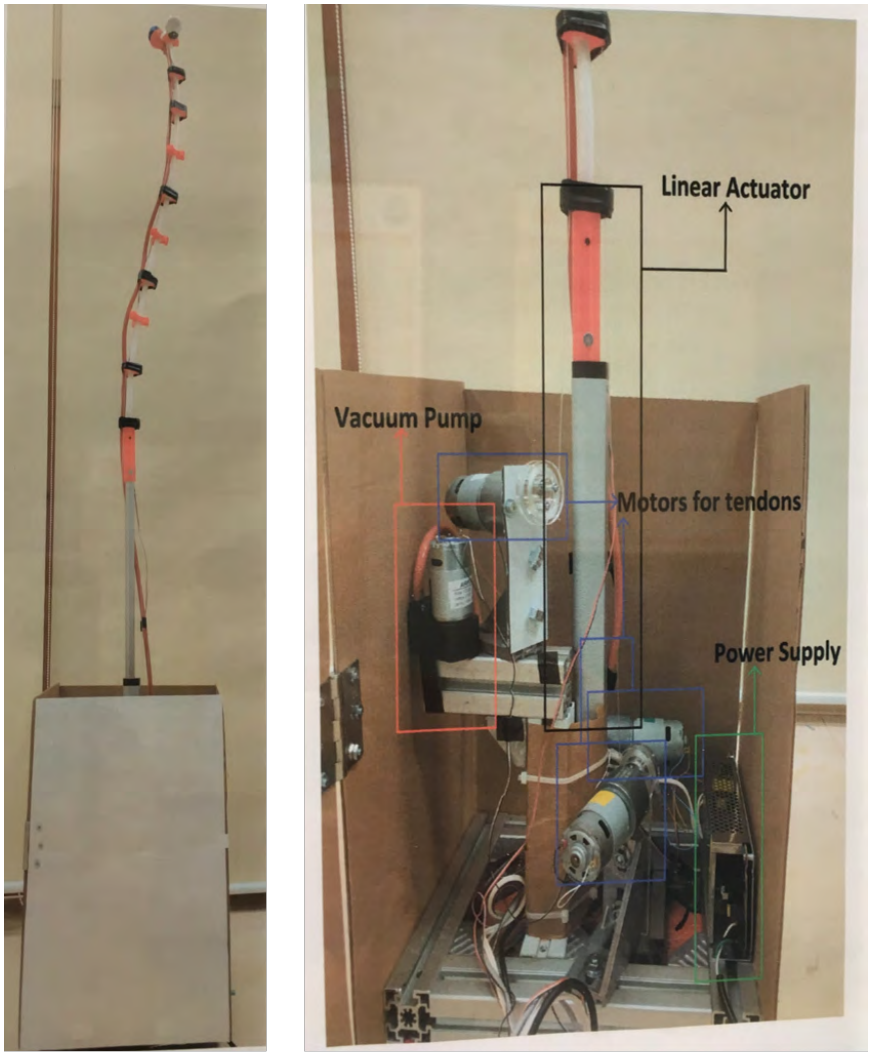


Figure 2.2: First generation lamp hardware, as detailed in [1].

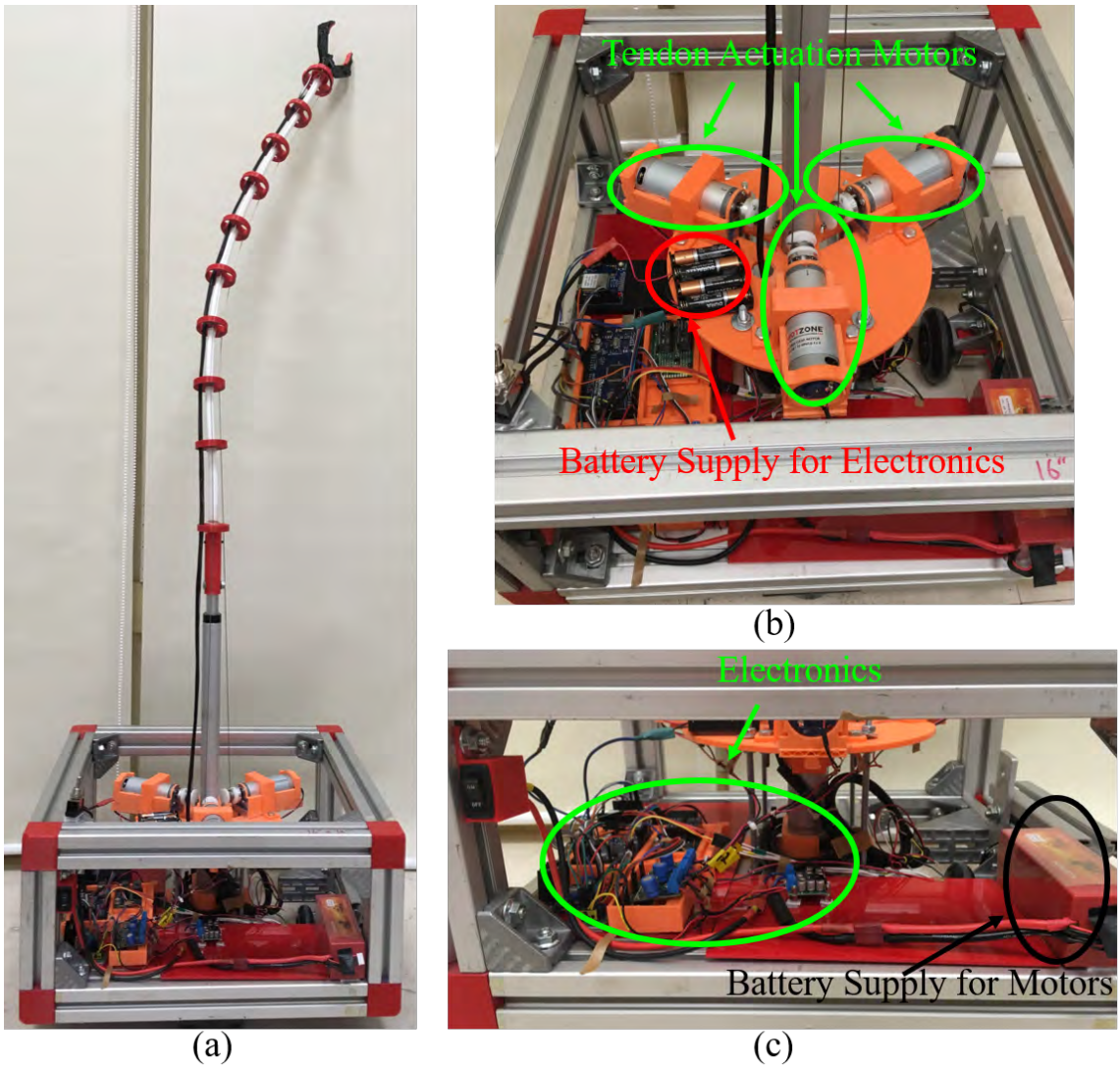


Figure 2.3: (a) The full replicated and upgraded hardware of the h+lamp. (b) A top-down view of the base of h+lamp showing the tendon motors and electronics. (c) A side-view of the of the base of h+lamp showing all of the electronics (i.e. Arduino Mega, motor drivers, voltage regulator, power supply, Wi-Fi shield)

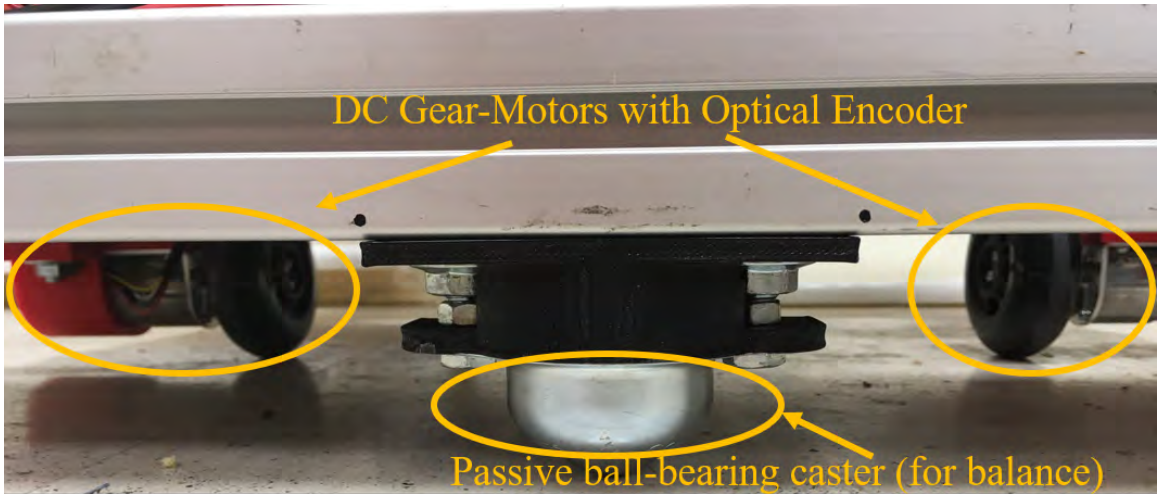


Figure 2.4: A side-view of the h+lamp drive subsystem.

Figure 2.5: Tele-operation method of the h+lamp hardware using custom-built controller and Bluetooth.

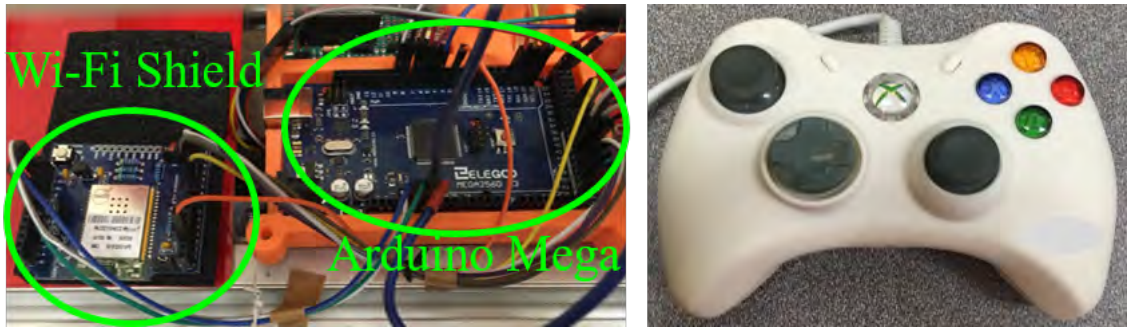


Figure 2.6: Tele-operation method of the h+lamp hardware using Xbox360®controller and wireless LAN.

enable. As a first attempt to do this, the passive mobile base (realized by steel ball-bearing casters) was replaced with a differential drive system. This is detailed in section 2.2. To implement full autonomy, sensing capabilities would have to be realized for the h+lamp to recognize user input via voice command and detect the environment obstacles, in addition to navigating the task space. The realization of this is beyond the scope of the project reported in this thesis. As described later in Chapter 5, in this work the user's commands (i.e. the goal configurations) and the location of all obstacles are assumed known *a priori* and we demonstrate progress towards autonomy with the novel motion planning algorithms we developed.

2.2 Implementation of Path Planning for Mobile Base of h+lamp

The differential drive system developed as part of this thesis work consisted of two encoded DC gear-motors and a dual H-bridge motor driver controlled by PWM signals from the Arduino Mega development board that functioned as the micro-controller for the robot. The drive system, shown in Fig. 2.4, was powered by a 4S LiPo battery and step-down voltage regulator, while the electronics were separately powered by 4 AA batteries.

Fig. 2.3 shows the h+lamp in this prototype version.

As mentioned earlier, the h+lamp connected to a central computer over Wi-Fi. Rather than transmit user inputs from an Xbox360®controller, however, the central computer ran the motion planning algorithms to autonomously move the robot from configuration to configuration. These algorithms, RRT and A*, are detailed in Chapter 5.

The output of the RRT/A* algorithms was the path the robot must follow to reach the goal. The path consisted of the set of actions $U = \{\mu_1, \mu_2, \ldots \mu_n\}$, which was wirelessly transmitted, one action at a time, to the robot via a TCP/IP socket connection. The robot acknowledged the initial transmission ("init") and began executing the actions as they arrived. If a new action arrived before the robot finished its current action, the new action was queued and executed next.

The motors were controlled by a PID controller implemented in the Arduino IDE. As each action arrived, the robot calculated the new set-point of each wheel based on its dynamics. A simple state-machine controled the flow of execution. The robot was idle until it received the "init" command from the central computer. At this point, the robot acknowledged the central computer and entered a waiting state until an action command arrived. Once an action arrived, the robot moved through the action vector, first performing a rotation, then translation, then the final rotation. Each individual movement was executed by calculating the distance each wheel would travel. For a rotation, the distance σ_i traveled by each wheel is given by Eqn. (2.1) where L_{wheels} is the distance between the wheels and φ_i is the rotation of the current action.

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} = \begin{bmatrix} \varphi_i * \frac{L_{\text{wheels}}}{2} \\ \varphi_i * \frac{L\text{wheels}}{2} \end{bmatrix} \tag{2.1}$$

For a translation, the distance δ_i traveled by each wheel is given by Eqn. (2.2) where δ is the translation distance of the current action μ.

$$\begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \delta \\ \delta \end{bmatrix} \tag{2.2}$$

Either δ_i or σ_i was added to the current position of the wheel to give the desired set-point for each wheel. The current position of each wheel p is given by Eqn. (2.3) where E_{count} is the current encoder count, C_{rev} is the counts per revolution of the wheel, and r_{wheel} is the radius of the wheel.

$$p = E_{\text{count}} * C_{rev} * r_{\text{wheel}} * 2\pi \tag{2.3}$$

Once the desired set-point was calculated, the PID controller calculated the error for each wheel and updated the velocity of the motors to move the error to zero. Once the error was consistently below the minimum threshold value, the state-machine moved to the next value of the action vector. If the action vector was complete and there were actions waiting in the queue, the top value was popped out of the queue and execution began again; otherwise the system moved back to the ready state awaiting the next action.

2.3 Results

After successfully running the RRT, watching the graph grow, and watching the simulated robot successfully navigate the path (Chapter 5), we implemented the RRT on the h+lamp. Before running the full path, the PID motor controller was thoroughly tested

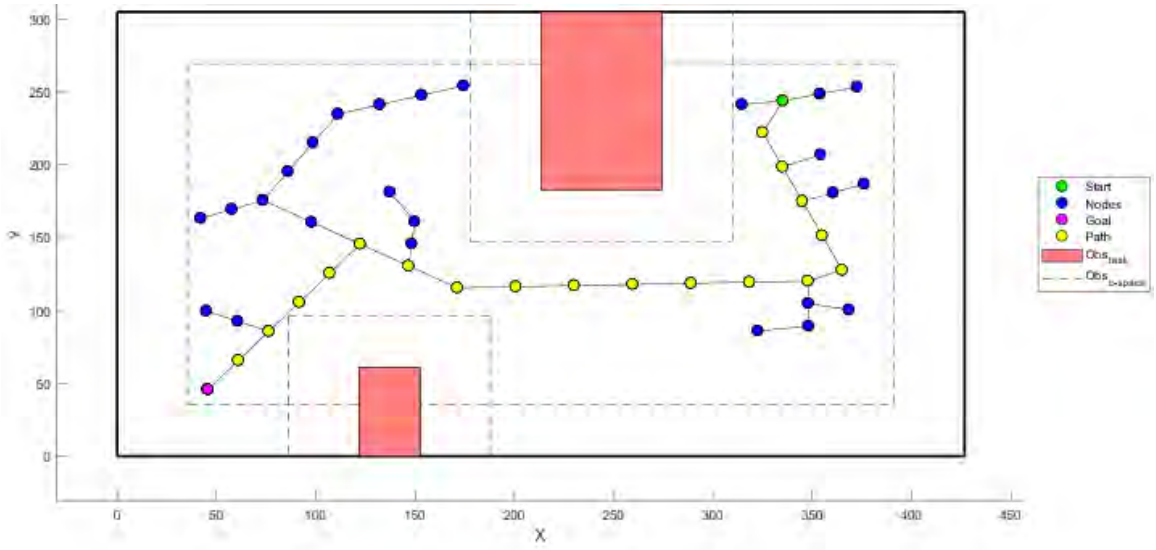


Figure 2.7: The path for the h+lamp to follow through the in-home scenario experiment.

and tuned by serially sending the robot one action at a time. We also verified the state machine logic and saw the robot successfully queue actions as it received them, executing them in order one at a time. After placing the robot in the work space, which is shown in Fig. 2.8, we executed the RRT and sent the path to the robot over Wi-Fi. The path for the robot to follow is shown in Fig. 2.7. The best results of the robot executing this path are shown in Fig. 2.9.

2.4 Analysis

The results revealed the need for improvements in the prototype hardware. The robot began execution, but due to errors in the motor control, was not able to successfully navigate to the goal. Errors in the motor control occured due to slight differences in the velocities of the two different drive motors. For rotations, the velocities of the wheels were not an issue and final position was reached. Since the PID control operated on errors in the position, all rotations were successful. For translations, however, the unsensed velocities

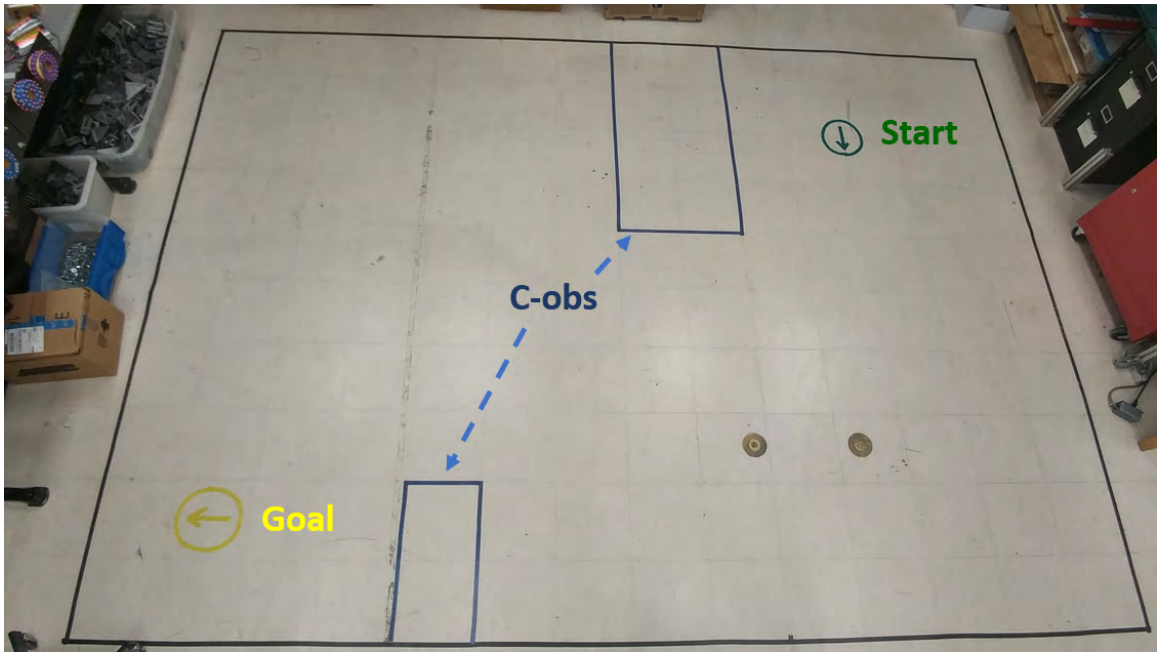


Figure 2.8: A top-down view of the physical task space of the in-home scenario used to validate the motion planning algorithms using the h+lamp robot hardware. The start location is shown in the top right in green with the goal location shown in the bottom left in yellow. The configuration space obstacles are shown in blue.

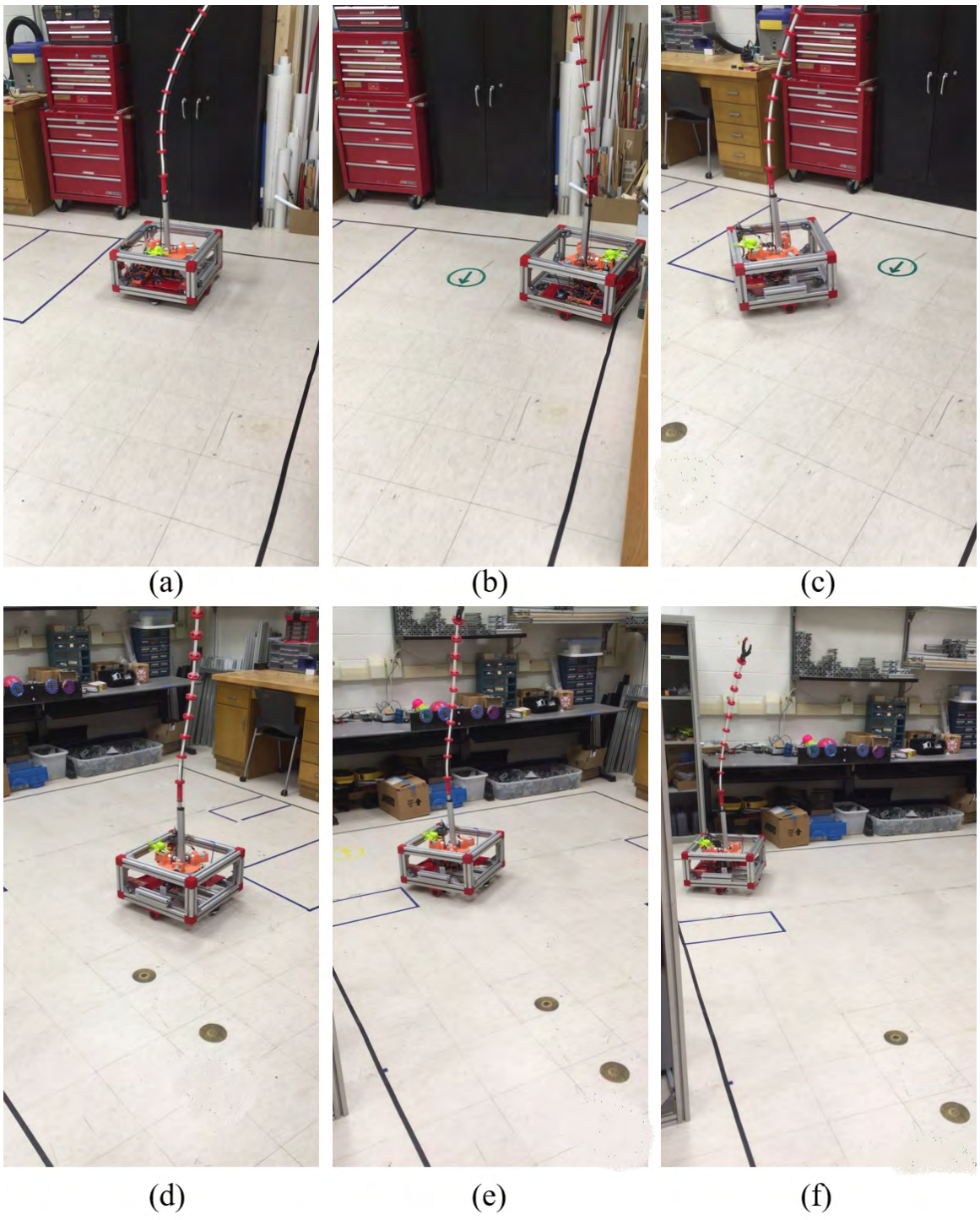


Figure 2.9: A series of top-down views showing the h+lamp moving through the physical task space of the in-home scenario.

of the wheels matter significantly. If one wheel moves faster than the other, there will be slight drift in the path towards the slower wheel. While the PID controller will force the positional error to 0 for both wheels, this occurs at different times. The faster wheel stops while other keeps moving. As a result, the robot ends up in the wrong position, and since all the planning happens offline, the system is unable to sense and correct for the error and does not update its trajectory as a result. As the errors accumulated, eventually the robot collided with obstacles, (Fig. 2.9(c)) or left the work space (Fig. 2.9(b)).

Another significant cause of error occured due to the height mis-match between the active drive elements (the wheels connected to the motors) and the passive drive elements (the ball-bearing casters). In order to improve balance and stability, we added two passive casters to the bottom of the h+lamp frame. We designed the casters to be the same height as the wheel assembly, but due to the uneven surface of the task space floor, situations arose where the passive elements maintained contact with the floor while the wheels were ”lifted” enough to cause slipping. To correct this, we significantly reduced the passive element height, thereby sacrificing four points of contact with the floor for three, but guaranteeing that the wheels would always maintain contact with the floor. A by-product of this modification was that robot would ”rock” back and forth around the wheel axis when it accelerated. This caused further drift when executing the path from the RRT.

Our first attempt at validating the motion planning algorithms for the mobile base was successful in the sense that robot received the commands and was able to serially execute them. Due to hardware limitations and errors in the closed-loop control, the robot did not accurately reach the goal. However, we showed that the motion planning algorithms we developed allow the mobile base of the h+lamp to move through the task space. In the next Chapter, we describe work done to fully upgrade the h+lamp to reduce the hardware issues seen in this Chapter.

Chapter 3

Third Generation Continuum Robotic Mobile Lamp: CuRLE

Given the problems observed during experimentation discussed in Chapter 2, we constructed brand new robot hardware and installed upgraded features to solve the issues we experienced, as well as to expand the capabilities of the robot. We named this third generation robot CuRLE: **C**contin**u**um **R**obotic **L**amp **E**lement, shown in Fig. 3.1. This chapter describes our motivation for, and the details of, the extensive upgrades to produce the new hardware.

3.1 Structural Overview

The structural hardware of CuRLE is comprised of three main components: the base frame, the center structure, and the continuum backbone. The base frame, shown in Fig. 3.2 is built from aluminum beams and 3D-printed plastic corner pieces. The frame is square with rounded corners. Mounted below the frame is a differential drive system, described in section 3.3.2. Mounted inside to the bottom of the frame is a turntable mechanism,

Figure 3.1: Third generation continuum robotic mobile lamp (CuRLE) with internal LED stip lit.

described in section 3.3.1. Mounted on the turntable is the center structure.

The center structure is comprised of a 3D-printed plastic base plate, 3D-printed walls, and a 3D-printed top plate. The linear actuator (the only vestigial hardware from the h+lamp) is mounted to the base plate along with the bottom acrylic electronics plate (Fig. 3.5(b-c)). The walls attach to plastic struts that connect to the top and bottom plates. Small protruding shelves can be attached to the inside of each wall, which support the top acrylic electronics plate (Fig. 3.5(a)). All of the motor assemblies, described in section 3.3.3, are mounted to the top plate of the center structure (i.e. the motor plate).

The continuum element is mounted at the end of the linear actuator and is comprised of a PEX backbone (same material, but wider diameter than used for h+lamp's backbone) and new 3D-printed vertebrae. The actual "lamp" features of CuRLE are comprised of LED's mounted to the final vertebra (i.e. the end-effector) and an LED strip that runs through the backbone. These lighting features are described in section 3.3.5. Also mounted to the end-effector is a gripper, detailed in section 3.3.4. The final subsystem is the power system, described in section 3.3.6, which is mounted to the base of the frame.

3.2 Electronics Upgrades

To enable the hardware to execute as intended for the work in this thesis, the electronics of the h+lamp were necessarily upgraded. The Arduino Mega was replaced with the Arduino Due, which has more computational power and higher clock speeds. The faster clock is needed to enable the encoder counter chips, which replaced the old method of directly reading the encoders via interrupts on the Arduino. The new chips allow the Arduino to focus its execution on other tasks (it no longer has to constantly handle interrupts from the encoders) and reduce the number of digital pins required to interface with the motors.

To replace the Wi-Fi shield of the h+lamp, and eventually move CuRLE to a fully

Figure 3.2: The full base of CuRLE.

stand-alone unit, a Raspberry Pi was installed to serve as the ”central computer” for the robot. Via remote wireless access, this computer serves as the connection point for the robot and relays external commands to the Arduino via a hardware serial connection. The Raspberry Pi can also relay traffic from the Arduino, enabling two-way communication which was non-existent on the h+lamp. Figs. 3.3 and 3.4 show the electronics as they are mounted within CuRLE.

All of the electronics, excluding those mounted to the continuum element and those mounted beneath the center structure, are attached to the two acrylic plates shown in Fig. 3.5(a-c). The base plate holds the power electronics and the two computing devices (Raspberry Pi computer and Arduino Due micro-controller). The top plate holds the encoder counter chips, relay switches, and dual H-bridge motor drivers (same type as those in h+lamp). The top plate also contains all the connection points for every motor (tendon, turntable, and drive).

3.3 Details of the Hardware Upgrades

In the following sections, we describe the different hardware components of CuRLE.

3.3.1 Turntable

Since CuRLE needed to possess the capacity to autonomously execute the output from motion planning algorithms developed for the continuum element (detailed later in Chapter 4), we sought to fully separate the continuum c-space from the c-space of the mobile base by adding a revolute joint at the base of the lamp. This joint allows the continuum element to rotate independent of the movement of the mobile base. This added redundancy creates new configuration opportunities for CuRLE.

The revolute joint, (variable ω), is realized by a worm drive run by DC gear-motor

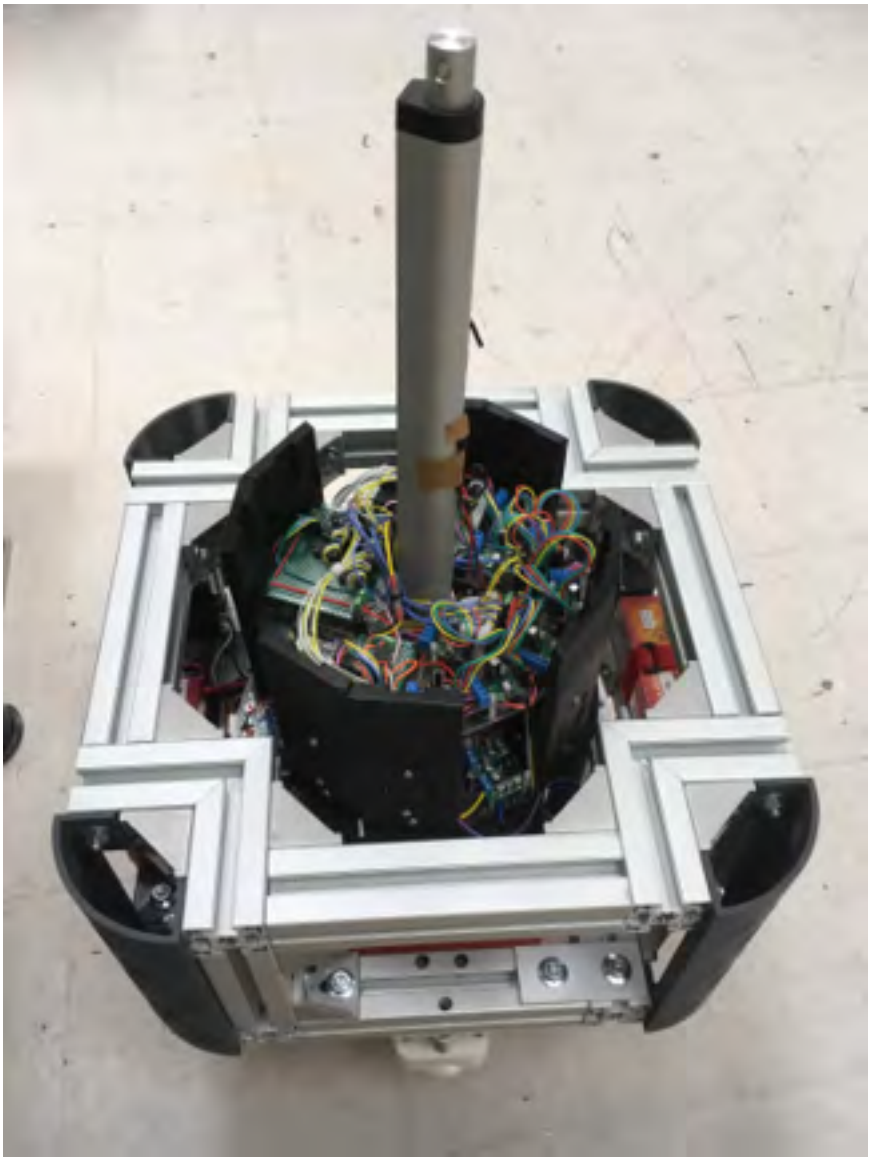
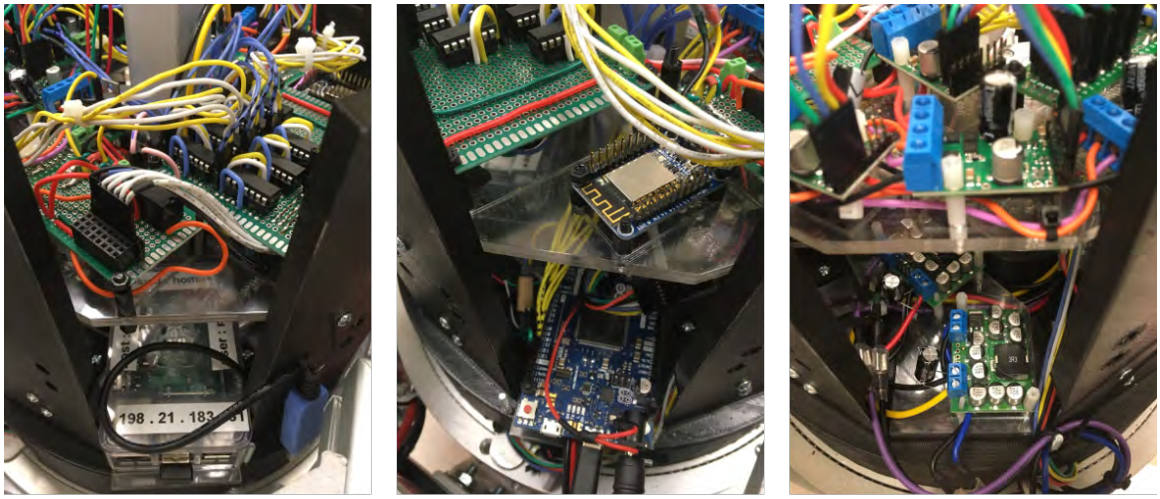
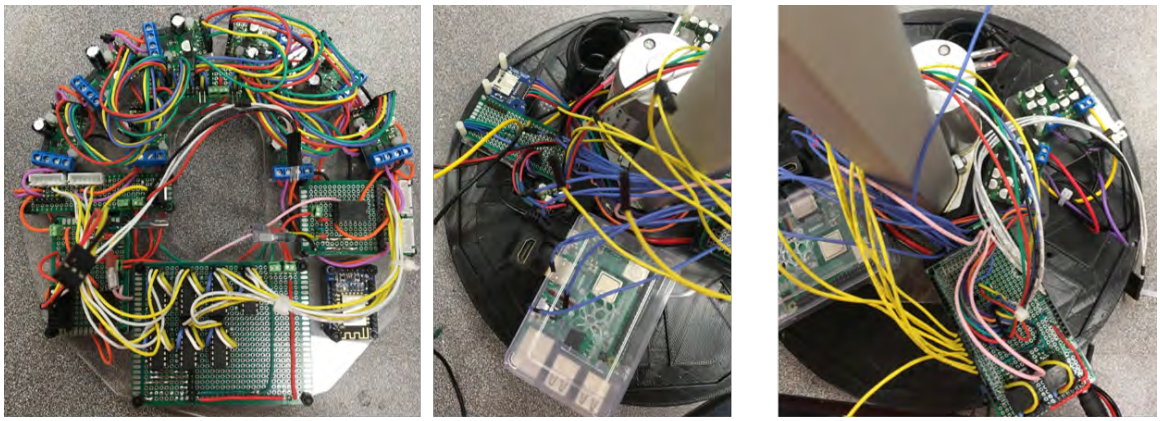


Figure 3.3: The central structure of CuRLE that houses all of the electronics is shown with the motor plate removed.



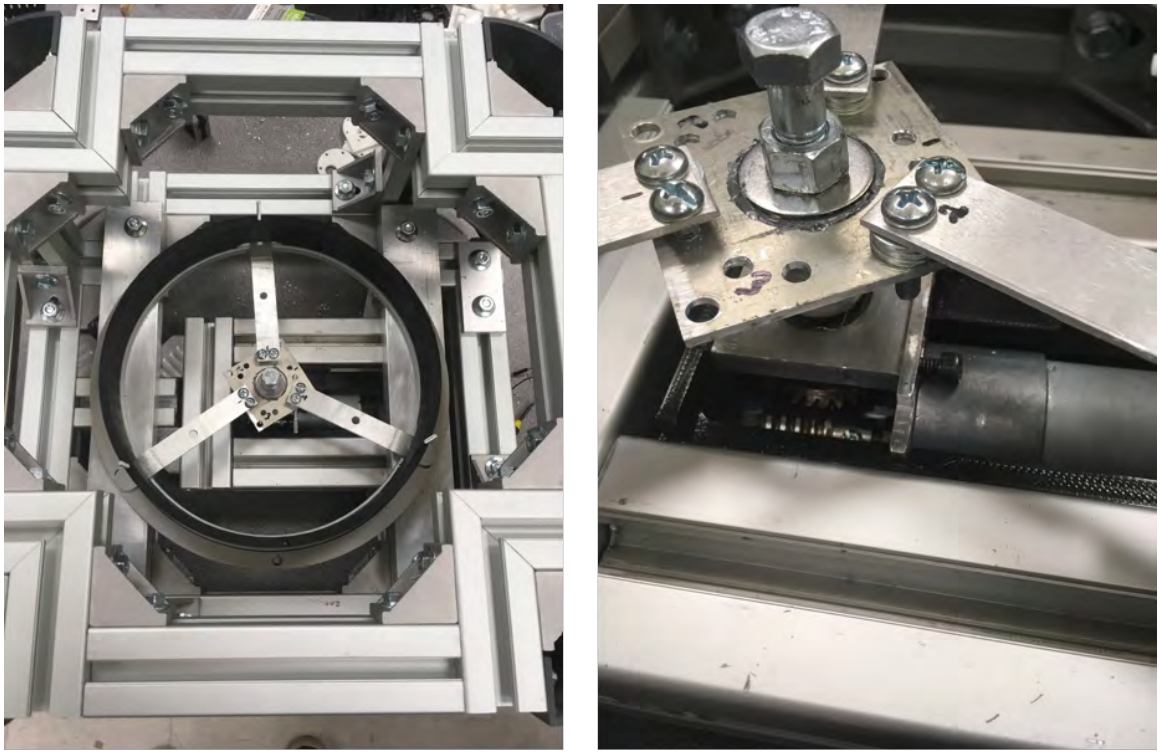
(a) (b) (c)

Figure 3.4: The assembled center structure with different electronic components visable. (a) The Raspberry Pi. (b) The Arduino Due (c) Voltage regulator (bottom) and motor drivers (top)



(a) (b) (c)

Figure 3.5: The two main electronics plates removed from the central structure.(a) Top plate. (b) Left-half and (c) right-half of bottom plate.



(a) (b)

Figure 3.6: (a) The turntable mechanism of CuRLE. (b) Zoomed in view of the connection between the worm drive and the turntable.

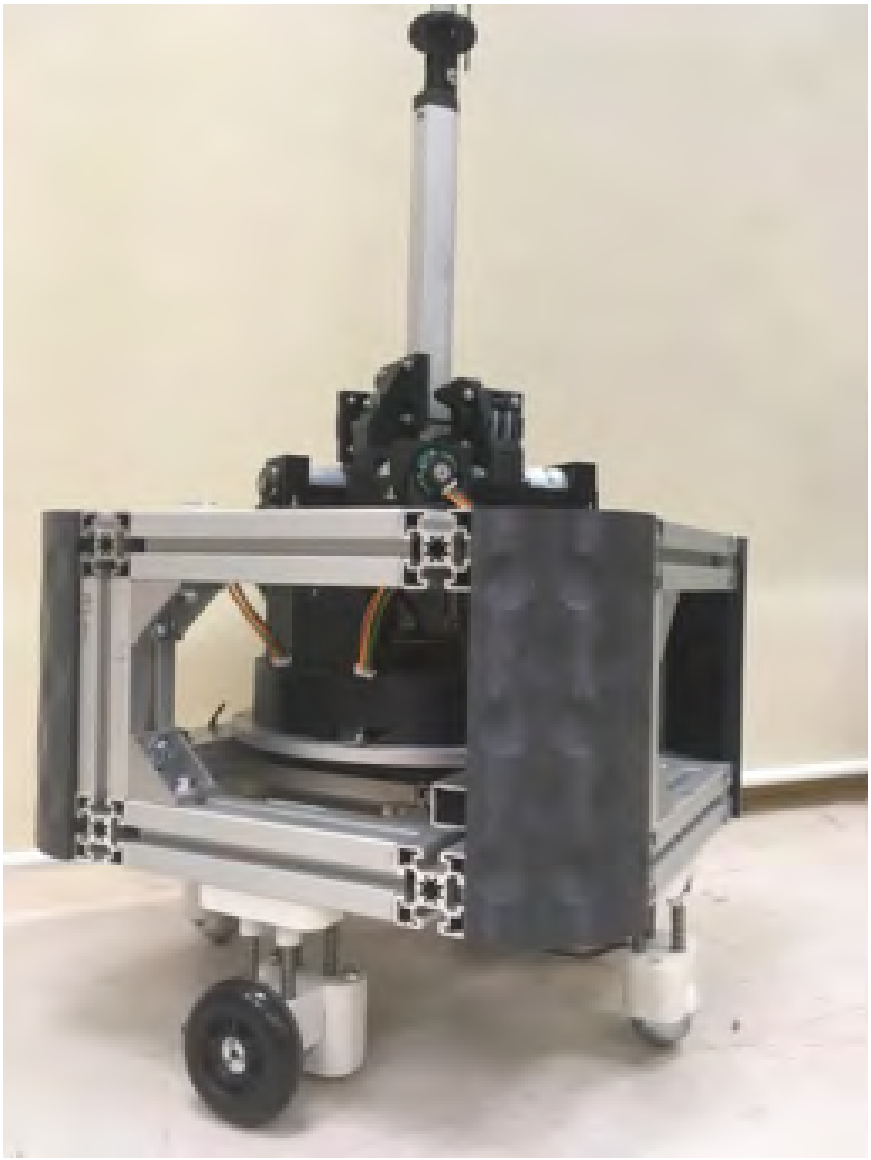


Figure 3.7: The central structure mounted on the turntable mechanism.

with an optical encoder. The worm drive is mounted to the base of the frame such that the axis of rotation of the worm gear is perpendicular to the plane of the floor and runs through the center of CuRLE (i.e. the axis of rotation is the central z-axis of CuRLE). The worm drive is attached to a turntable which consists of concentric metal discs attached by ball-bearings to facilitate rotation. The turntable mechanism is shown in Fig. 3.6. The outer ring of the turntable is fixed to the frame, and the central structure of CuRLE is mounted to the inner ring (Fig. 3.7). As such, when the worm drive actuates, the entire center structure, including the continuum element, rotates around the central z-axis of the robot.

Since the drive motors and the two of the three power supplies do not rotate with the turntable, we used coiled wire to maintain the electrical connections as ω varies. Since these wires still have a maximum extension before disconnecting, the turntable is restricted to a set range of rotation, which is discussed further in Chapter 4.

3.3.2 Differential Drive Subsystem

To remove the ”rocking” behavior of the h+lamp's drive system, spring-loaded shocks were designed and built for the passive drive elements (ball-bearing casters) and the active drive elements (differential drive motors). These are shown in Fig. 3.8 and Fig. 3.9 respectively. The shock system serves to remove the ”rocking”, and also to enable CuRLE to navigate uneven terrain. Since CuRLE's operating environment is the home of the user, we envision varying floor surfaces (e.g carpet, tile, wood) and potential obstacles (e.g. clothing, children toys). With the suspension, CuRLE will successfully maintain the required contact with the floor to navigate the space.

The assembly is composed of two 3D-printed pieces: one (top) which mounts to the frame and one (bottom) which attaches to the drive element (both passive and active). Metal rods are attached to the bottom piece with extremely strong adhesive. The springs

Figure 3.8: (a) Top-side view of the passive drive element showing the through-holes for the metal rods that provide stability and alignment in the shocks. (b) Side view of the shock assembly mounted to the ball-bearing casters (passive drive element).

are then threaded into grooves in the bottom plastic such that the metal rods run through the spring. Plastic caps are threaded to the other end of the springs. These caps have linear ball-bearing bushings attached to the other end and the metal rods then run through these bushings. The plastic caps with bushings are then inserted into a groove into the top plastic and the same adhesive (the grey substance in Fig. 3.8 and Fig. 3.9) is used to weld the caps to the top piece. Holes run through the top piece for the metal rods to pass through as the springs compress. The springs were selected such that the estimated weight of the robot (50 lb) would compress the springs to half of the their total displacement (1 in), thereby allowing the shocks to increase/decrease equally to traverse uneven terrain.

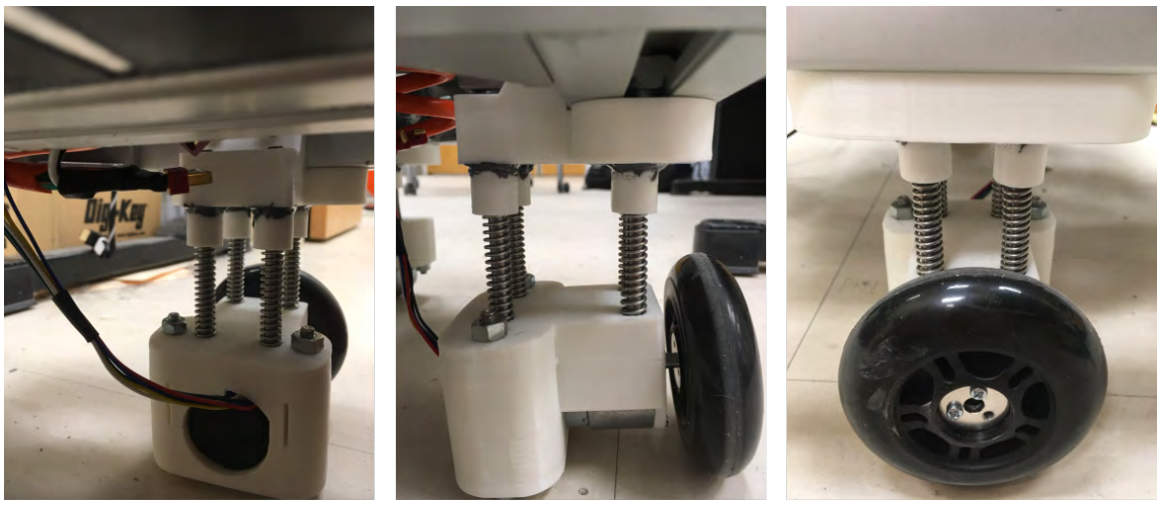


Figure 3.9: (a) Side view (under frame) of the shock assembly for one differential drive motor. (b) In-line view of the drive motor shock assembly. (c) Side view (outside frame) of the shock assembly for the drive motor.

3.3.3 Tendon Motors

To simplify the continuum kinematics, we modified CuRLE to a 4-tendon continuum section compared to the 3-tendon design in h+lamp. The new kinematics are detailed in section 3.4. With four tendons instead of three, control of only u and only v is possible by actuating a single pair of tendons. This, in a addition to new revolute variable ω detailed in section 3.3.1, allows us to easily restrict v to be constant (specifically $v = 0$) and still achieve a desirable workspace. The use and advantage of this will be discussed in Chapter 4.

The hardware to attach the motors was upgraded from that in the h+lamp to account for the increased motor count. In addition, we modified the spools the tendons wind around to more accurately measure and track the length of the tendon. We also added a tension sensing subsystem to prevent slack from developing in the tendons. Each motor assembly, consisting of a DC gear-motor with optical encoder, 3D-printed mount, 3D-printed spool,

tension sensor, and 3D-printed tension sensor mount, is mounted at a 90^o offset from its neighbors and is shown in Fig. 3.11(a). All four motor assemblies are attached to the top plate of the center structure, shown in Fig. 3.10.

3.3.3.1 Tension Sensors

Because the kinematics rely on accurately tracking the length of the tendons, it is essential that the tendons do not develop slack. Change in tendon length is calculated from the change in the motor encoder count based on the current circumference of the tendon spool. Slack in the tendon causes the amount of tendon spooled to be less than the amount reported by the motor encoder, thereby forcing the controller to evolve the continuum element into a shape that is not accurately defined by its current configuration.

The tension sensor consists of a spring-loaded linear potentiometer mounted in-line with the tendon. The tendon is threaded through an enclosed "spool", shown in Fig. 3.11(b), that is attached to the end of the tension sensor. Tension in the tendon compresses the potentiometer, and slack allows the it to extend. By keeping the "tension" reading above a suitably selected threshold, a controller keeps slack out of the tendon and enables accurate changes to CuRLE's configuration.

3.3.3.2 Tendon Spools

To accurately a measure change in tendon length, we measure how much of the tendon has wound/unwound from the spool by calculating the change in the motor's encoder counts. Knowing the ratio of counts measured (ΔE_{cnt} as determined by the encoder counter chips mentioned in section 3.2) to counts per revolution (C_{PR}) of the motor shaft and the

Figure 3.10: The motor plate which is mounted to the top of the central structure. Here is where the tendon motors are mounted with the tension sensors and the spools to wind the tendons and measure length.

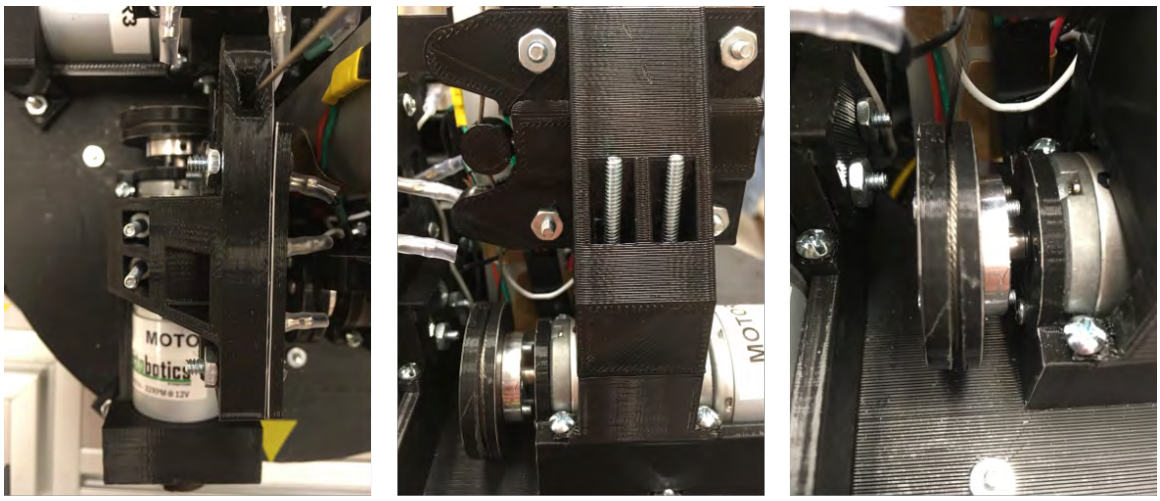


Figure 3.11: (a) The 3D-printed assembly of a single tendon motor

radius of the spool (r_{spool}) gives Eqn. (3.1) for the change in length (Δl).

$$\Delta l = \left(\frac{\Delta E_{cnt}}{C_{PR}} \right) * \left(2\pi r_{\text{spool}} \right) \tag{3.1}$$

As the tendon winds/unwinds around the spool, r_{spool} increases/decreases. To accurately "track" r_{spool}, we designed the spool groove to be the width of the tendon (shown in Fig. 3.11(c)), which guarantees, in theory, that every revolution of the spool increases the radius by exactly the width of the tendon (1mm in the case of CuRLE). By tracking the revolutions of the spool (via the encoder counts) we can thereby accurately measure length and thereby achieve valid configurations.

3.3.4 End-Effector

The gripper of the h+lamp was limited to open-loop control due to the lack of feedback on its controlling motor. As such, the motor would often continue to "grasp" after the object was already acquired, causing the plastic joints to jam and eventually snap. The

new gripper's "grasping" mechanism is controlled by a servo motor with built-in position control. By varying the duty cycle of the controlling PWM signal, this gripper will evolve its configuration and hold a specified position.

The new gripper, seen in Fig. 3.12(a-b), also contains second servo motor (identical to the first) that functions as a "wrist" for the end-effector. The wrist (variable γ) enables the gripper to grasp items in configurations that its predecessor could not. The nature of continuum element kinematics allows for a wide set of end-effector Cartesian locations, but few in this set orient the end-effector in a convenient way to grasp objects. The ability to rotate the end-effector via the wrist increases the set.

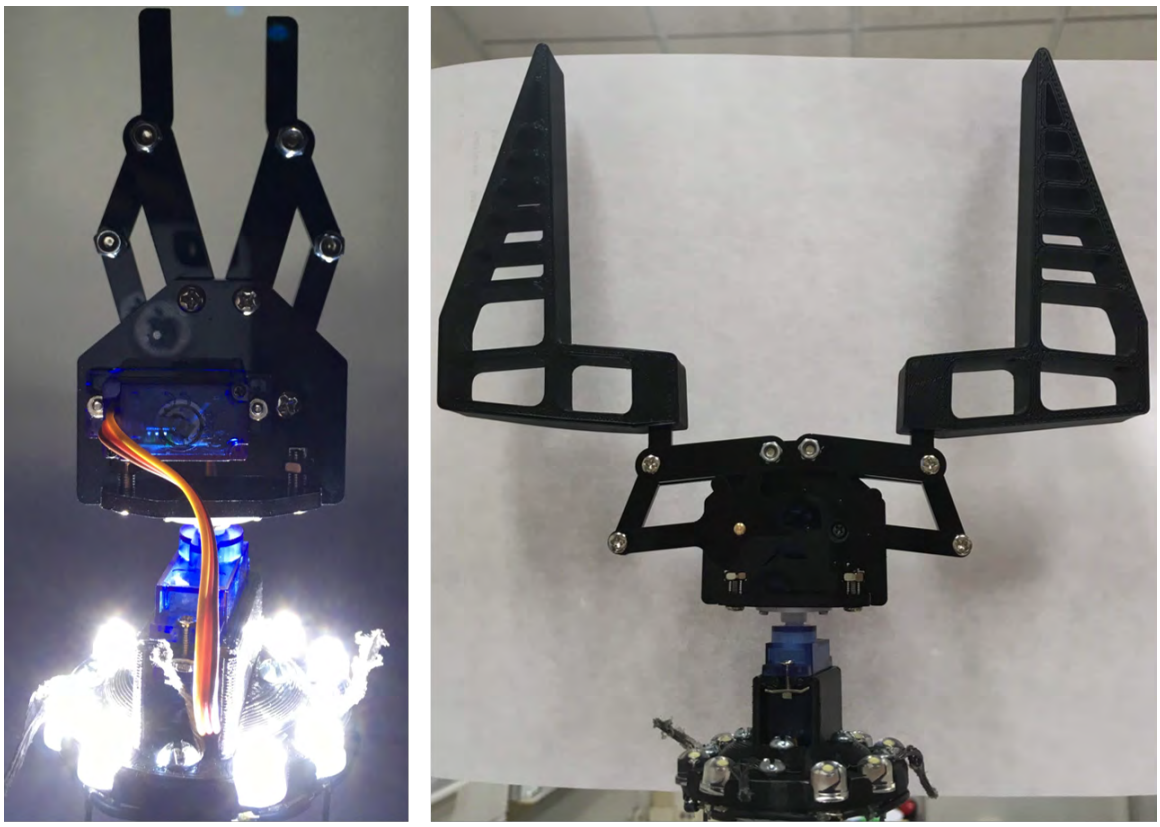
The final upgrades to the gripper are flexible "finger" attachments that enhance grasping. The "fingers" are 3D printed using a flexible thermoplastic polyurethane material that wraps around an object when the claw grasps it. These "fingers" are shown in Fig. 3.12(b).

3.3.5 Lighting Elements

Fig. 3.12(b) shows the LED's mounted on the end-effector to realize CuRLE's function as a lamp. There are eight 9mm white LEDs connected in parallel with a 5V source. A digital switch controlled by the Arduino Due toggles the LED's. In addition, an LED strip was run down the center of the PEX backbone to provide more lighting to the user. The lit LED strip is shown in Fig. 3.1.

3.3.6 Power Subsystem

Since CuRLE now possess more actuators and electronics than its predecessor, we installed additional power supplies to robot. The 4S LiPo battery used to power the motors in h+lamp (see Chapter 2) was replaced with a 6S LiPo with a greater capacity. The same



(a) (b)

Figure 3.12: (a) The gripper mounted as CuRLE's end-effector with the LED lights lit and the flexible grasping "fingers" removed. (b) The gripper with the "fingers" added.

12V switching voltage regulator provides a constant DC supply to the drive motors, linear actuator, tendon motors, and worm drive motor. The 4 AA batteries used to power the electronics in h+lamp were replaced with a 4S LiPo battery to power the new electronics in CuRLE. This supply is passed to two separate voltage regulators, one that provides 12V and one that provides 5V, to power the Arduino Due and Raspberry Pi, respectively. Finally, we installed a third 4S LiPo battery to power the new LED strips and the servo motors of the gripper. This battery is also connected to a 12V (LED strip) and a 5V (LED's and servo motors) voltage regulators. Fig. 3.13 shows the power supplies of CuRLE.

3.4 Kinematics

We next present the kinematics for a single section 4-tendon continuum section. While these equations hold for the more general case of an extensible section, the arc length s of the continuum element remains fixed in CuRLE. The tendons are arranged such that tendons 1 and 3 (variables l_1 and l_3) are an opposing pair in the v-plane ($u = 0$) and tendons 2 and 4 (variables l_2 and l_4) are an opposing pair in the u-plane ($v = 0$).

$$l_1 = s + (-d) \cdot v \tag{3.2}$$

$$l_2 = s + (d) \cdot u \tag{3.3}$$

$$l_3 = s + (d) \cdot v \tag{3.4}$$

$$l_4 = s + (-d) \cdot u \tag{3.5}$$

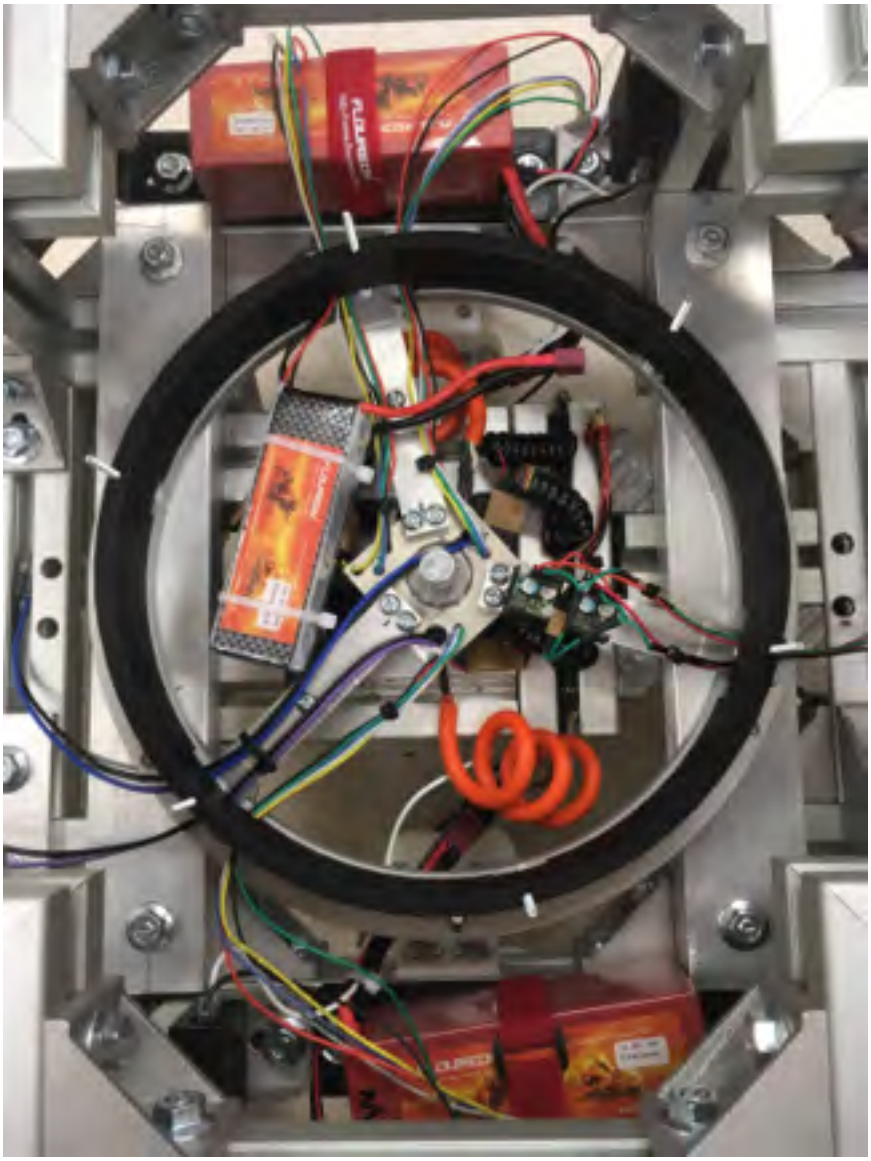


Figure 3.13: The power system of CuRLE. (top) 4S LiPo battery to power the electronics. (middle) 4S LiPo battery to power the LEDs, LED strip, and servo motors in the gripper. (bot) 4S LiPo battery (later replaced with a 6S) to power all of the motors.

$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix} = \begin{bmatrix} 0 & -d & 1 \\ d & 0 & 1 \\ 0 & d & 1 \\ -d & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ s \end{bmatrix} \tag{3.6}$$

$$u = \frac{l_2 - l_4}{2d} \tag{3.7}$$

$$v = \frac{l_3 - l_1}{2d} \tag{3.8}$$

$$s = \frac{l_1 + l_3}{2} = \frac{l_2 + l_4}{2} = \frac{l_1 + l_2 + l_3 + l_4}{4} \tag{3.9}$$

$$\begin{bmatrix} u \\ v \\ s \end{bmatrix} = \left(\frac{1}{4d}\right) \begin{bmatrix} -2 & 0 & 2 & 0 \\ 0 & 2 & 0 & -2 \\ d & d & d & d \end{bmatrix} \cdot \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l4 \end{bmatrix} \tag{3.10}$$

Chapter 4

Configuration Space of CuRLE

This chapter starts by defining for the first time the general configuration space for a single section extensible continuum element [25]. We identify several interesting and unique practical aspects of continuum section C-space, particularly in the case of tendon-actuated sections. To highlight these unique aspects of continuum robots, we compare to this configuration space with that of a rigid-link robot that is selected to have a similar task space. We then discuss the constraints to the continuum configuration space that are imposed by the physical construction of CuRLE. Finally, we discuss the configuration space of CuRLE's mobile base, and how we can isolate it from the configuration space of the continuum section by making suitable assumptions about the task space.

4.1 Continuum Configuration Space

We begin by considering the underlying structure of continuum robot configuration space in the presence of physical and actuation constraints. Specifically, we consider the c-space of the basic element of continuum robots: a single extensible section. We assume the section to be of constant curvature, a common assumption in the literature [4].

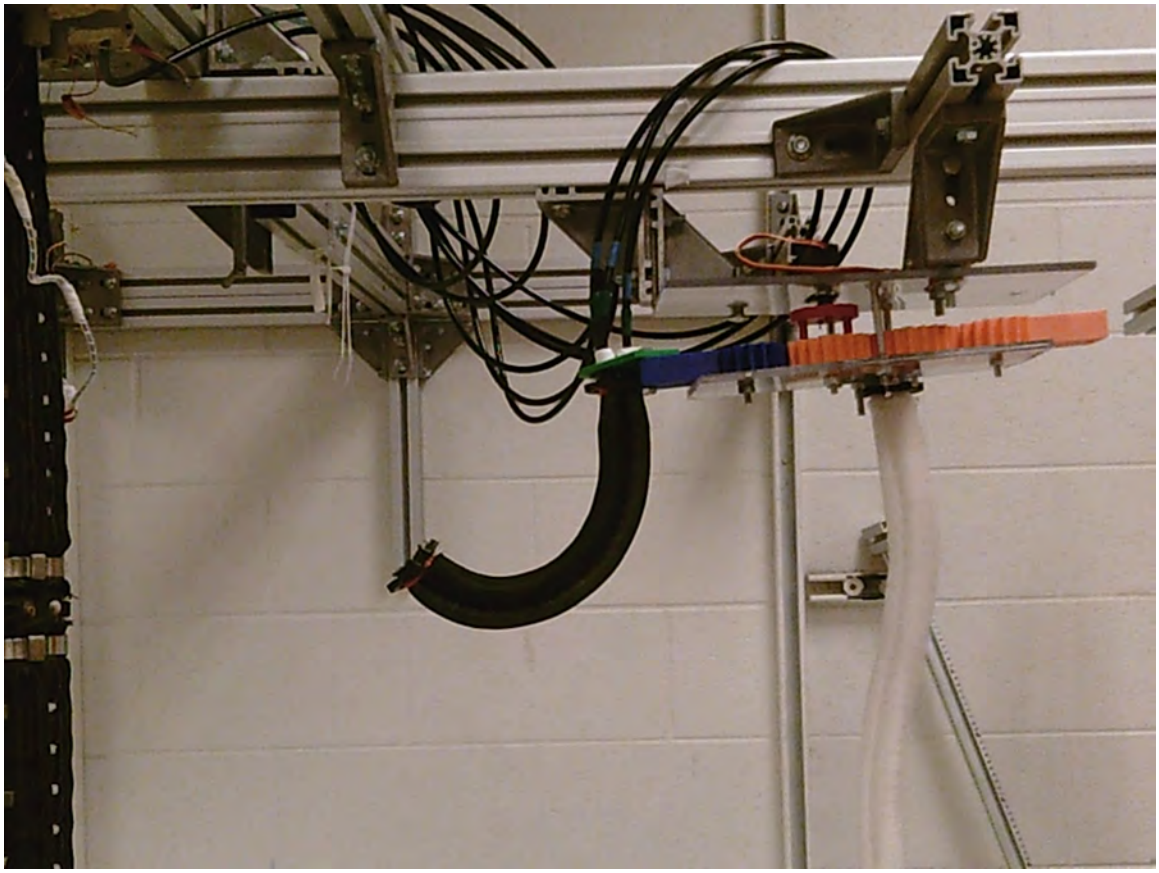


Figure 4.1: An example of a single section extensible continuum robot [2].

4.1.1 Single Section Continuum Robot

A single section extensible continuum robot, such as the one in Fig. 4.1, has three Degrees of Freedom (DoF) and can be described by three kinematic variables: $\{u, v, s\}$ where is s is the arc-length of the section and u and v represent the components of a rotation axis with respect to the base of the section [26]. Fig. 4.2 illustrates this. Let $c \in C^3_{space}$ be a configuration in the configuration space of the robot where $c = [u \ \ v \ \ s]^T$.

To better envision C^3_{space}, let us first consider the configuration space C^1_{space} where only u varies. We thus restrict the length $s = s_{fixed}$ and set $v = 0$. A configuration is then defined as $c = [u] \in C^1_{space}$. As we vary u in the positive direction (equating to counter-clockwise rotation), the section will bend to the left in the yz-plane, as shown in Fig. 4.3a.

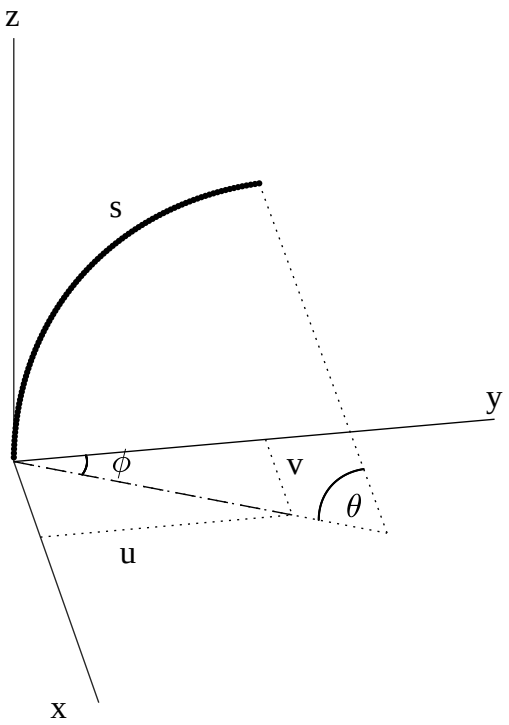


Figure 4.2: A simple sketch demonstrating the kinematic variables of a single section extensible continuum element.

Once $u = 2\pi$, the section's tip will meet the base and form a perfect circle. Continuing to increase u will cause the robot to "bend within itself"' and theoretically it would continue to "encircle" itself as $u \to \infty$. Increasing u in the negative direction (clockwise) will cause the same planar motion mirrored across the z-axis (Fig 4.3b). As $u \to (-\infty)$, the robot will continue to encircle itself to generate the remaining set of possible planar configurations of the section. Therefore, the configuration space of the robot where only u varies is $C^1_{space} \equiv \mathbb{R}$.

If we remove the restriction on v and allow it to also vary, then the configuration space changes to a 2D space where any configuration is defined as $c = [u \ v]^T \in C^2_{space}$. The

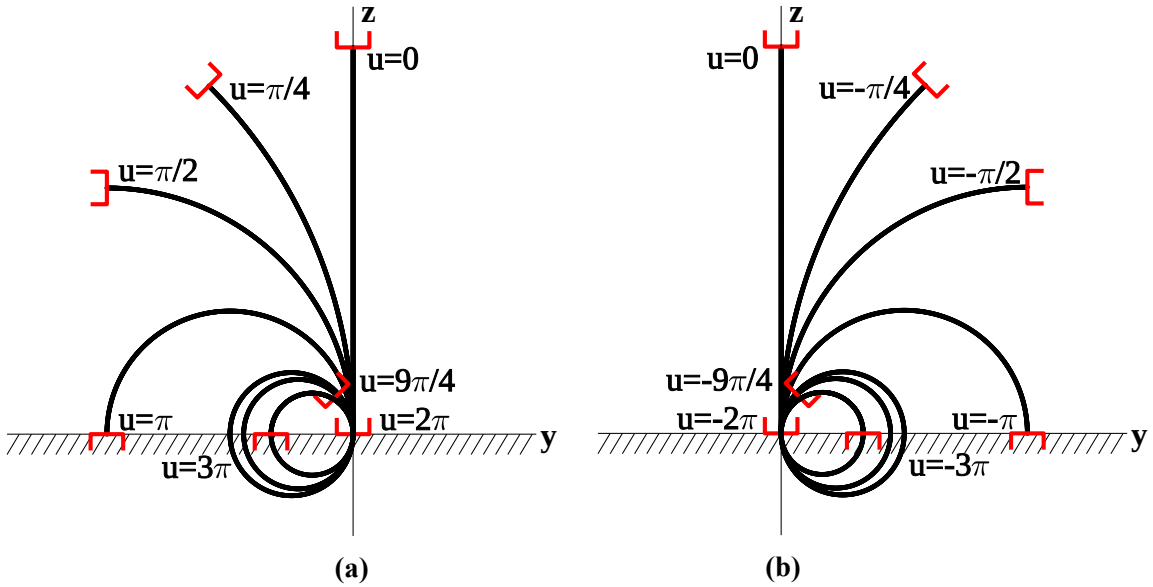


Figure 4.3: Single section continuum robot bending (a) counter-clockwise and (b) clockwise in the yz-plane.

total "bend" of the robot, θ, in the plane of curvature is defined by [26].

$$\theta = \sqrt{u^2 + v^2} \tag{4.1}$$

When $\theta = 2\pi$, the section once more forms a perfect circle, with its tip touching the base. Varying the vector $[u \;\; v]^T$ generates all bending directions (planes of curvature) and increasing the magnitude of the vector generates all possible configurations in each of these planes via (4.1). Since $u, v \in \mathbb{R}$, $C^2_{space} \equiv \mathbb{R}^2$ and can be visualized as the infinite plane described in Fig. 4.4.

If we now allow s to vary as well, a configuration is defined as $c = [u \;\; v \;\; s]^T \in C^3_{space}$. Since $s \in (0, \infty)$, then $C^3_{space} \in \mathbb{R}^3$ s.t. $s > 0$. Fig. 4.5 illustrates this space.

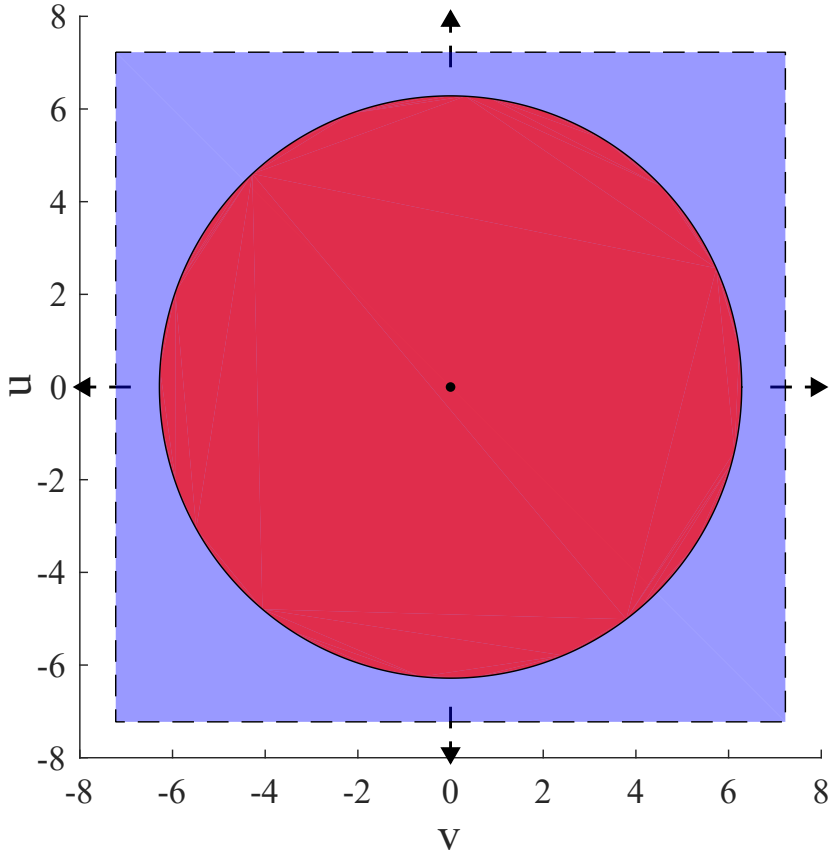


Figure 4.4: A visualization of C^2_{space}. The blue plane extends to $\pm\infty$. The red circle indicates C^2_{space} with the physical constraint of $\theta \leq 2\pi$.

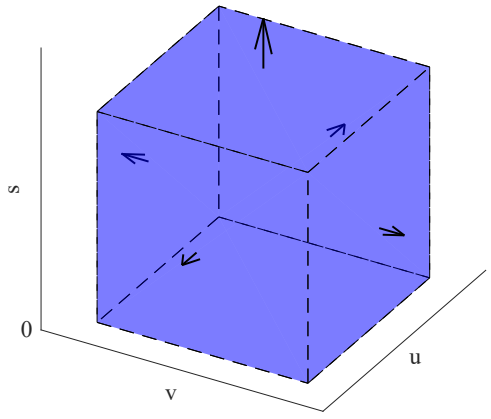


Figure 4.5: A visualization of C^3_{space}. The dotted lines bordering the solid and the arrows indicate that $u, v \rightarrow (\pm\infty)$ and $s \rightarrow (+\infty)$. The prism is bounded by the plane $s = 0$.

4.1.2 Physical Constraints in a Single Section Continuum Robot

At this point, however, we introduce constraints in the configuration space imposed by physical limitations of the robot. The first constraint to consider is on length. Any physical robot will have a maximum and minimum length, imposing an upper and lower bound on arc-length: $s_{min} \leq s \leq s_{max}$.

Another constraint is imposed by the physical width of the backbone of tendon actuated continuum robots. This physical distance, $d_r > 0$, is the distance from the center of the backbone to its outer edge. With s the length down the exact center of the backbone, and L_1 and L_2 the tendon lengths along its outside in the plane of bending (Fig. 4.6), when the robot is perfectly straight (i.e. $u = v = 0$), then $L_1 = L_2 = s$ (Fig. 4.6a). For the robot to bend counter-clockwise in the plane, the length of the left side of the robot, L_2, must shorten at the same rate that the length of the opposite side of the robot, L_1, lengthens. This is illustrated in Fig. 4.6b.

Because of this, the section cannot bend at all when it is at maximum or minimum length. When $s = s_{max}$ and $u = v = 0$, then $L_1 = L_2 = s_{max}$. To bend, L_1 or L_2 must lengthen, but each is already at the maximum length. The same reasoning is applied when $s = s_{min}$. At maximum/minimum length, $C^3_{space} = \{ \begin{bmatrix} 0 & 0 & s_{max/min} \end{bmatrix}^T \}$. For intermediate values of s, a similar situation holds —bending can occur up to one tendon achieving max length. The practical configuration space can now be visualized in Fig. 4.7(a). When $s = \frac{s_{max} - s_{min}}{2}$ the robot will be able to achieve the greatest amount of bending and will have the largest "*uv*-plane".

This pyramid shape assumes there are a pair of opposing tendons in the u-plane and another pair of opposing tendons in the v plane, which is consistent with the CuRLE hardware. The flat surfaces of the pyramid reflect this alignment, meaning that if the pairs tendons were rotated to align in other planes, the pyramid would rotate such that the flat

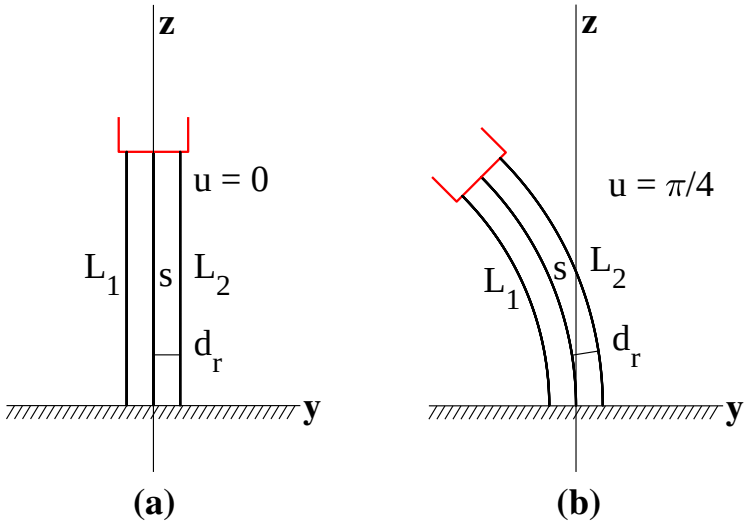


Figure 4.6: An illustration of physical constraints of bending a continuum robot. In (a), $L_1 = L_2 = s$. In (b), the robot has bent counter-clockwise, causing L_2 to lengthen and L_1 to shorten, while s remains constant.

surfaces would face the tendon directions.

A further practical constraint arises due to "encircling" imposed when $\theta > 2\pi$. Given a specific physical construction of a practical continuum section, it is likely that it will not be able to "encircle" itself. Even if it could, it cannot continue doing so as u, $v \to (\pm)\infty$. Therefore, there will exist some boundary for u and v imposed by physical constraints. In this thesis, and consistent with our hardware, we set this boundary to be at $\theta = 2\pi$, which bounds C^2_{space} as shown in Fig. 4.4. Expanding the $\theta \leq 2\pi$ constraint to C^3_{space} gives the space seen in Fig. 4.7(b), which is the practical configuration space for a single section extensible continuum robot with physical constraints exploited herein.

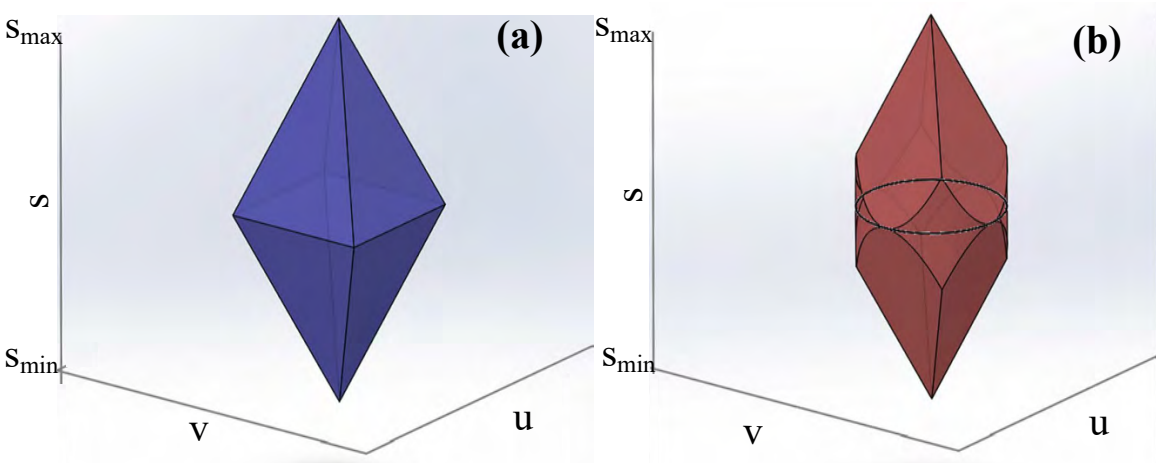


Figure 4.7: A visualization of C^3_{space}. In (a) the physical constraints of the backbone are illustrated. The maximum bend can be achieved when $s = \frac{s_{max}-s_{min}}{2}$, which is the widest plane in the center of the pyramid. In (b), the physical constraint of $\theta \leq 2\pi$ is applied to (a), which forms the "rounded" pyramid shape. The largest "uv-plane" indicated circle in (b) is the same circle shown in Fig. 4.4.

4.2 A Comparison: Equivalent Rigid-Link Robot Configuration Space

To highlight the unique issues presented by continuum section structures, we compare with the case of a kinematically similar rigid link robot.

4.2.1 Equivalent Rigid Link Robot

To analyze a rigid link robot structure with the same DoF as the continuum robot section, we use for comparison a 3 DoF robot (Fig. 4.8) with a constrained RRPRR joint configuration similar to the planar RPR robot described in [27].

In this, we constrain the third and fourth revolute joint angles to exactly match the first and second revolute joint angles, respectively, which gives the configuration vector $q = [\theta_1 \ \theta_2 \ d \ \theta_1 \ \theta_2]^T$. The prismatic joint can extend/retract between known maximum and

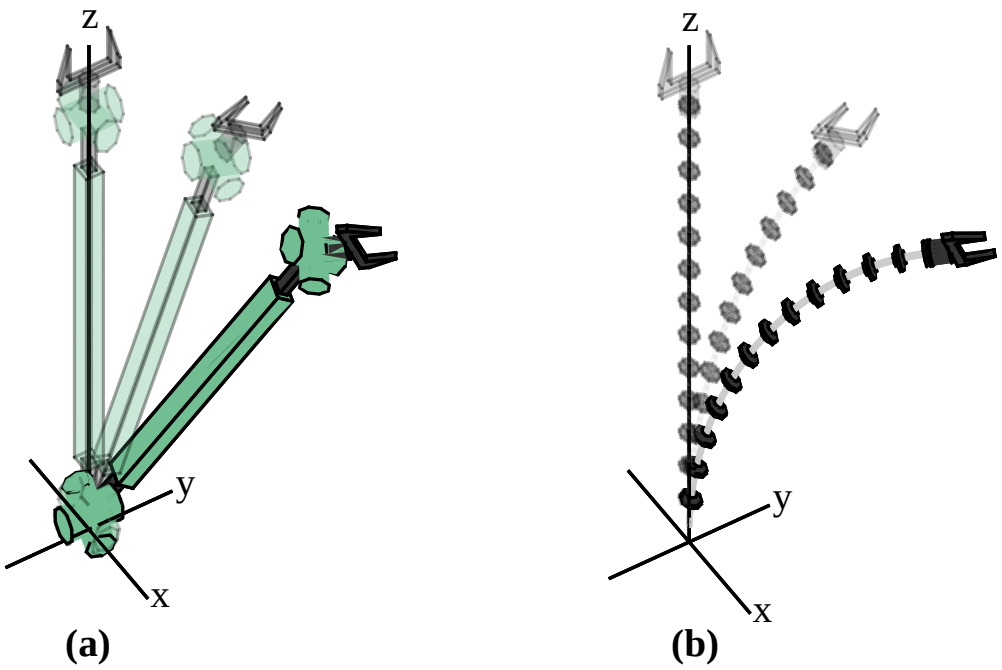


Figure 4.8: A sketch showing (a) the task-space-equivalent rigid-link RRPRR robot in the same configuration(s) as (b) the continuum element. This is the result of the kinematic mapping F.

minimum lengths. We select this rigid link configuration since we can construct a kinematic mapping between its configuration space and the configuration space of the continuum robot section that restricts the rigid-link robot task space to the equivalent task space of the continuum section. This mapping, F, is described in section 4.2.4. Fig. 4.9 shows the two robots sharing the same task space.

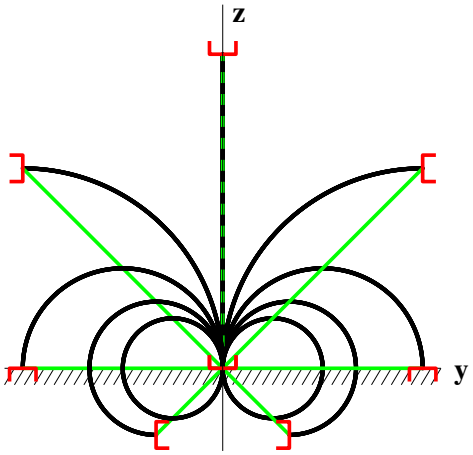


Figure 4.9: The task space of both the continuum section (black) and rigid link structure (green) for different values of u for the continuum section and $[\theta_1, d]$ for the rigid-link robot.

4.2.2 Rigid-Link Robot Configuration Space

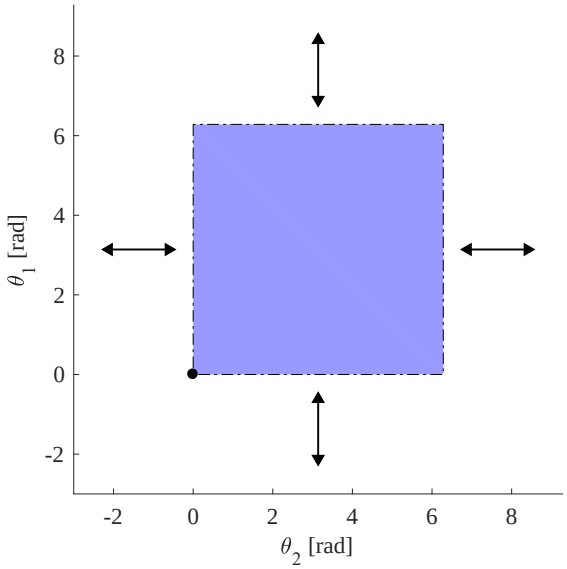


Figure 4.10: Q^2_{space} of the rigid-link robot.The arrows indicate the "wrapping" phenomenon that occurs when θ_1 and θ_2 go beyond the bounds $[0, 2\pi)$.

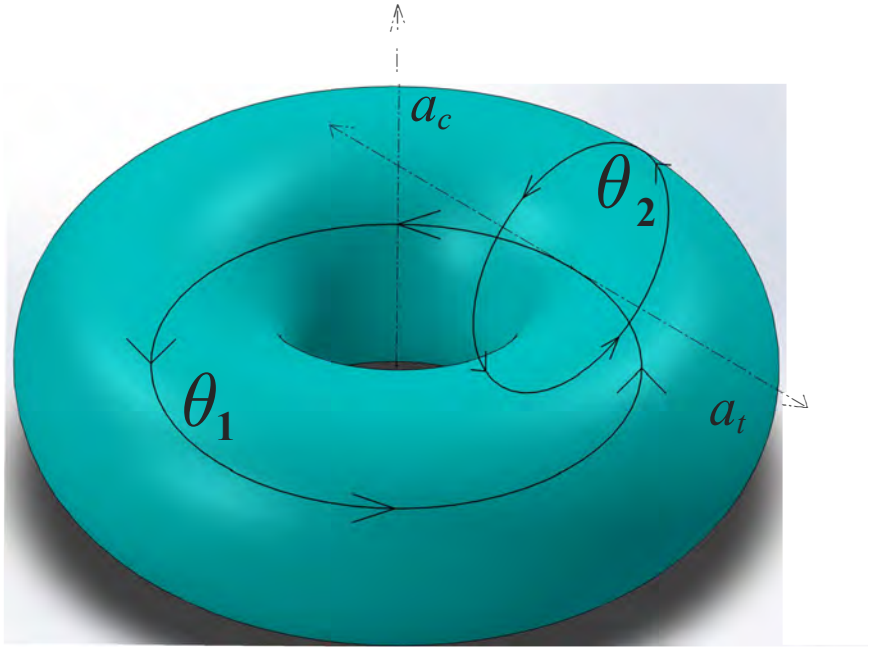


Figure 4.11: A visualization of Q^2_{space} of the rigid-link robot to show the "wrapping" phenomenom. A configuration q is any point on the surface. Changing θ_1 "rotates" q around the axis, a_c, running through the center of the torus. Changing θ_2 "rotates" q around a_t which is the tangent to the path of rotation of θ_1.

For the two independent revolute DoF: $\theta_1, \theta_2 \in [0, 2\pi)$ we define $q = [\theta_1 \;\; \theta_2]^T \in Q^2_{space}$. Rather than the infinite plane in Fig. 4.4, the space manifests as a square with a "wrapping" phenomenon that causes $\theta_1, \theta_2 \geq 2\pi, \vee\ \theta_1, \theta_2 < 0$ to "wrap" back to $0 \leq \theta_1, \theta_2 < 2\pi$, as seen in Fig. 4.10. This space, while locally 2D, is globally best visualized as the surface of a torus, like that in Fig. 4.11.

Adding the prismatic joint modifies the "square" in Fig. 4.10 into the "rectangular prism" shown in Fig. 4.12. As with the Q^2_{space}, the same "wrapping" phenomenon occurs whenever one of the joints goes beyond 0 or 2π. The configuration space can be defined as $\forall q = [\theta_1 \;\; \theta_2 \;\; d]^T \in Q^3_{space}$ s.t. $0 \leq \theta_1 < 2\pi,\;\; 0 \leq \theta_2 < 2\pi,\;\; d_{min} \leq d \leq d_{max}$.

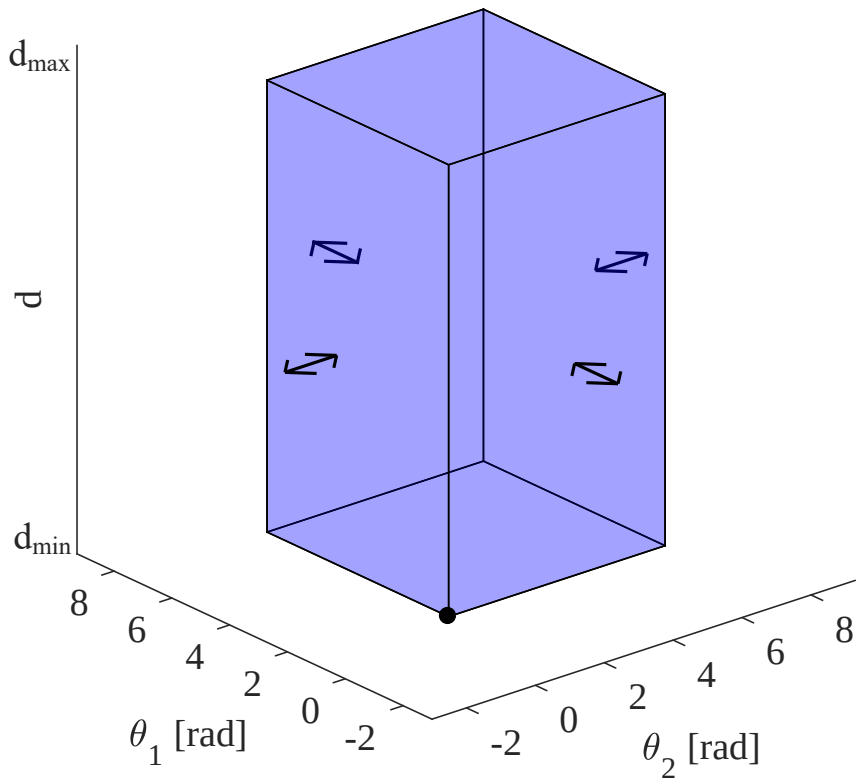


Figure 4.12: Q^3_{space} of the rigid-link robot. The arrows indicate the "wrapping" phenomenon that occurs when θ_1 and θ_2 go beyond the bounds $[0, 2\pi)$.

4.2.3 C-Space of Continuum Section vs Rigid-Link Structure

The key difference between the continuum and rigid-link configuration spaces (c-space) is the "wrapping" phenomenon that occurs in Q^3_{space}. In the ideal continuum c-space, there exists exactly 1 straight path connecting any two configurations $c_1,\ c_2 \in C^3_{space}$. For the rigid-link robot, there are always 2 straight paths between any configurations that involve a change in θ_1 or θ_2. For configuration changes that exclusively involve the prismatic joint, there is exactly 1 path.

An interesting difference between the c-spaces is their sizes. Since both C^3_{space} and Q^3_{space} are finite their sizes can be compared by calculating the volume of each space. This can be done by calculating the volume of the 3D shapes shown in Fig. 4.7(b) and Fig. 4.12.

The size of Q^3_{space} is the volume of the rectangular prism. The size of C^3_{space} is the volume of the "rounded" pyramid, which is slightly less than the full 2- sided square pyramid but greater than a 2-sided cone where the base has $radius = 2\pi$.

$$(4\pi^2)(s_{max} - s_{min})\left(\frac{\pi}{3}\right) < V_{\text{cont}} < (4\pi^2)(s_{max} - s_{min})\left(\frac{8}{3}\right)$$
$$V_{\text{rigid}} = (4\pi^2)(d_{max} - d_{min}) \tag{4.2}$$

Since the rigid-link robot was chosen to be kinematically similar to the continuum robot, we can conclude that the configuration space for a kinematically equivalent continuum section is larger than the rigid-link robot's configuration space (Eqn. 4.3).

$$s_{max} = d_{max} \ , \ s_{min} = d_{min}$$
$$\Rightarrow V_{\text{rigid}} < V_{\text{cont}} \tag{4.3}$$

4.2.4 Ensuring Equivalent Task Spaces

The only difference between the two task spaces is the physical shape of the arm of the robot that creates them. This is either a constant curvature curve between the base and the end-effector (continuum) or a straight line (rigid-link), as shown in Fig. 4.9. This shape, however, is important in the context of motion planning, as it has to be accounted for when checking for collision-free paths through the space.

For the rigid-link robot to have the same task-space as the continuum robot, we construct a function F that maps every configuration $c \in C^3_{space}$ to a configuration $q \in Q^3_{space}$. This function, F, is neither one-to-one nor onto, and is shown in Eqn. (4.4).

$$F : C^3_{space} \rightarrow Q^3_{space} \text{ s.t. } F(c) = q \text{ where}$$

$$c = \begin{bmatrix} u \\ v \\ s \end{bmatrix} \in C^3_{space} \text{ and } q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d \end{bmatrix} \in Q^3_{space}$$

$$\Rightarrow F(c) = \begin{bmatrix} \frac{u}{2} \\ \frac{v}{2} \\ \left(\frac{2s}{\sqrt{u^2+v^2}}\right) sin\left(\frac{\sqrt{u^2+v^2}}{2}\right) \end{bmatrix} \tag{4.4}$$

Let $Q^C_{space} \triangleq \mathbb{R}\{F\}$ where $\mathbb{R}\{F\}$ is the range of the function F. Then $Q^C_{space} \in Q^3_{space}$ is a subspace of Q^3_{space}. Since C^3_{space} is centered at $u = v = 0$, and has a radius of 2π, Q^C_{space} will be centered at $\theta_1 = \theta_2 = 0$. Now, we restrict θ_1, θ_2 such that $\sqrt{\theta_1^2 + \theta_2^2} \leq \pi$. This is the effect of applying F to Eqn. (4.1). If this is not done, then the value of d, as described be Eqn. (4.4), would take on negative values, because of the sin $\left(\frac{\sqrt{u^2+v^2}}{2}\right)$ term. This restriction of θ_1, θ_2 also removes the "wrapping" phenomenon and causes Q^C_{space} to have the same "rounded" pyramid shape as C^3_{space}, as shown in Fig. 4.13.

Finally, by comparing (4.2) and (4.4), the volume of Q^C_{space} is significantly smaller than the volume of the C^3_{space}, but the task space is equivalent.

$$(\pi^2)(s_{max} - s_{min})\left(\frac{\pi}{3}\right) < V_{\text{equiv}} < (4\pi^2)(s_{max} - s_{min})\left(\frac{2}{3}\right) \tag{4.5}$$

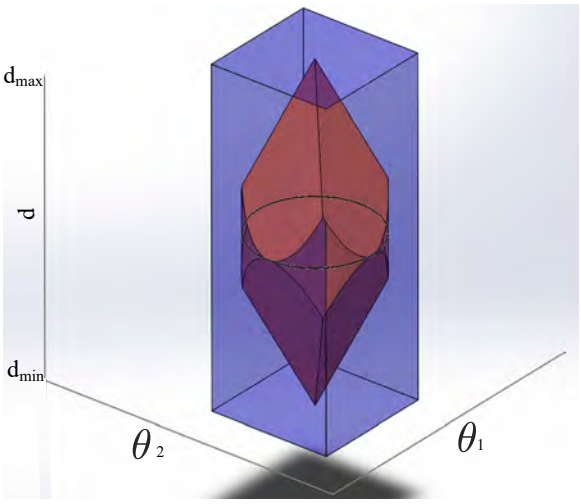


Figure 4.13: The configuration space (red) of the rigid-link structure, Q^C_{space}, once it has been restricted by F to have the equivalent task space as the continuum section. This is displayed within the full c-space (blue), Q^3_{space}, from Fig. 4.12

4.3 Continuum Robotic Lamp Element: CuRLE

Recall from Chapter 3, CuRLE is a tendon-driven, non-extensible, single-section continuum arm mounted onto a mobile base which is controlled by a differential drive. CuRLE 's end-effector is a 2-fingered gripper featuring a series of LEDs to give the lamp light. Additional lighting is provided by LED strips inside the continuum body of the lamp.

The continuum arm is mounted on a prismatic joint (variable length L) which serves to raise/lower the base of the continuum arm but does not change the continuum arc length s. The prismatic joint is further mounted on a revolute joint (variable ω) which allows the entire arm to be rotated about the z-axis (yaw). Another revolute joint (variable γ) is mounted at the end of the continuum arm to serve as a "wrist" for the gripper.

4.3.1 Adding Constraints

In order to visualize the configuration space of CuRLE and conduct practical motion planning through the home environment space, we constrain several DoF. For the re-

mainder of this work, we fix L to its minimum length. Since the continuum arm is non-extensible, the arc-length s is necessarily constrained to be constant. The revolute joint γ serving as the wrist for the gripper adds redundancy, but remains fixed in the experimentation described here.

With these restrictions, we discuss the mobile base and the kinematic variables $[\omega \ \ u \ \ v]$ for the continuum arm. Since the base serves to move the continuum lamp element through the home space and the continuum element performs manipulation, we divide the configuration space into two parts. We assume that there will not be obstacles that CuRLE has to "pass under" meaning that nothing in the task space would collide with the continuum arm but not the mobile base. As such, we form the configuration space of the continuum arm $c = [\omega \ \ u \ \ v] \in C_{space}$ and the configuration space of the mobile base $q = [x \ \ y \ \ \theta_b]^T \in C_{space}^{base}$.

4.3.2 Configuration Space of CuRLE

Recall that for a fixed arc-length s, a single section continuum robot has a practical configuration space of a circle bounded by $\theta = 2\pi$, shown in Fig. 4.4. Since the continuum arm of CuRLE can physically collide with the mobile base, we further modify the boundary to $\theta \leq \pi\sqrt{2}$ (the value of the θ when $u = v = \pi$).

The revolute joint at the base of the continuum arm is described by $\omega \in [0, 2\pi)$. In the ideal case, ω displays the same "wrapping" behavior that the revolute joints in the rigid-link configuration space. As such, adding ω changes the configuration space to 3 dimensions by revolving the "uv-circle" around a central axis. The shape, depicted in Fig. 4.14, echoes the torus described by Q^2_{space}, the c-space of 2 revolute DoF. Unlike Q^2_{space}, however, CuRLE 's is "solid", i.e. true 3D, meaning that configurations c are not limited to the surface of the torus only. ω selects the "slice" (a circle) of the torus and u and v select

the point c within that circle.

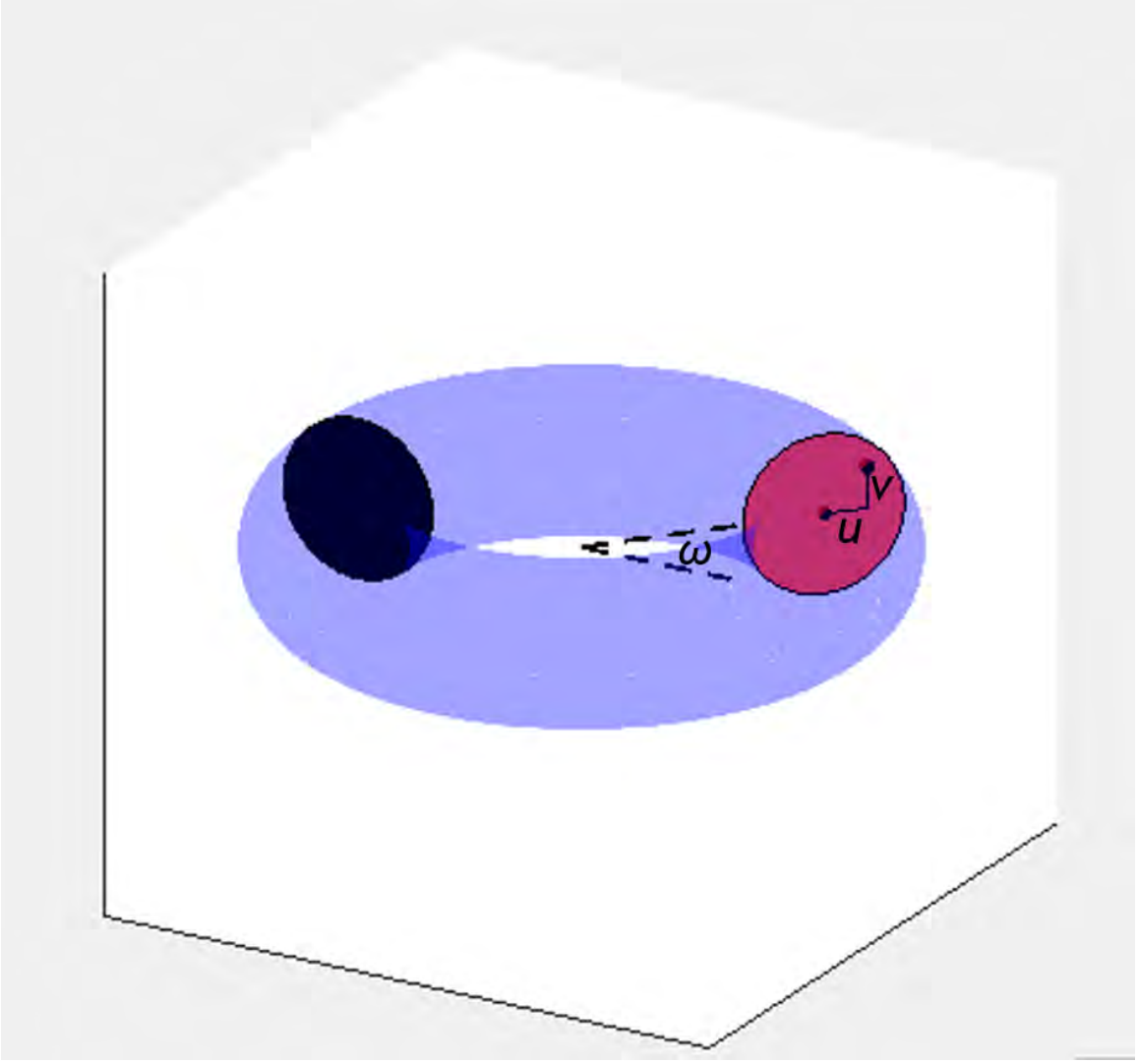


Figure 4.14: C_{space} of CuRLE.

Due to physical constraints, however, we limit $\omega \in [-\pi, \pi)$ and do not allow wrapping, which represented in Fig. 4.14 by the solid black plane at $\omega = -\pi$. Configurations take on any value in the volume except those in the black plane.

4.4 Configuration Space of Mobile Base

Since we separate the configuration space of the continuum element from the C-space of the mobile base by making the assumptions described above, we can formally define the configuration space of the mobile base C_{space}^{base} in (4.6) where q is a configuration of the robot in the configuration space C_{space}^{base}, x and y are the positions of the robot along the x-axis and y-axis respectively, and $\theta_b \in [0, 2\pi)$ is the orientation of the robot with respect to the positive x-axis.

$$\forall q \in C_{space}^{base} : q = [x\ y\ \theta_b]^T \tag{4.6}$$

The ideal configuration space (i.e. no physical constraints) has the shape of a rectangular prism with an infinite base (i.e. the x, y-plane) and a height, $\theta_b \in [0, 2\pi)$, of 2π. In any practical application, however, the parameters x and y will always be bounded by the walls of the room.

Obstacles in this configuration space are described as the Minkowski difference of the robot and the obstacle in the task space [11]. All the obstacles to the mobile base used in our experiments were designed to be convex polygonal in shape, since collision-avoidance can often be simplified by drawing a "bounding-box" around the obstacle. The base of CuRLE is a rectangular box with rounded corners (Fig. 4.15) meaning that the obstacles would resembled "twisted pillars" in the configuration space [13]. To again simplify the design, the robot is treated as a disc with a diameter equal to the diagonal of the frame. With the robot as a disc, collision detection can disregard θ_b and simply check for collisions in the x,y-plane [10].

For the robot to move between two configurations, q_0 and q_1, an action vector μ is

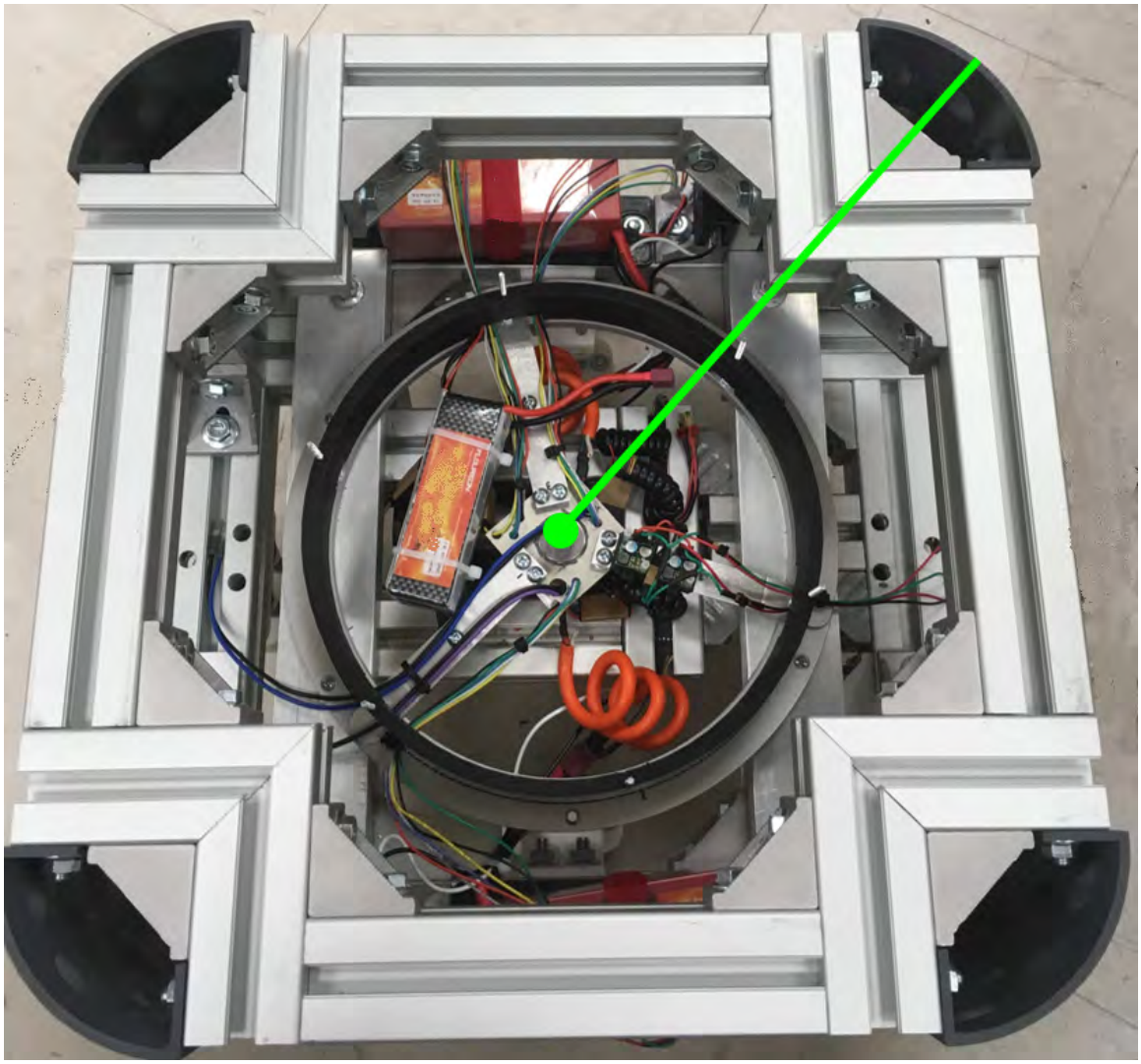


Figure 4.15: CuRLE's base is a square frame with rounded corners. The radius of the "disc" used to estimate the base of CuRLE is shown in green.

defined in (4.7) based on the robot kinematics.

$$\begin{aligned}
\mu : q_0 \rightarrow q_1, \text{where } q_0 &= [x_0\ y_0\ \theta_0]^T \text{and } q_1 = [x_1\ y_1\ \theta_1]^T \\
\text{then } \mu &= [\varphi_1\ \delta\ \varphi_2]^T \text{ where} \\
\varphi_1 &= min_rot\left(\arctan\left(\frac{y_1 - y_0}{x_1 - x_0}\right) - \theta_0\right) \\
\delta &= \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \\
\varphi_2 &= min_rot(\theta_1 - \varphi_1)
\end{aligned} \tag{4.7}$$

The $min\_rot(\varphi)$ function in (4.7) returns the minimum rotation needed to move between two angles. If the calculated rotation is greater in magnitude than π radians, then a rotation in the opposite direction provides an overall shorter rotation. (i.e. a counter-clockwise rotation might be shorter than rotating in a clockwise direction and vice versa).

The final aspect of defining the configuration space is a metric $d(q_0, q_1)$ that defines the "distance" between the two configurations q_0 and q_1 [13]. For CuRLE , the simplifications made to the configuration space allow the metric $d(q_0, q_1)$ to be defined as the Euclidean distance (L2-norm) between the x and y coordinates of the q_0 and q_1.

With the configuration spaces of the continuum element and the mobile base defined, we now have the foundation to discuss the motion planning algorithms we developed to navigate these spaces.

Chapter 5

Motion Planning

In this chapter, we introduce new motion planning algorithms specifically created for CuRLE and simulations used to test and verify them. First, we discuss the RRT used for the mobile base of the robot (Chapter 2), using the configuration space discussed in Chapter 4. Next, we detail the simulation environment for the mobile base used to test the algorithms' output. This is the simulation from the discussion in Chapter 2 which was used to verify the RRT before implementing on the h+lamp hardware.

We then discuss the necessary modifications made to the RRT to path plan for the continuum arm element of the robot. The new simulated environment developed to test this second algorithm is also described. Finally, we will show how the two RRTs can be combined to run in parallel and provide the results for those (simulated) experiments. The physical implementations corresponding to these simulated tests will be discussed in Chapter 6.

5.1 Path Planning for the Mobile Base

5.1.1 Rapidly-Exploring Random Tree (RRT) Algorithm

An RRT is a probabilistic approach to motion planning [11]. The objective is to connect a pre-defined start configuration with a desired goal location by constructing a tree of randomly selected nodes from the configuration space. The RRT is an iterative process that rapidly expands to cover the free-space of an environment. At each iteration, a random configuration is sampled and connected to the tree via the closest node that is currently in the tree, as long as the new node is located in the free-space and the new edge is collision-free. After a N iterations, the goal node is "selected" as the random node and connected to the tree. The start and goal configuration are now connected and a search algorithm (e.g. A*) can be run to determine the best path through the tree. Since the RRT is a probabilistic method, it will always connect to the goal provided that a solution exists and given infinite execution time. Since it randomly samples the free space to generate paths, it has a fast solve time for relatively open free spaces [13]. As such, it is often used as an anytime-approach in dynamic environments.

To implement the RRT, we used the open-source Boost Graph library for C++ [28]. Each vertex of the graph contains a configuration q, a node id, a heuristic value, and a cost value. Both heuristic and cost values are later used in the A* search algorithm (see below). The edges of the graph contain the distance between vertices, measured as the Euclidean distance, as well as the action vector, μ, to move between the two nodes. The graph is directed, connected, and contains no cycles. Most importantly, the graph can be reconfigured.

In our design, there are three types of RRTs that can be selected. These types are defined as VERTEX, EDGE, and EXT_VERTEX. The VERTEX type is the simplest RRT. As new nodes are randomly generated (q_r) and verified as valid configurations (i.e. no

collision with the configuration obstacles), the algorithm will search the graph for the node that is closest to q_r, using the metric $d(q_0, q_1)$ defined for the configuration space (L2-norm). Once the closest node is found, the two vertices are connected after first checking that the new edge does not pass through an obstacle and second, checking that the new edge is greater than a minimum threshold (passed as a configuration parameter to the RRT). This collision detection is done by sampling along the edge at a fixed δ, which varies based on the specific configuration variables used. The samples start at both the q_{closest} and q_r nodes and move towards the center of the edge. If any of the samples collide, then q_r is discarded and the loop continues. If there is no collision, then the edge distance is verified to be greater than the minimum threshold. This is done to prevent unnecessary nodes stacking on each other. If this threshold is met, then q_r is added to the graph along with the new edge connecting q_r and q_{closest} and the process repeats.

With the EDGE type of RRT, the edges of the graph are also considered when searching for the closest connection point. While this increases the search time, it generates a graph that better covers the free space. This process is done by calculating the normal distance from q_r to the line that contains each edge. The point of intersection, if it lies on the line segment that defines the edge, will become a new node in the graph q_i. The same process then repeats for collision detection of edges, this time checking the new edge between q_i and q_r. If all edges are collision free and greater than the threshold, then the old edge is deleted and both q_i and q_r are added to the graph, along with the appropriate new edges.

The EXT_VERTEX type of RRT extends the simple VERTEX type by breaking up long edges into multiple nodes and edges along the same path. The algorithm compares the new edge distance to a maximum threshold (passed as a parameter) and will split the edge based on the size. This method prevents the need to check edges, but helps create a wider spread of nodes that cover the free space. Fig. 5.1a-5.1c show a comparison of the three

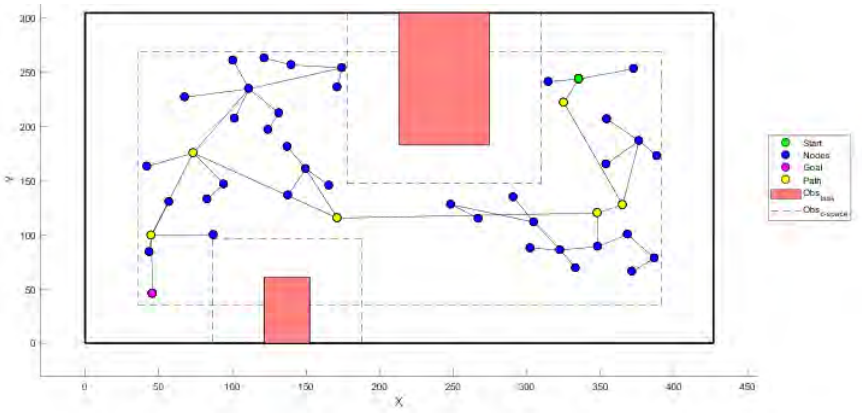
different types of RRT with same random seed, another configuration of the RRT.

To generate random nodes q_r, a separate uniform random number generator (URNG) is run for each variable of q. The bounds of each URNG correspond to the bounds of each variable in the configuration space (as detailed in Chapter 4). These values are specified to the RRT by a configuration file that is read at run-time. Fig. 5.2 shows that the spread of the RRT evenly covers the free space.

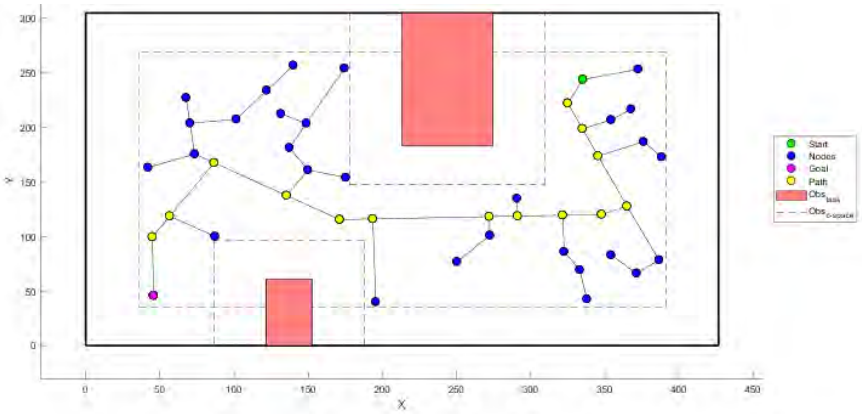
In addition to the parameters already discussed, (RRT type, minimum edge distance, maximum edge distance, random seed) the RRT can be configured in regard to how many nodes are generated. As with all RRTs, after a certain number of iterations of generating random nodes, the goal node is "selected" as the q_r and the algorithm then attempts to connect the goal to the tree [13]. For all our experiments, the algorithm searched for the goal 4% of the time. There is another parameter that specifies the minimum number of nodes in the tree. Even if the goal connects on the first try, it can be worth running the RRT longer to cover the free space more uniformly and thereby find better paths. The final parameter is the maximum number of nodes to add. This parameter serves to cut off the RRT after a certain point if it cannot reach the goal node. Since the RRT is a sampling based approach, it is only probabilistically complete. With infinite time, the goal will be reached if there exists a path. Rather than halt the RRT after a specified time, our algorithm halts after a maximum number of nodes are added and a flag is set to indicate if the goal was not found.

5.1.2 A*

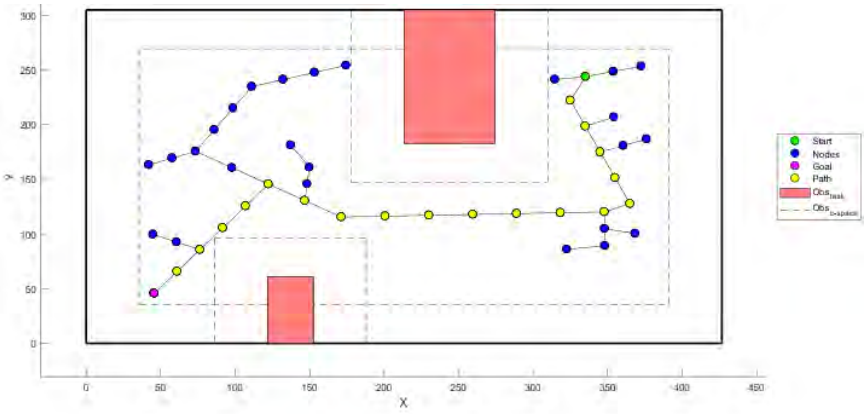
The output of the RRT is a connected graph of valid configurations for the robot. To evolve from the start to the goal configuration, an A* search algorithm is run on the graph. The A* heuristic function $h(q)$ is the L2-norm Euclidean distance from q to q_{goal}.



(a) RRT type VERTEX.

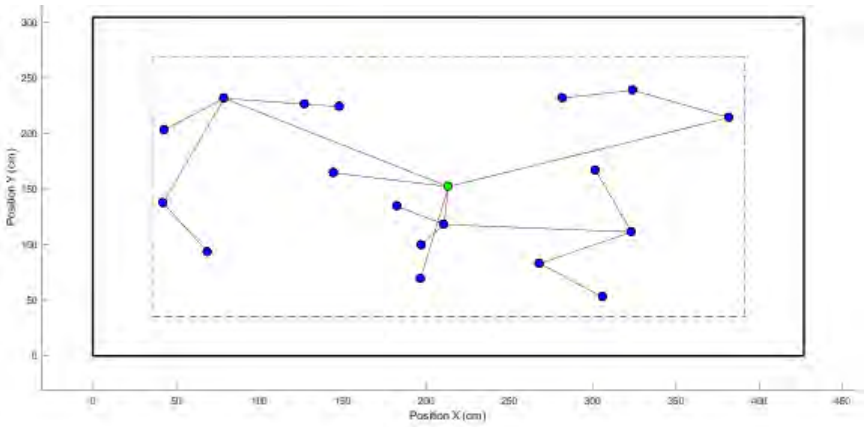


(b) RRT type EDGE.

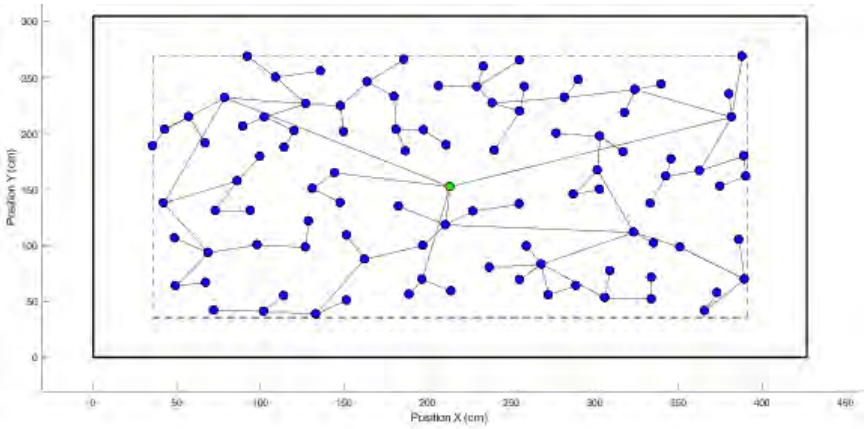


(c) RRT type EXT_VERTEX.

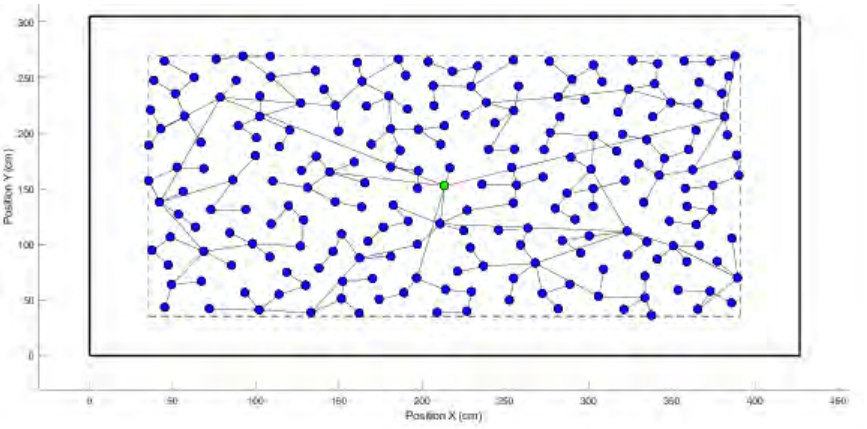
Figure 5.1: Plots showing the different graphs through the simulated environment for different types of RRTs. The path started at the green node and followed the yellow path to the magenta goal node, avoiding the configuration obstacles defined by the slashed lines surrounding the red obstacles in the task space.



(a) RRT with 20 nodes.



(b) RRT with 100 Nodes.



(c) RRT with 215 Nodes.

Figure 5.2: Plot showing the RRT (type=VERTEX) expanding into an open environment (no obstacles). The green node indicates the start.

The cost function $g(q)$ is defined to be the L2-norm traveled along each edge from the q_{start} to the q. With this cost function, our heuristic function is admissible, following the triangle inequality [13]. An admissible heuristic function always underestimates the cost from any configuration to the goal configuration. Unless the q_{start} is directly connected to q_{goal}, the path will consist of at least two edges. Let $q_0 = q_{start}$ and $q_2 = q_{goal}$. Let $g_{0,1}$ be the L2-norm (i.e. cost) from q_0 to q_1 and $g_{1,2}$ be the L2-norm (i.e. cost) from q_1 to q_2. While q_0 and q_2 are not connected, let $g_{0,2}$ be the the cost to travel from q_0 directly to q_2 (i.e. $h(q_0) = g_{0,2}$). If q_{start} is directly connected to q_{goal} then the heuristic value is equal to the cost, so the admissibility still holds. Therefore, the actual cost to reach the goal is $g_{0,1} + g_{1,2} \leq g_{0,2}$ by the triangle inequality. This will guarantee that our A* provides the optimum path from q_{start} to q_{goal} given the tree generated by the RRT. If the RRT was not successful, then an error is generated and A* is never run.

The output of the A* is the set of action vectors $U = \{\mu_1, \mu_2, \ldots \mu_n\}$ to move the robot from configuration to configuration. For the mobile base, each action vector μ has two rotations and a translation. The robot must first rotate to face the new configurations, translate in a straight line to reach the configuration's x and y position, then finally rotate to orient itself with the configuration's θ. The resulting movement of performing U serially results in superfluous rotations. When traveling from q_0 to q_1 then on to q_2, there is no reason for the robot to orient itself with θ_{q_1}, but rather simply orient itself to face $(x,y)_{q_2}$. The set U is passed to a smoothing function Eqn. (5.1) that combines the second rotation with the first rotation of the following action. The $min\_rot()\varphi)$ function is the same as that used in Eqn. (4.7).

$$smooth\left([\varphi_{i,1}\ \varphi_{i,2}]^T, [\varphi_{i+1,1}\ \varphi_{i+1,2}]^T\right) = \tag{5.1}$$

$$\{$$

$$\varphi_{i,2} = min\_rot\left(\varphi_{i,2} + \varphi_{i+1,1}\right)\ ;$$

$$\varphi_{i+1,1} = 0;$$

$$\}$$

5.1.3 Testing the RRT and A*

To collect data about the RRT and A* performance, we performed two sets of experiments and the results are shown in the tables Fig. 5.3. For each of these experiments, 20 random seeds were chosen. The RRT/A* algorithm was then run with each of these seeds and the results (number of nodes, time, path length, etc.) were averaged (μ) and the standard deviations calculated (σ).

Parameters	Goal Search		4%		Minimum Nodes		45									
	Number of Nodes		Number of Misses		Time of RRT [ms]		Time of A* [ms]		Path Length [nodes]		Path Translation [cm]		Path Rotation [rad]		Optimized Path Rotation [rad]	
Type	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
VERTEX	46.2	2.46	48.7	12.2	3.65	0.587	1.55	0.510	6.85	1.663	614.57	95.58	21.74	6.12	8.56	2.15
EXT_VERTEX	57.8	14.99	44.2	34.3	3.70	2.23	2.40	1.046	20.95	4.407	563.24	82.76	59.48	19.28	7.82	2.13
EDGE	51.2	9.37	113.4	78.1	138.8	129.3	1.75	0.550	16.65	3.281	564.32	85.70	44.83	14.78	8.35	2.42

(a)

Parameters	Goal Search		2%		Minimum Nodes		75									
	Number of Nodes		Number of Misses		Time of RRT [ms]		Time of A* [ms]		Path Length [nodes]		Path Translation [cm]		Path Rotation [rad]		Optimized Path Rotation [rad]	
Type	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
VERTEX	75.0	0	111.3	15.5	7.6	0.598	2.50	0.607	7.6	1.90	634.1	99.73	23.57	7.20	9.33	2.486
EXT_VERTEX	81.6	11.6	99.5	41.6	7.55	3.32	3.45	1.191	21.2	4.30	564.9	81.44	60.96	19.52	8.18	2.785
EDGE	79.3	9.8	353.6	202.3	684.2	533.83	2.50	0.688	20.1	3.02	568.9	84.31	54.34	15.95	8.95	2.433

(b)

Figure 5.3: (a) RRT Experiment 1 Results. (b) RRT Experiment 2 Results.

Within Experiment 1 and 2, the type of the tree (see above) was changed and the same 20 seeds were used again. Between Experiment 1 and 2, the **Parameters** shown in the first row of the tables were varied and the experiment run again.

From the data, we conclude that the EXT_VERTEX is the best type of tree to run. The time of the RRT execution, while almost doubling that of the VERTEX run-time, is orders of magnitude less than the EDGE time. The total distance traveled for the EXT_VERTEX is roughly equivalent to the EDGE type, which is an indication of a more "optimal" path. While the raw **Path Rotation** is greater than EDGE, the optimized path rotation is less in both experiments. Another benefit of the EXT_VERTEX is that the edges of the path are, on average, shorter. This can be inferred from the **Path Length** as measured in number of nodes. Since small errors are magnified over larger distances, the robot has a better chance to maintain course. As the data shows, EXT_VERTEX performs the best for the scenario we have used, and is the method used in all of the simulations in the following sections.

5.1.4 Simulation of Mobile Base RRT

To evaluate the RRT approach, a scenario was designed in which the mobile base would move from one side of an environment to the other while avoiding obstacles. Fig.5.4 shows the scenario. In the image, the red areas represent the obstacles in the task space. The thick black border represents the extreme limits of the x-axis and y-axis for this work space (i.e. the walls of the room). The dashed lines represent the boundaries of the configuration space obstacles.

While the robot is estimated as a disc, meaning the configuration obstacles formally should have rounded borders from the Minkowski differences, the obstacles are simply estimated as the bounding rectangle that encloses the Minkowski difference. This is done

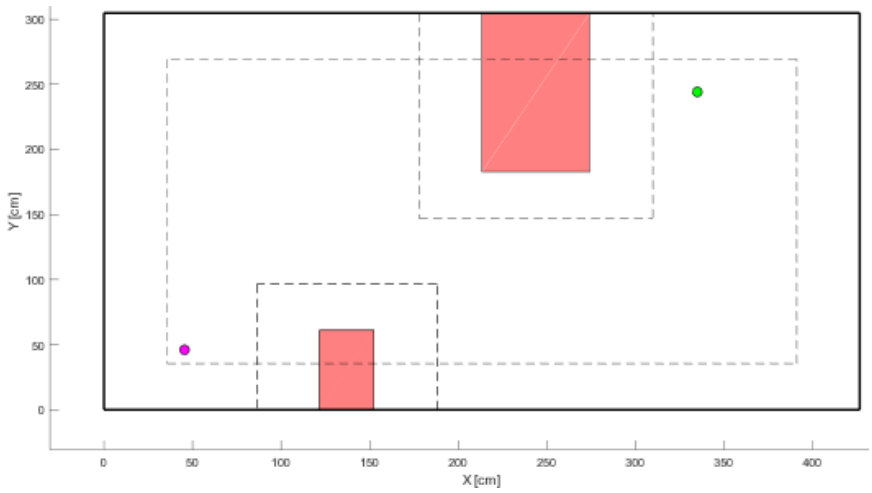


Figure 5.4: The configuration space of the mobile base in the scenario presented above

for simplicity in the collision detection. In the scenario, the start position is shown as the green node and the goal is the pink node.

Preliminarily to implementing the path execution on the robot, animations were created in MATLAB to visualize the paths as they are created. Fig. 5.5 shows snapshots of the trees as they are animated, as well as the final RRT with the path generated by the A* algorithm highlighted.

After the central computer runs the RRT and A* algorithms, it generates *.csv files of the graph and path. These files are parsed by a MATLAB script to generate the animations. A different script creates a robot trajectory based on the actions and simulates the robot as it would follow the path through the environment, as seen in Fig.5.6.

5.2 Path Planning for the Continuum Section

5.2.1 RRT and A*

Using the configuration space for CuRLE that we established in Chapter 4, we apply the same classical motion planning techniques to map and navigate the continuum

element c-space as we did for the mobile base c-space.

The same RRT described in section 5.1.1 was used to generate a path for the continuum element. The C++ code was modified so that the configurations in each vertex were continuum configurations ($c = [u\ v\ \omega]^T \in C_{space}$) and every edge is the "action" vector μ needed to move from one configuration to the next. Actions in this space are simply the difference in each configuration variable. The start and goal location, along with the location of all configuration space obstacles, were known *a priori* and all of the obstacles remain convex polyhedrons. Following the evaluation described in section 5.1.3, the RRT type EXT_VERTEX was chosen for CuRLE . All the other RRT configuration parameters (percent of iterations to connect goal, minimum nodes, maximum nodes, etc.) were set to be the same as the mobile base RRT.

Given the unique kinematic constraints of continuum robots identified earlier, the RRT algorithm had to be modified to be applicable to CuRLE . In order to pick up an object, for instance a cup on a shelf, the continuum section has to bend in such a way so that the object ends between the fingers on the gripper. Since the gripper has a fixed maximum width, the room for error is very small. In the configuration space, this means that the goal location is always in a narrow "canyon" created by the configuration space obstacles. For RRT implementations, this can make it very difficult to connect to the goal. We solved this issue by "projecting" the goal node along a straight line until it was out of the "canyon". Once this projected goal could be attached to the graph, the goal was then achieved moving u or v in a straight line.

We solved collision detection and avoidance by taking a sampling based approach. Since the kinematics of the continuum element are complex, manifestation of task space obstacles in the configuration space of a continuum element is complex. Rather than formally defining a mapping between the task space and the configuration space for a generic obstacle, we sampled the task space with our simulated version of CuRLE and recorded all

the configurations of any collisions, which are detected by sampling along the backbone of the continuum element and checking for a collision at each point. Fig. 5.8 shows the configuration space manifestation of the task space obstacles shown in Fig. 5.7. This offline sampling method generates a discrete lookup-table for every possible configuration, sampled at a rate of 0.01 between the minimum and maximum values, for each configuration variable.

Once the RRT algorithm connected the goal configuration to the graph, we ran the same A* algorithm as for the base to determine the optimal path. As with the mobile base, the A* used a heuristic of the L2-norm between a given configuration and the goal configuration. The cost function was the L2-norm between each node in the current path from the start node. We also implemented a weighting function to encourage movements of the continuum arm (u and v) and penalize movements of the revolute joint at the base of CuRLE (ω). This was done to reduce the overall execution time in the physical hardware. The algorithm then output the full graph and optimal path to a *.csv file.

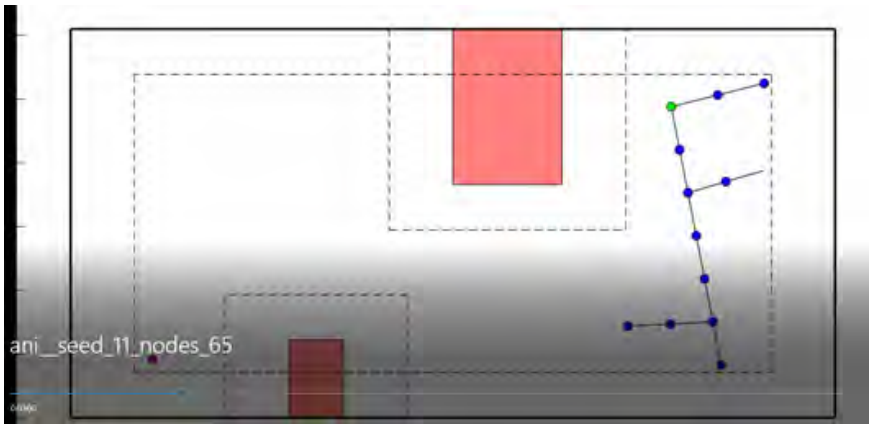
5.2.2 Simulation Results

To verify the motion planning algorithms, a simulated environment was created in MATLAB and a model of CuRLE was developed and added to the simulation. The simulation discussed herein involves a scenario where CuRLE is instructed to pick up a cup off a shelf by the user (see Fig. 5.7). The task space obstacles were converted to configuration space obstacles, shown in Fig. 5.8. The start configuration of CuRLE was $c_{start} = [0\ 0\ 0]^T$ and the goal configuration for the example reported herein was $c_{goal} = [-1.76\ 0\ -\pi/2]^T$, (i.e. the configuration needed to pick up the cup). This goal was determined by using the interactive GUI developed with the environment to bend the simulated CuRLE until it was in the correct configuration. Fig. 5.9 shows the interactive GUI with CuRLE in the start

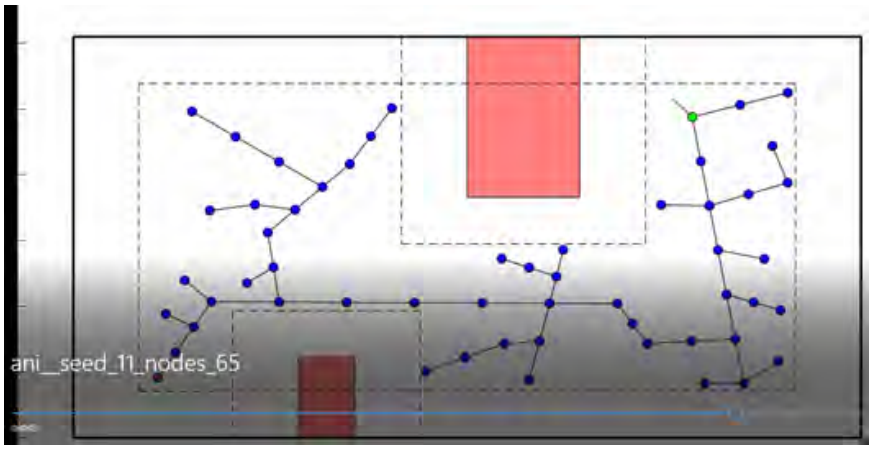
configuration.

The output of the RRT/A* was fed into the simulation and Fig. 5.10 shows CuRLE achieving the goal configuration of grasping the "cup". For this simulation, we set $v = 0$ to keep the configuration space 2D and allow for easy visualization of the nodes from the RRT algorithm.

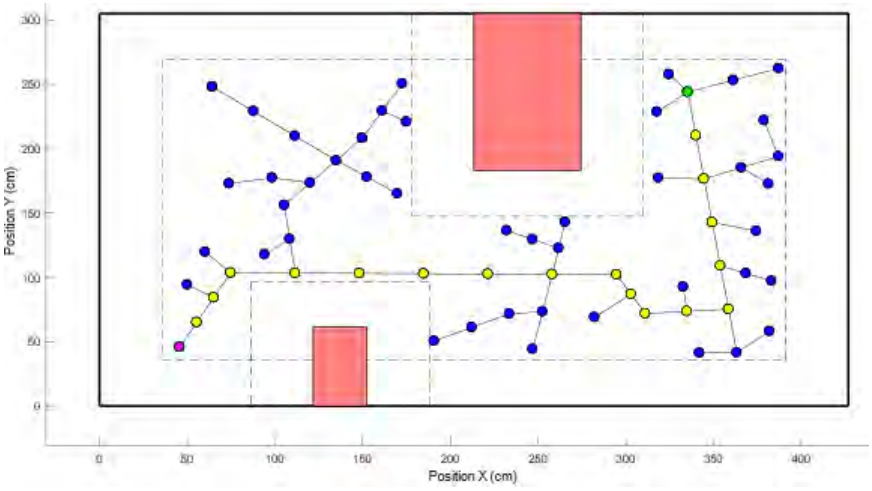
From the simulation that we ran for both the mobile base RRT/A*, we can predict that our motion planning algorithms will successfully navigate the configuration space to guide the robot through the task space. In the proposed scenario, a path for the mobile base is found rapidly and the simulation shows that the path is collision-free. In addition, we can conclude that our work to design the RRT to be re-configurable and extensible was a success. By simply changing the "state" definition, we generated a collision-free path for the continuum element through its task space by using its configuration space, which we verified by our simulation of CuRLE.



(a)

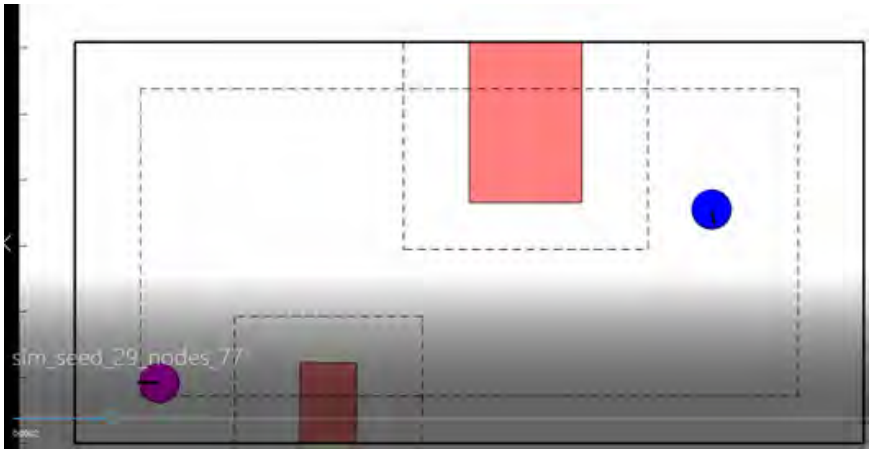


(b)

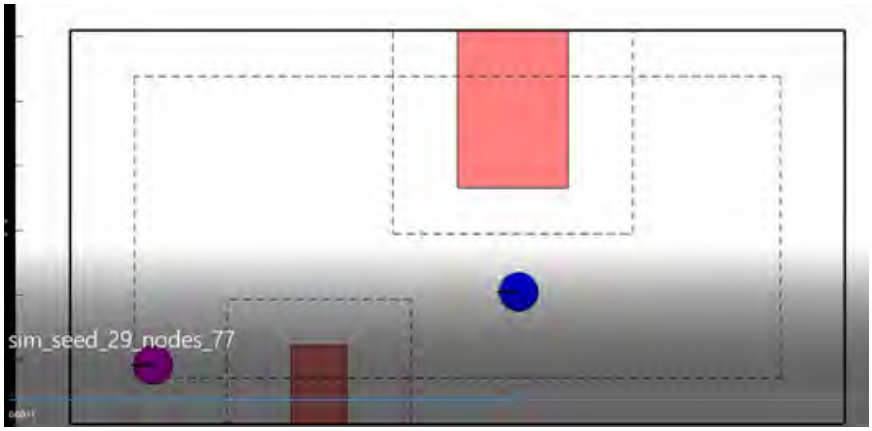


(c)

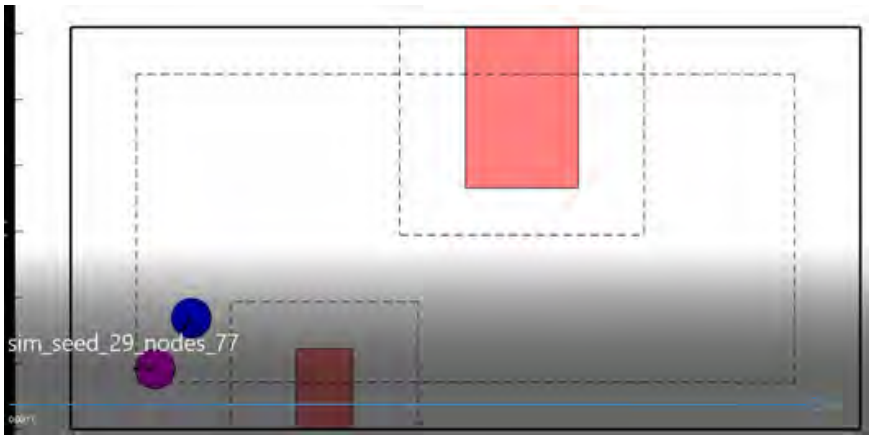
Figure 5.5: The RRT growth over time.



(a)



(b)



(c)

Figure 5.6: A progression of images showing the simulated robot moving through the scenario.

Figure 5.7: The task space showing the cup from the scenario. The middle cup is the ”goal” cup that CuRLE will pick up and deliver to the user.

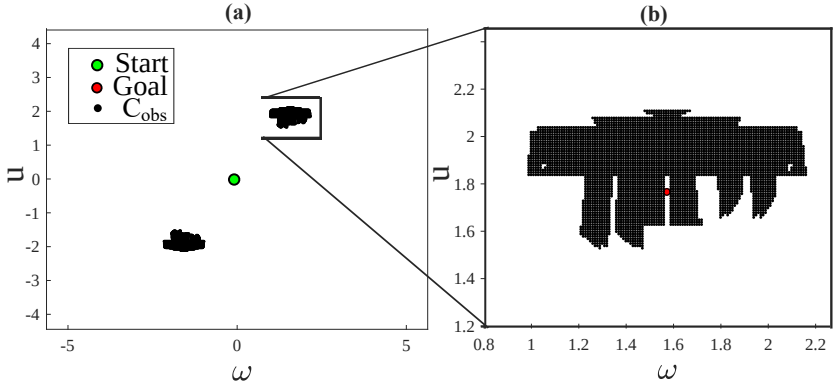


Figure 5.8: The configuration space obstacles of the task space. The start configuration, is shown in (a), while (b) is the magnified obstacle from (a) where the goal configuration is located. (b) is the c-space obstacle of the shelf shown in Fig. 5.7

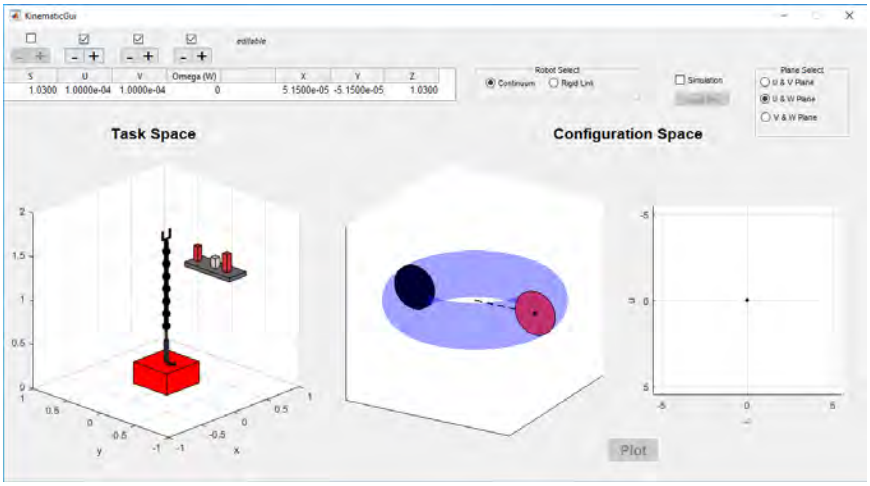


Figure 5.9: The interactive GUI that serves as the front end for simulation environment.

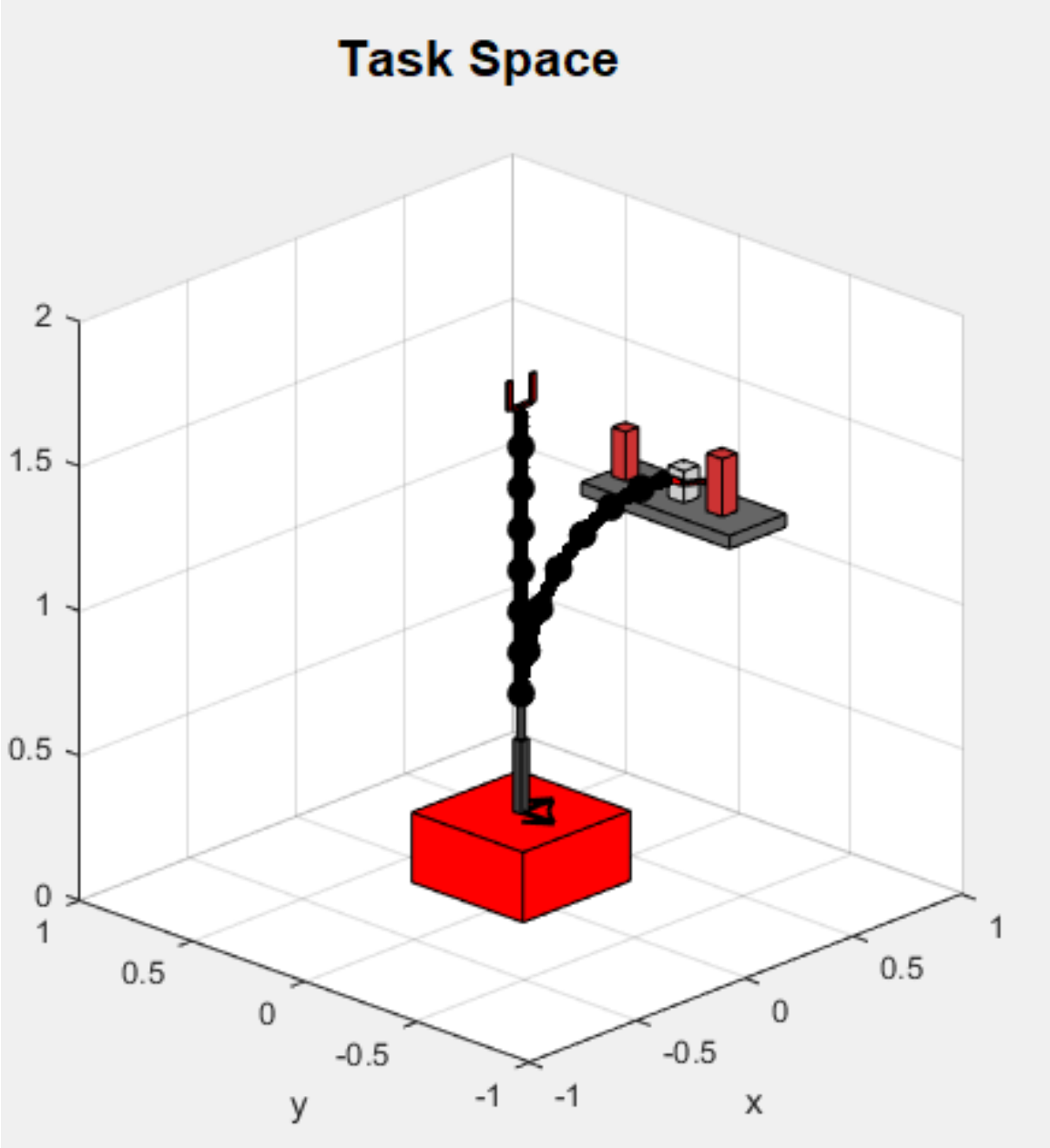


Figure 5.10: The simulated CuRLE in the start (vertical) and goal (bent) configuration. The objective of the scenario was to pick up the cup, shown as a light grey prism, on the shelf.

Chapter 6

Validation of Motion Planning with CuRLE Robot Hardware

This Chapter details experimentation done to validate the RRT/A* algorithms developed in Chapter 5 by implementing them on CuRLE. First we describe the software running on CuRLE to implement the autonomous motion given a path through the configuration spaces. Next we describe an initial experiment to validate the RRT output for the continuum element on the CuRLE hardware. We then lay out the over-arching in-home scenario that we used to fully validate the work in this thesis regarding the configuration space and motion planning for a practical hybrid tendon-actuated continuum element/mobile base operating in free-space. The final sections detail the experiments done and evaluate the results.

6.1 CuRLE Software Implementation

The software running on the CuRLE hardware is split between the Arduino Due micro-controller and the Raspberry Pi computer. The Arduino is responsible for interfacing

with all of the actuators (motors) and sensors (encoders and tension sensors) in the system. Implemented in the Arduino IDE (C++), the software controllers running on the Arduino execute the transitions from the current configuration to the next. The Raspberry Pi serves as the central computer of CuRLE and is responsible for interfacing with the Arduino. Primarily running Python scripts, it passes external messages to the Arduino and relays any response. The Raspberry Pi also stores the current state of the robot, for persistence of memory when power is lost. Finally, since the Raspberry Pi can be accessed remotely over the network, we can take direct control of the Pi (and by extension the Arduino) to update code, execute individual commands, and initiate a debugging session.

6.1.1 Configuration Controller

Much like the controller of the h+lamp (see Chapter 2), the controller(s) running on the Arduino constantly monitor the current configuration of the robot (i.e. reading the sensors) and maintain a desired configuration by actuating the motors when the current configuration strays from the set point (i.e. the error is nonzero). Two PID controllers generate signals for the motors based on the current error values: one for the mobile base state and one for the continuum element state.

The controller for the mobile base is fundamentally the same as the one discussed for the h+lamp. The method for calculating error signals and set points for each drive motor is as detailed in Chapter 2. The gain values for the PID were tuned differently to work for the new hardware, but the underlying software is the same.

Because of the unique properties of the continuum element kinematics, the controller for the continuum element fundamentally operates differently from the mobile base controller. As mentioned in Chapter 3 knowing the length of the tendons is essential for determining the configuration of the robot. In order to accurately measure the length, the

radius of the spool must be known. As such, the controller must also track the radius of the spool as it changes over time. Because of this, the absolute position of the motor shaft must always be known and saved.

After initial calibration when CuRLE was first constructed, the current state of the motor shaft (measured in encoder counts) is saved to a configuration file on the Raspberry Pi. In addition, the continuum configuration is stored ($[u\ v\ \omega]$). Every minute, the Arduino communicates the state (motor encoder count and the $[u\ v\ \omega]$ corresponding to those counts) to Pi and the Pi updates the configuration file. When the system first boots, the Arduino requests the saved state from the Pi and remains idle until the Pi sends the state. The controller is able to use the "zero" state and the current encoder readings to track the radius of the spools and thereby the lengths of the tendons.

To reach a desired configurations, the same PID controller operates independently on all four tendons. This controller was tuned until the robot could accurately reach a desired configuration from its current configuration.

6.1.2 Execution Sequence

To implement the RRT output on CuRLE, the motion planning was done offline on an external computer. We then transmitted the output files to the Raspberry Pi over the local network. Rather than have the external computer execute this process automatically, we found that experimentation was easier if we manually transferred the files. Once the files were on the Raspberry Pi, we remotely accessed it via SSH and ran an interactive (via command line) Python program that initiated the Arduino. Once the Arduino received the saved state and finished its initializing, we executed a command through the Python program to load the paths (mobile and continuum element) and transmit commands to the Arduino. Once the Arduino reached the set configuration (i.e. all error signals dropped

below a minimum threshold), it sent a message back to the Raspberry Pi to indicate that it has completed its task. In addition, the Arduino flashed the lamp LEDs to visually communicate that the set configuration had been reached. The Raspberry Pi, idle until it received this message from the Arduino, then sent the next configuration in the path. This process repeated until the goal was reached.

6.2 Validating the Continuum Section Controller

As we did with the h+lamp controller, we first tested the continuum controller's ability to "follow" an RRT path by serially executing the transition from node to node. Recall from Chapter 5 the simulation experiment done to test the RRT output for the continuum element. In simulation, we demonstrated the software model of CuRLE bending to grasp a cup on shelf. We took the same RRT output (shown in Fig. 6.10) and passed this path to CuRLE. We issued "grasp" command to grab the cup and then passed CuRLE a second path from the RRT to lift the cup from the shelf. The results of this experiment are shown in Fig. 6.1.

6.3 In-Home Scenario

To demonstrate the full functionality of the system, we explored a simple in-home scenario where the user has asked CuRLE to fetch a cup from shelf, bring the cup into an adjacent room and set it down on a different shelf, then navigate to its "docking station" in another portion of the smaller room. The environment for this scenario is shown in Fig. 6.2. This scenario can be further broken down into three sub-scenarios, each involving planning for the mobile base and the continuum element. In each sub-scenario, the path(s) the robot must follow is shown for both the mobile base and the continuum element. We

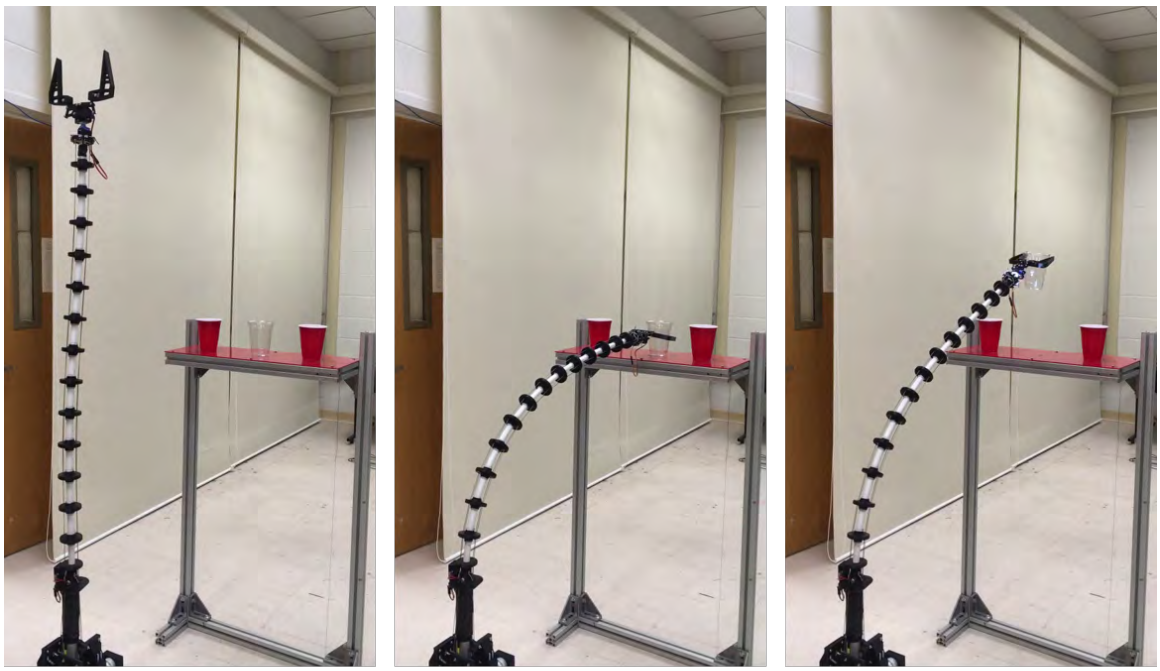


Figure 6.1: (a) The state of CuRLE after ω has aligned with the goal ω. (b) CuRLE has grasped the cup. (c) The results of a second path generated by the RRT (shown in Fig. 6.11) that guided CuRLE to pick the cup off the shelf.

then show and discuss the results from each experiment.

6.4 Sub-Scenario 1: Simple Base Movement and Grasping the Cup with Continuum Element

In this sub-scenario, the first set of actions moves CuRLE from the start (Location 1) to the goal (Location 2) to position the robot in front of the shelf. Next, the continuum element bends to grasp the cup. A ”grasp” command is then issued and CuRLE lifts the cup from the shelf. Fig. 6.3 shows the path generated by the mobile base RRT, and Fig. 6.4 and 6.5 show the two paths generated by the continuum element RRT.

Figure 6.2: The in-home scenario explored to demonstrate the full functionality of CuRLE. The Locations discussed below are numbered in the image. The first shelf is located at Location 2 and the second shelf is at Location 3.

6.4.1 Results and Discussion

Images from video footage of the experiment are shown in Fig. 6.6, Fig. 6.7, and Fig. 6.8a. The mobile base reached its goal with minimal error, demonstrating the vast improvement of CuRLE's mobile base performance over that of its predecessor (h+lamp). The effort reported in Chapter 3 to upgrade the robot is thus validated in this situation.

Since the mobile base had minimal error in its position, the continuum element successfully picked up the cup from the shelf. Since there is no external sensor providing feedback to CuRLE, small errors in the position will cause failure when executing the RRT output. For instance, if the base is off by even a small distance, the continuum element will

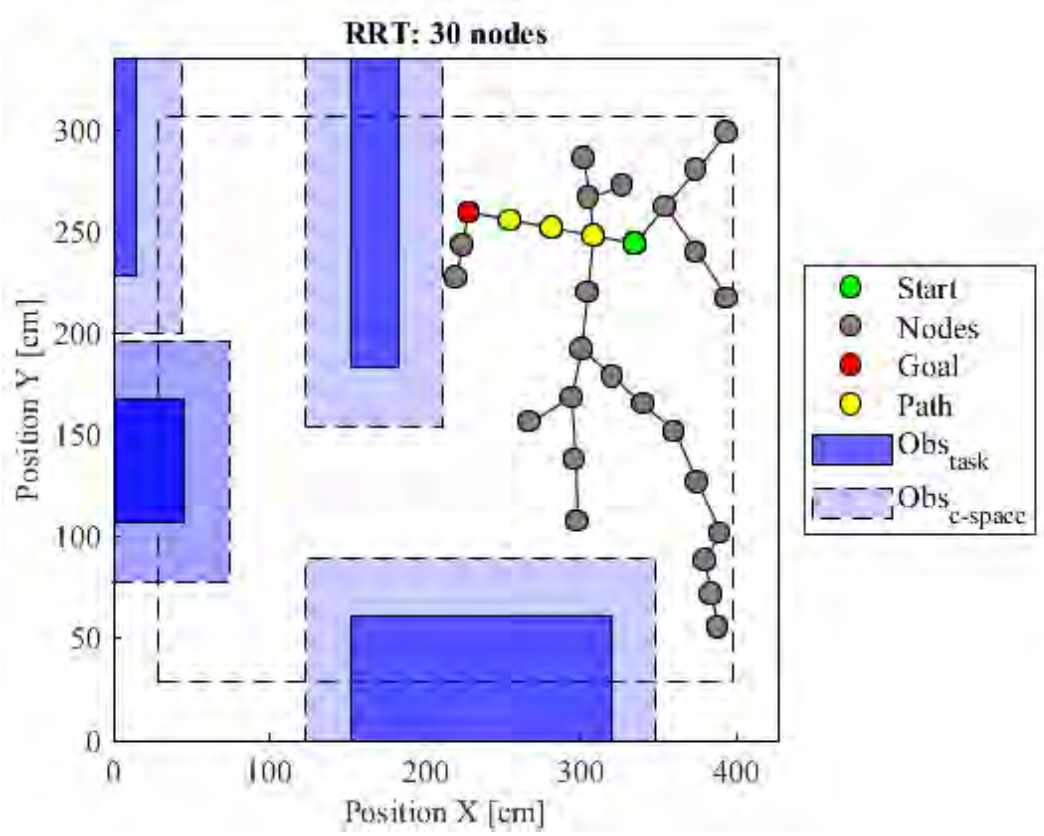
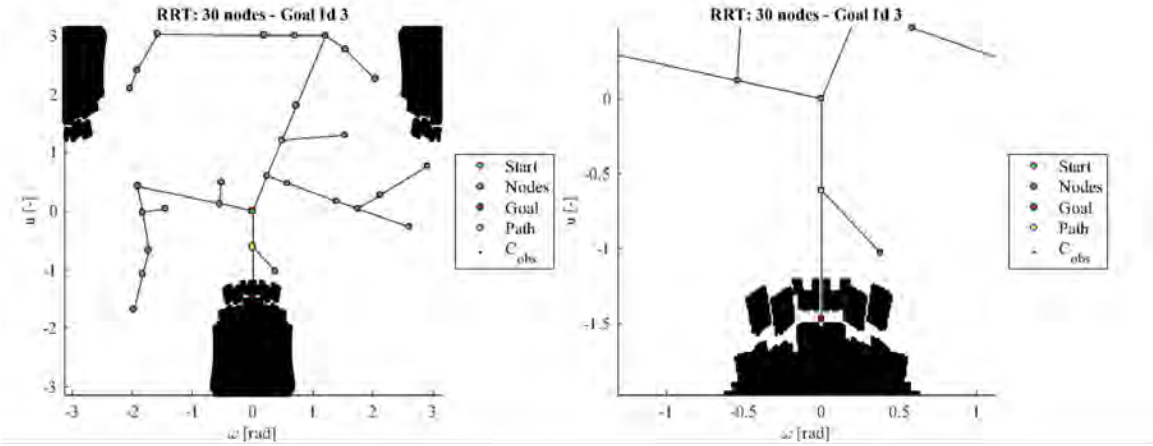
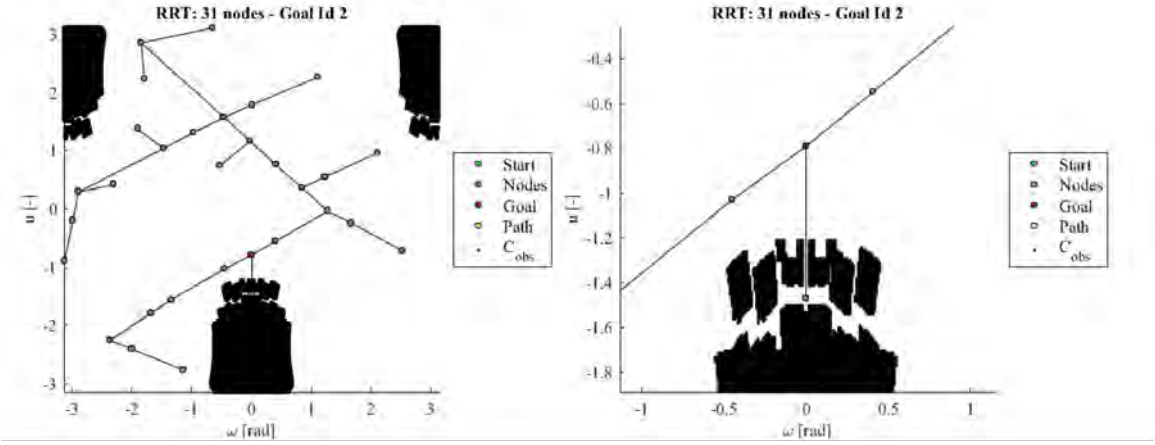


Figure 6.3: Output of the RRT showing the path required for the mobile base of CuRLE to navigate from the start location to the first shelf.



(a) (b)

Figure 6.4: Output of the RRT showing the path required for CuRLE to grasp the cup on the first shelf. (b) is a magnified portion (a) to better show the start and goal configurations.



(a) (b)

Figure 6.5: Output of the RRT showing the path required for CuRLE to pick the cup up off the first shelf. (b) is a magnified portion (a) to better show the start and goal configurations.

miss its goal location. This was seen in initial experimentation and resulted from poor PID control. By tuning the gains, we reduced the error enough such that the continuum element was able to grab the cup.

6.5 Sub-Scenario 2: Complex Base Movement and Placing the Cup with Continuum Element

In this sub-scenario, the first set of actions moves CuRLE from the first shelf (Location 2) to the goal (Location 3) to position the robot in front of the shelf. To do this, CuRLE must travel to the ”second” room, which is on the far side of a ”wall” indicated by the long vertical obstacle in the center of the task space. Once in the other room, the continuum element bends to place the cup on the second shelf there. A ”release” command is then issued and CuRLE bends away from the shelf. Fig. 6.9 shows the path generated by the mobile base RRT, and Fig. 6.10 and 6.11 show the two paths generated by the continuum element RRT.

6.5.1 Results and Discussion

Images from video footage of the experiment are shown in Fig. 6.8b, Fig. 6.12, and Fig. 6.13. As with Sub-Scenario 1, the mobile base successfully followed the path and reached its goal location with minimal error. The continuum element successfully held the cup during the entire transportation from the ”start” to the ”goal”. Once again, since the mobile base arrived with small enough error, the continuum element was able to place the cup on the shelf.

6.6 Sub-Scenario 3: Simultaneous Movement Between Base and Continuum Element

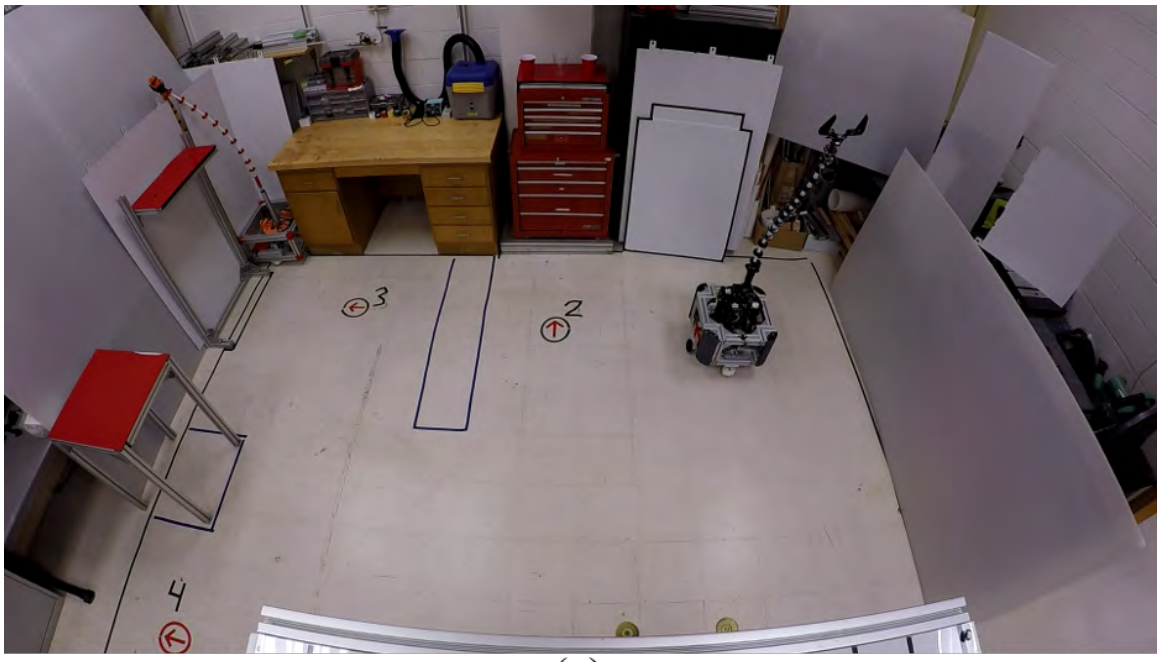
In this sub-scenario, the first set of actions moves CuRLE from the second shelf (Location 3) to the goal (Location 4) to position the robot at its ”docking station”. To do this, CuRLE must navigate the tight space in the ”second” room. While moving through this room, the continuum element bends to return to its ”home” configuration. This experiment demonstrates the ability for the robot to execute paths from both RRTs in parallel. Fig. 6.14 shows the path generated by the mobile base RRT, and Fig. 6.15 shows the path generated by the continuum element RRT.

6.6.1 Results and Discussion

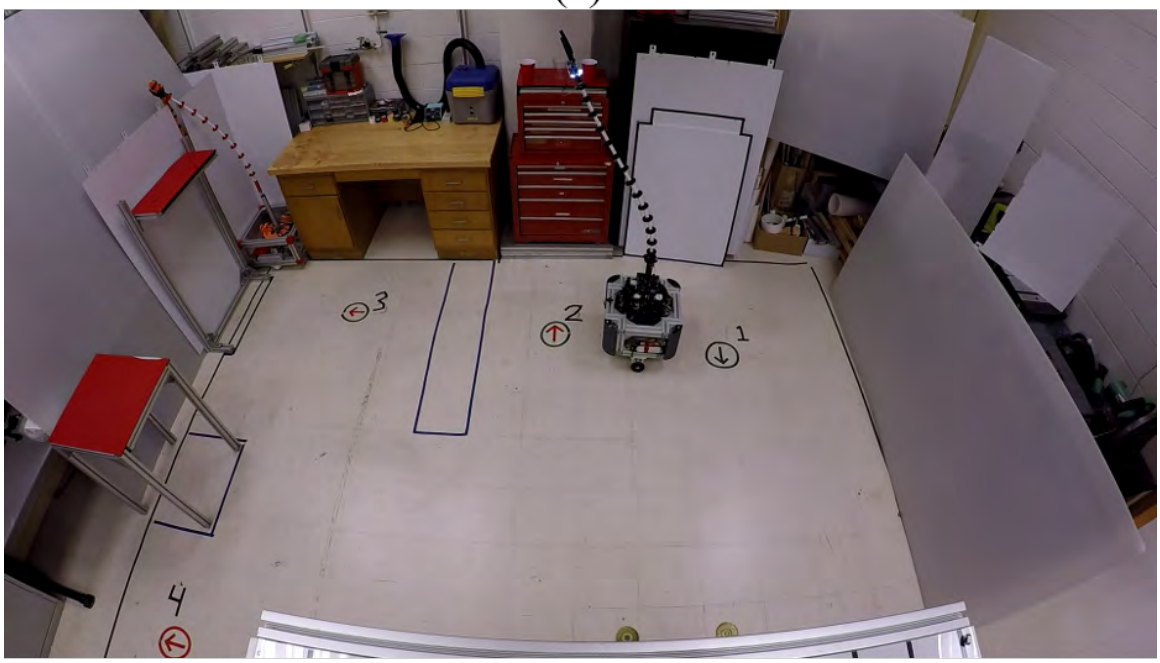
Images from video footage of the experiment are shown in Fig. 6.16 and Fig. 6.17. The final position of the robot is shown in Fig. 6.18. As with the previous two sub-scenarios, the mobile base successfully navigated from the ”start” configuration to the ”goal”. During this movement, the continuum element also executed its path from the continuum RRT. We successfully showed the ability of the robot hardware to execute both

RRTs in parallel, which was something our simulations had not been able to show.

These experiments show that our motion planning algorithms for both the mobile base and the continuum element were successfully implemented on the CuRLE robot hardware. As with CuRLE's predecessor, h+lamp, we saw the robot begin execution of the RRT output. Unlike h+lamp, CuRLE had the appropriate hardware and well-tuned controllers that enabled it to successfully complete the tasks. We serially executed multiple RRTs for the mobile base and continuum element, and we successfully executed the two RRTs in parallel. Any error generated from the PID control was small enough that the robot still retrieved and placed the cup on both shelves respectively.

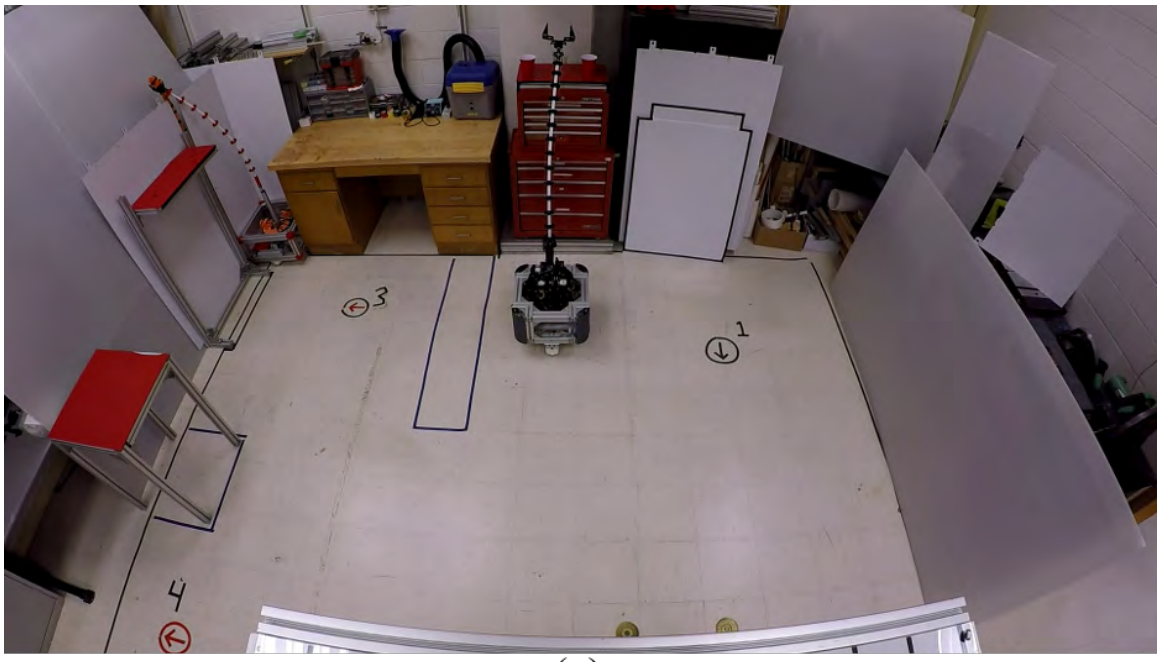


(a)

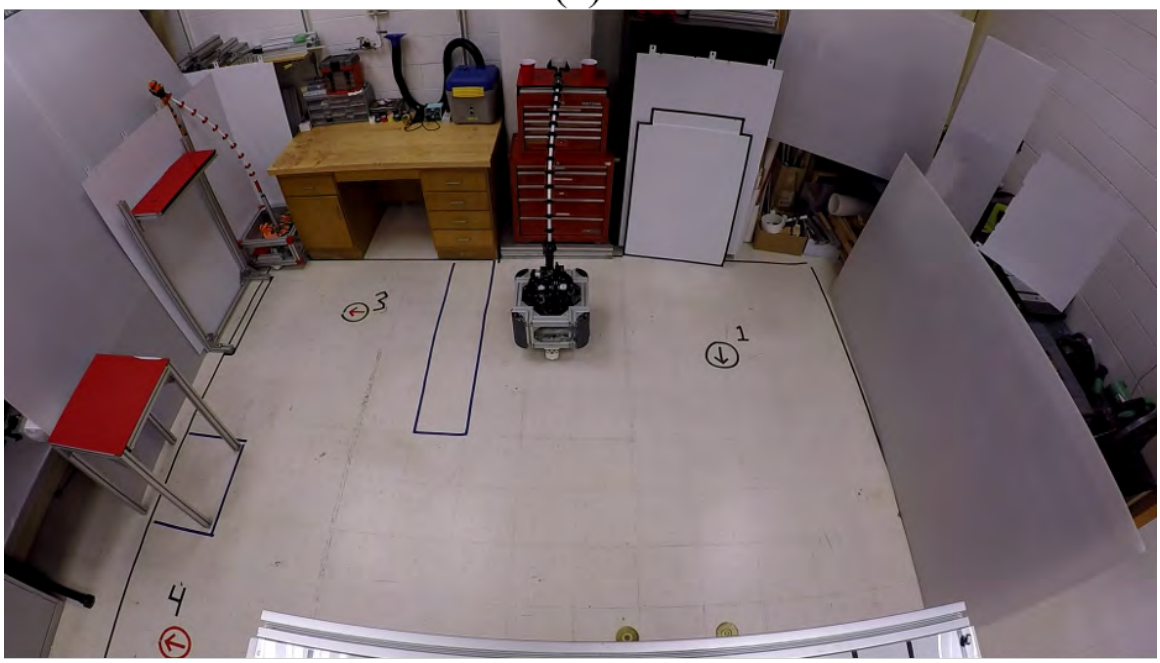


(b)

Figure 6.6: (a-b) Results from Sub-Scenario 1 showing the execution of the RRT from Fig. 6.3 for the mobile base (i.e. CuRLE move from Location 1 to Location 2).

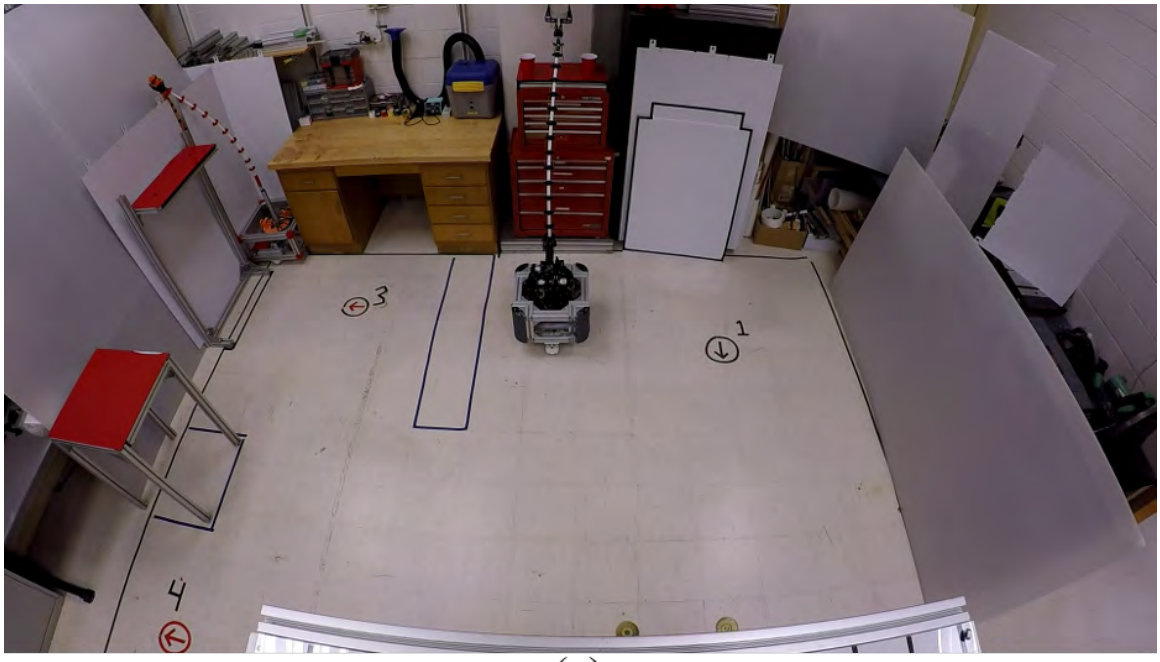


(a)

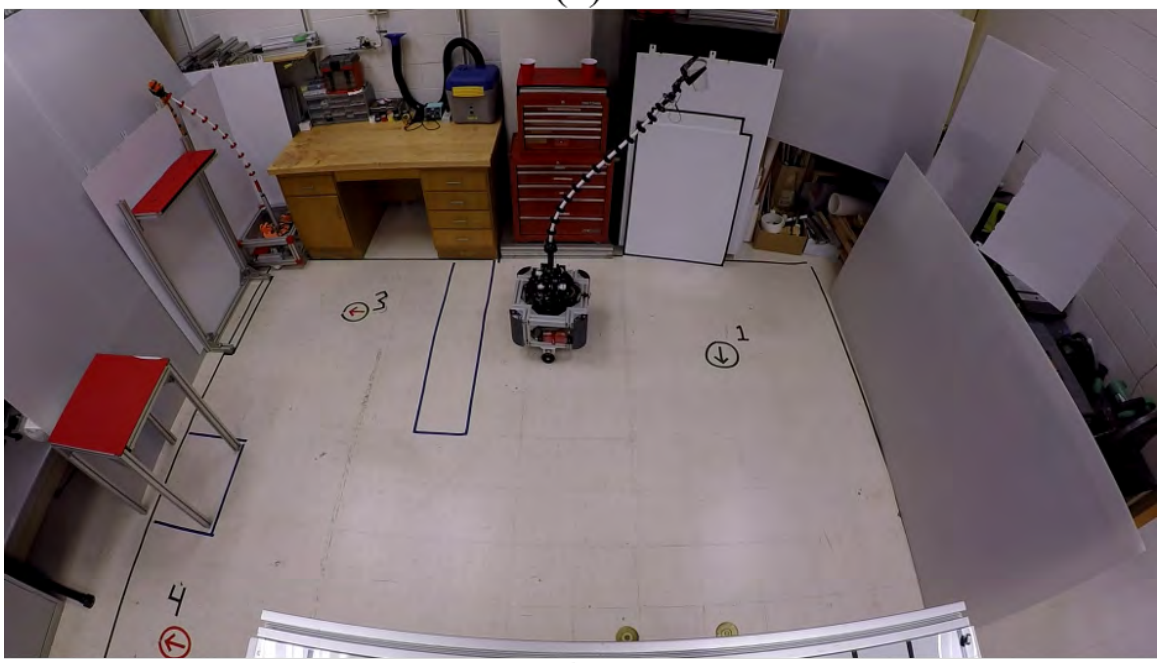


(b)

Figure 6.7: (a-b) Results from Sub-Scenario 1 showing the execution of the RRT from Fig. 6.4 (i.e. CuRLE grasp cup).



(a)



(b)

Figure 6.8: (a) Results from Sub-Scenario 1 showing the execution of the RRT from Fig. 6.5 (i.e. CuRLE lift cup off shelf). (b) Results from Sub-Scenario 2 showing the execution of the RRT from Fig. 6.9.

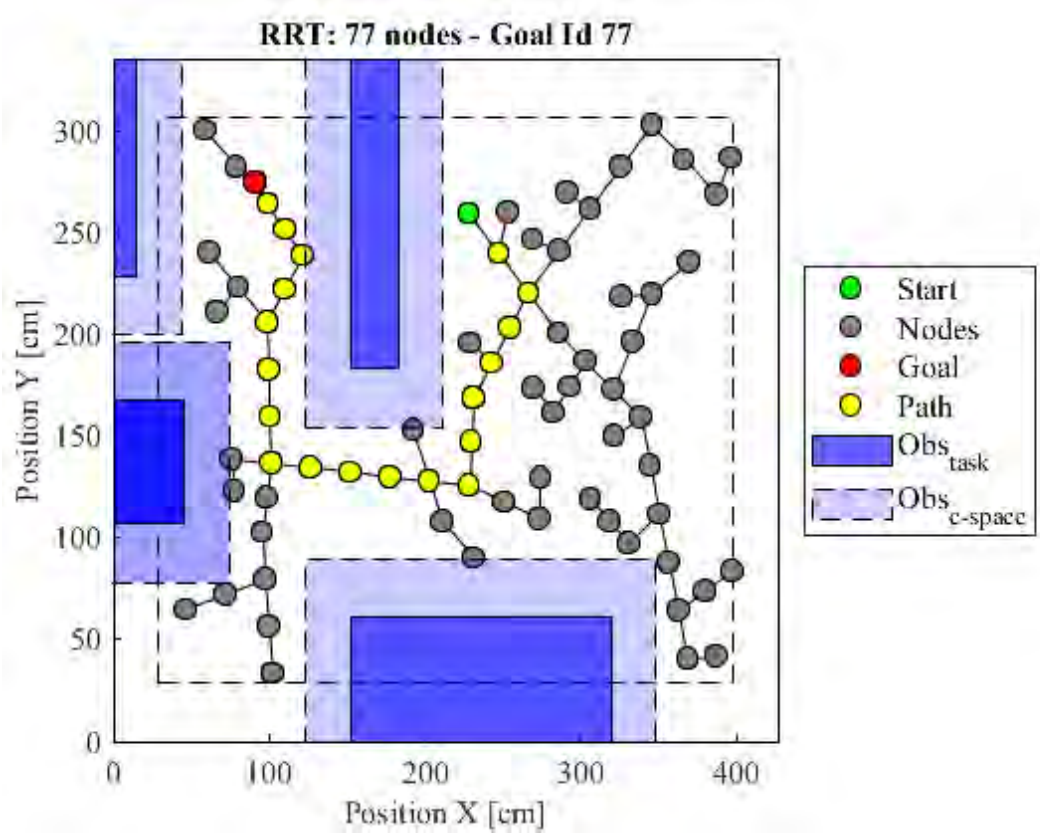
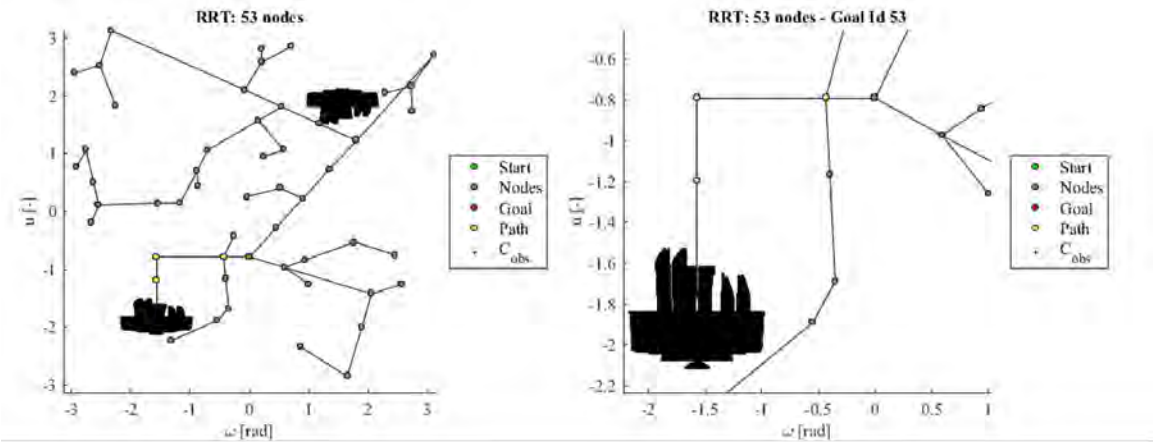
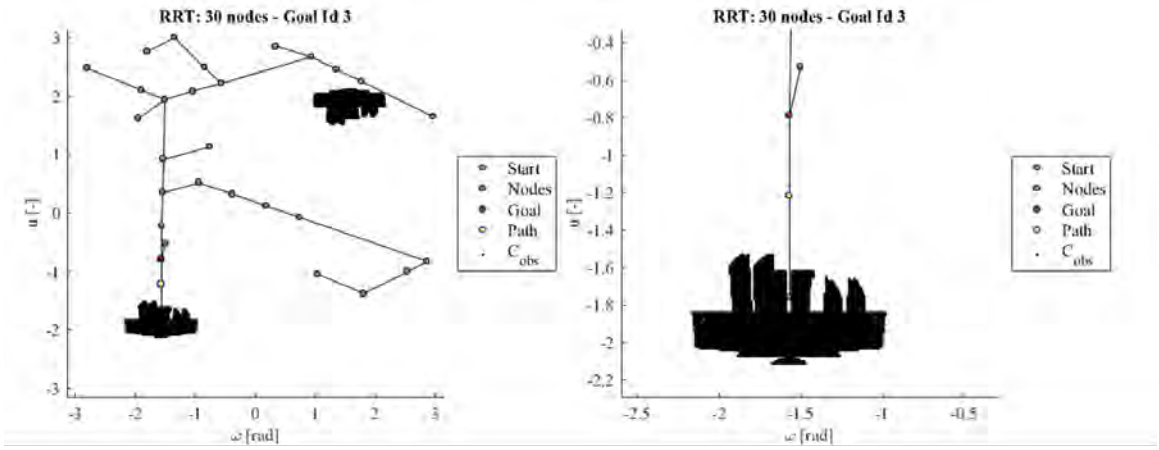


Figure 6.9: Output of the RRT showing the path required for the mobile base of CuRLE to navigate from the first shelf to the second shelf.



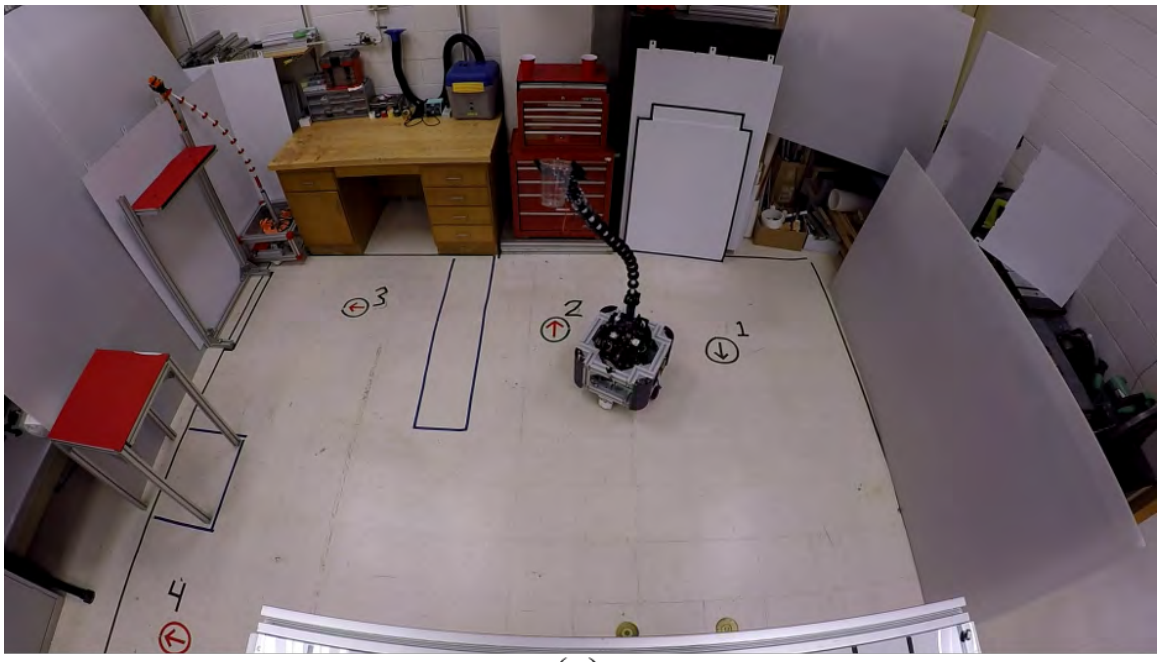
(a) (b)

Figure 6.10: Output of the RRT showing the path required for CuRLE to place the cup on the second shelf.(b) is a magnified portion (a) to better show the start and goal configurations.

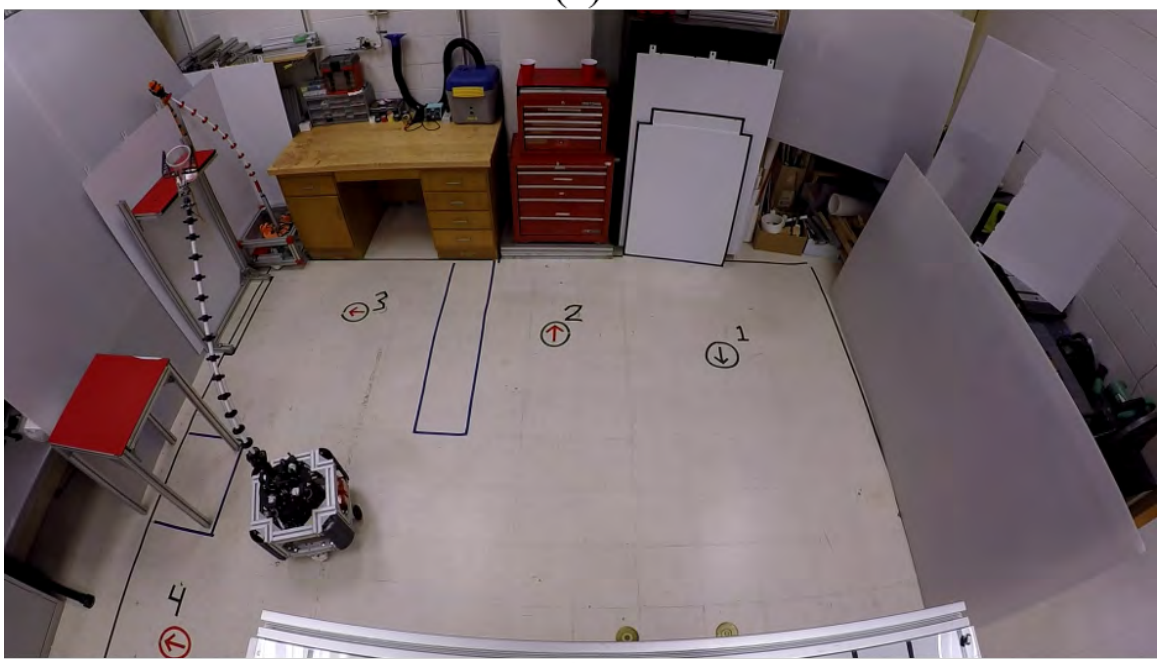


(a) (b)

Figure 6.11: Output of the RRT showing the path required for CuRLE to release the cup on the shelf and move away.(b) is a magnified portion (a) to better show the start and goal configurations.

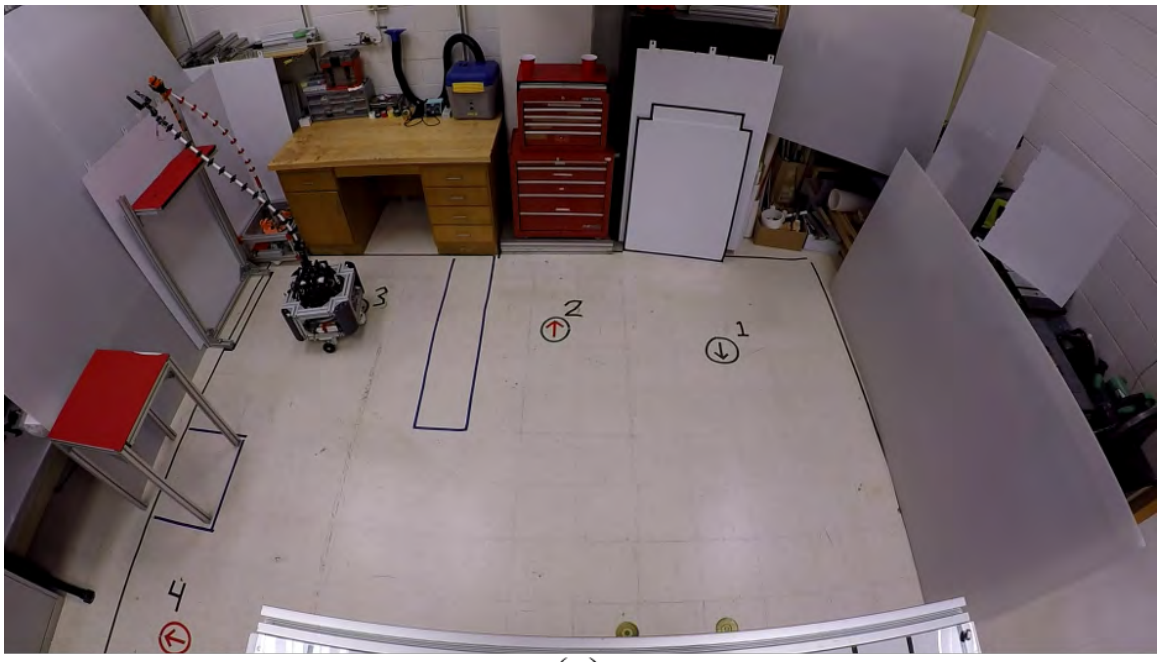


(a)



(b)

Figure 6.12: (a-b) Continued results from Sub-Scenario 2 showing the execution of the RRT from Fig. 6.9 for the mobile base (i.e. CuRLE move from Location 2 to Location 3).



(a)

(b)

Figure 6.13: (a-b) Results from Sub-Scenario 2 showing the execution of the RRT from Fig. 6.10 (i.e. CuRLE release the cup).

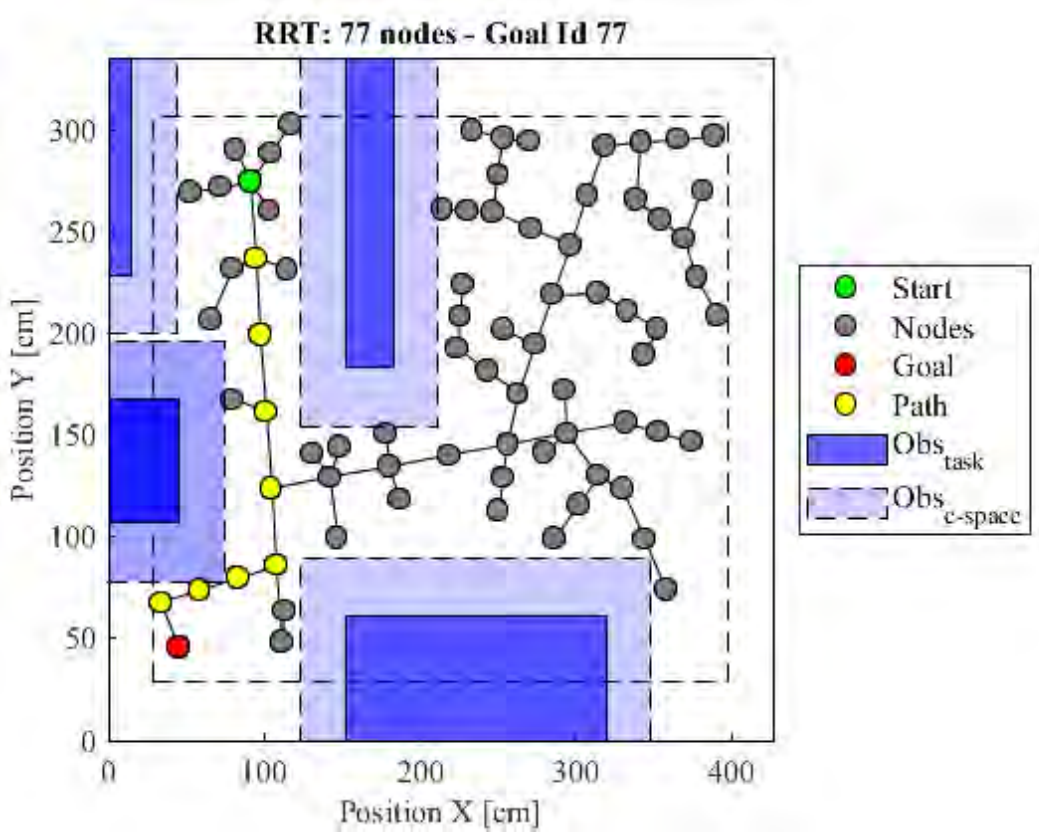


Figure 6.14: Output of the RRT showing the path required for the mobile base of CuRLE to navigate from the second shelf to the ”docking station”.

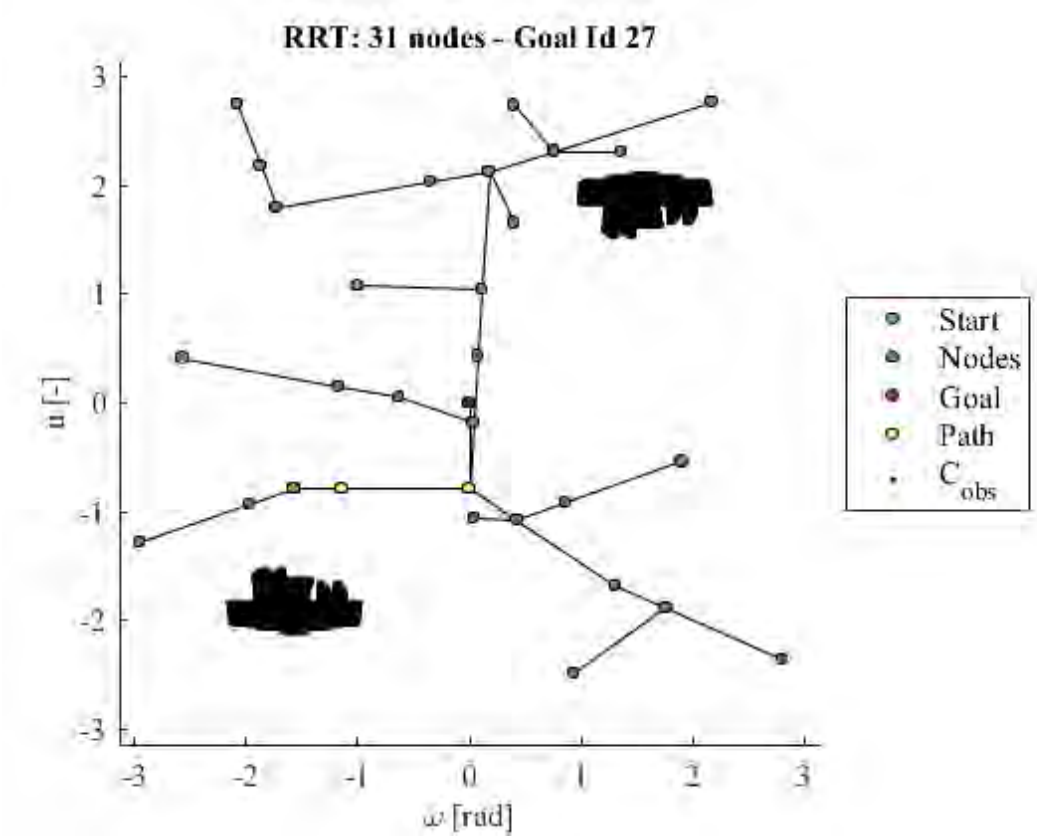


Figure 6.15: Output of the RRT showing the path required for CuRLE to return to its "home" state ($[u = 0, v = 0, w = 0]$)

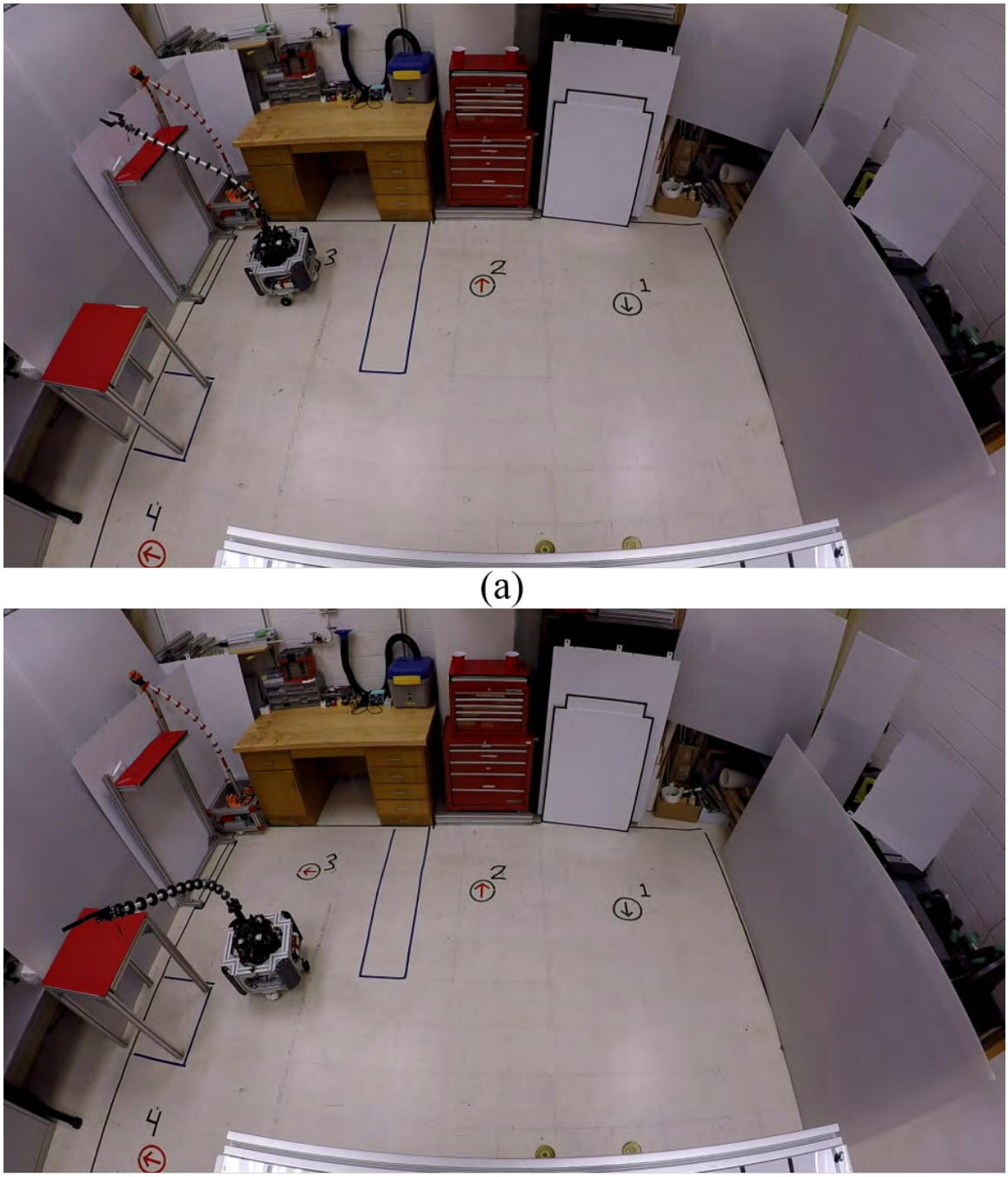
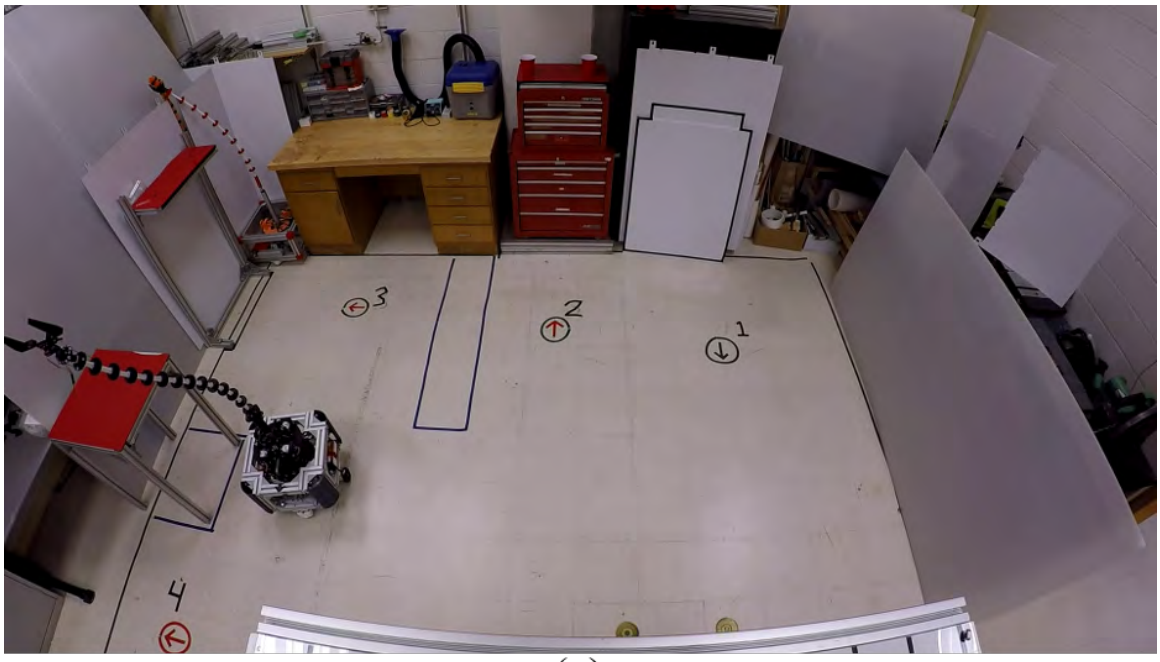


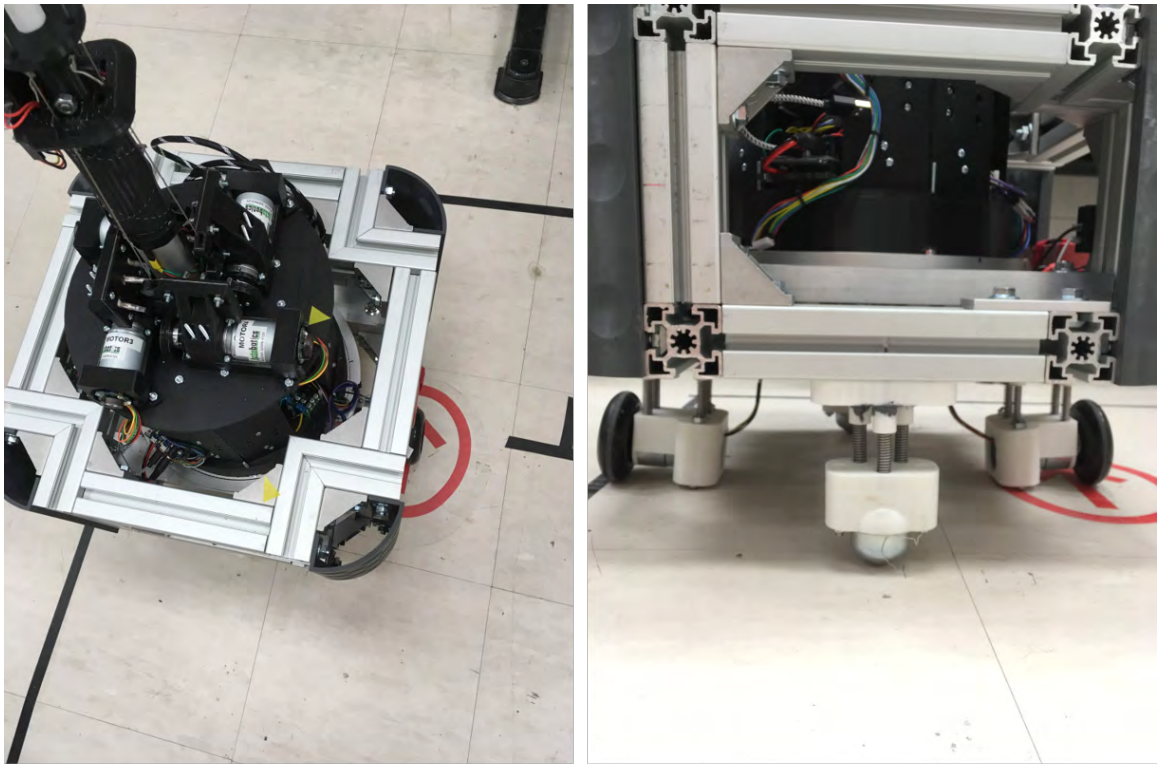
Figure 6.16: (a-b) Results from Sub-Scenario 3 showing the execution of the RRT from Fig. 6.14 for the mobile base executing in parallel with the output of the RRT from Fig. 6.15 (i.e. CuRLE return to ”home” state while moving from Location 3 to Location 4).



(a)

(b)

Figure 6.17: (a-b) Continued results from Sub-Scenario 3 demonstrating the parallel execution of the two RRTs.



(a) (b)

Figure 6.18: Final results of entire experimentation showing the small amount of error in the final position of the mobile base.

Chapter 7

Conclusions and Suggestions for Future Research

7.1 Conclusions

In Chapter 2, we detailed our first attempt at realizing our continuum robotic mobile lamp as an autonomous system. We recounted the wider history of the *home+* effort and described the motivation for the work later conducted and described in Chapters 4 & 5. In evaluating the level of control the user should have when interacting with the *home+* suite of robots, a spectrum arises with tele-operation at one end and full autonomy at the other. We described work done to implement tele-operation (i.e. the user is in full control of the robot) via different hardware controllers. To move towards full autonomy in the robot, we implemented the RRT/A* algorithms, detailed in Chapter 5, for the mobile base of the h+lamp. The robot successfully received and tried to execute the path through the in-home environment, but was partially unsuccessful due to hardware limitations and controller shortcomings.

In Chapter 3, we detailed the upgrades that we made to the original *home+* robot

hardware (i.e. h+lamp) to correct the issues seen in the experiments in Chapter 2. In addition, we expanded the robot's hardware and software, renaming it CuRLE in the process, to include the capacity to execute the output from motion planning algorithms, developed in Chapter 5, for the continuum element.

In Chapter 4, we defined the configuration space of a generic single-section extensible continuum element. To develop motion planning algorithms to control our CuRLE robot hardware, we first needed to understand and visualize the configuration space of the robot. While the c-space of a mobile robot is well understood, there has been no formal definition for the configuration space of a continuum element. This thesis presents the first definition of the configuration space in the general case, and then examines how the physical build of our continuum section introduces boundaries in the space. By understanding the unique features of continuum elements and how these features affect the c-space, we created a foundation to apply classical motion planning algorithms for a practical tendon-actuated continuum element. Further, we showed how we can separate the configuration space of the continuum element from that of the mobile base, and thereby characterize the full, complex configuration space of a hybrid continuum, rigid-link, mobile robot in terms of two simple 3D spaces.

In Chapter 5, we used the configuration spaces laid out in Chapter 4 to implement RRT/A* algorithms to plan collision-free paths through the task space of our CuRLE robot. We chose RRTs since their ability to be an anytime-approach fits well with the dynamic nature of CuRLE's operating environment. Given a start and goal configuration, as well as the location of all obstacles, our motion planning algorithms successfully found a collision-free path for both the mobile base and the continuum element. This is the first time that an RRT approach has been applied to a practical tendon-actuated continuum element. The hybrid nature of our CuRLE robot lends novelty to the motion planning of the mobile base, even though RRTs have widely been used to provide collision-free paths. We validated our

approach by creating software models of our robot and running simulations of practical in-home scenarios, all of which were successful.

In Chapter 6, we implement the RRT/A* algorithm for the first time on a practical robot featuring a tendon-actuated continuum element. Experiments described in this Chapter showed the robot successfully executing a path through the continuum task space. We integrated the controller from Chapter 2 with the software controlling the continuum element to create a cohesive solution that was able to simultaneously navigate both configuration spaces of the CuRLE robot hardware. Experiments reported in this Chapter show CuRLE moving through the in-home scenario to successfully complete a complex series of tasks, guided by the RRT/A* algorithms. The robot demonstrated its potential to provide in-home care and assist with aging-in-place.

7.2 Future Work

7.2.1 Migrate CuRLE to a Fully Independent System

While the CuRLE hardware was able to autonomously follow the path generated by the motion planning software, the central computer was still required to run the RRT/A* and communicate the path to the robot. Since CuRLE already has the Raspberry Pi onboard to handle its wireless communication and interface with the Arduino Due, we recommend that the motion planning code be ported over to the Pi. The code already runs in C++ and uses the Boost Graph library, which is platform independent. If the Raspberry Pi handles all of the path planning, then the robot will no longer have to rely on another computer to handle the motion planning. This will help streamline the development process and take the next step towards CuRLE being an autonomous, stand-alone unit. In addition, we envision several added features that would allow users and developers to interact more

directly with Raspberry Pi without relying on a SSH or Virtual Desktop connection. Having an LCD display and user input devices would facilitate future development and provide a method for future users to engage with the robot. In addition, having an audio system would allow CuRLE to provide added feedback to the user beyond the visual feedback of the LEDs described in Chapter 6. These relatively simple upgrades would allow future developers easier avenues for new system features and would provide users with a richer, more fulfilling experience.

7.2.2 Further Simulation Software Development

In addition to the upgrades to make CuRLE a stand-alone unit, we recommend further development of the software model and simulation environment of CuRLE. Combining the interactive GUI described in Chapter 5 for the continuum arm with the simulation code for the mobile base also detailed in that Chapter would allow future researchers to perform more comprehensive studies and experiments with the motion planning algorithms. In this thesis, we described the justification we used to separate the configurations spaces of the continuum element and the mobile base, and we were able to experiment in simulation with each individually, but we were unable to do so with both spaces in parallel. Having the capacities to run such studies would be useful in validating the motion planning output before implementing it on the hardware. In addition, while we studied different configurations of the RRT, more research can be done in this area, particularly in regards to optimization of the RRT using the RRT* algorithm. Rather than simply find a path to the goal, the motion planning can be used to find the optimal path. Since the robot will be interacting with humans to accomplish tasks, it will be important for it to execute relatively quickly. Since the CuRLE hardware and controllers may continue to present limitations, the more comprehensive simulation software that we are proposing will allow these new experiments to

be executed and tested before transporting them to the hardware.

7.2.3 Remove Restrictions on Degrees of Freedom

In this work, we restricted several of the DoF of the robot, and in future development, we intend these restrictions to be lifted. One of the advantages of the limited DoF was that we were able to visualize the RRT output in the configuration space of the robot. While removing the restrictions will necessarily make visualization impossible, advancements in the simulation software, as we have suggested, will allow us to test the full, unlocked capacity of the robot. CuRLE will be able to function with all of its DoF and we can use the simulations to verify the motion planning before executing it on the hardware.

7.2.4 Hardware Upgrades

Many of the existing hardware limitations would be eliminated by making upgrades to the CuRLE robot. First, the stiffness of the springs in the tension sensors should be increased. Given the magnitudes of the forces required in the tendons to realize the robot kinematics, the linear spring-loaded potentiometers saturated their tension readings without being able to accurately give a measurement of the tension in the tendon. While this allowed the controller to keep ”slack” out of the tendon by keeping the tension above a threshold close to the saturation value, it prevented the controller from keeping the tension balanced between opposing tendons. As such, the tension feedback controller was only marginally useful, and largely left unused during experimentation.

In addition to upgrading the tension sensors, the spool mechanism for winding the tendons around the motor shaft should be upgraded. As we previously mentioned, the spools are integral for accurately tracking the length of the tendons, which is crucial for the kinematics and control of the robot. The current hardware design does not perfectly

align the spool with the tendon as it travels up the backbone, meaning that there is a slight probability of the tendon slipping off of the spool. This occurred multiple times during testing, and were it not for the tension sensors, would have ruined controller's ability to estimate the tendon length. In addition to routing the tendons more accurately, a new, more reliable, method should be developed to track the tendon length. This issue could be potentially solved in software by creating a tracking model to account for dynamic noise (e.g. slipping) in the tendon winding, or it could solved with a more sophisticated hardware design. We envision a collection of passive spools that guarantee tendon alignment, with an optical encoder attached to one of the spools to measure the change in tendon length without having the current issue of a varying spool radius.

The final hardware upgrade that we propose for future research is to develop a new material to replace the PEX backbone. Over time, the material properties of the current backbone cause it to deform as it bends, especially when CuRLE bends to its extremes. In fact, even the constant strain of gravity causes the material to deform, which eliminates constant curvature. As the backbone strays from constant curvature, a foundational assumption for the kinematic model, the control of the robot degrades. Since we assume constant curvature, the actions needed to reach a configuration will evolve the robot into a shape that we do not model, which causes errors in the navigation. We recommend development or selection of a synthetic material (e.g. polyurethane, silicon rubber, etc.) to create a new backbone for CuRLE. The material must be elastic enough to return to its original shape after bending but also rigid enough to support its own weight (with the help of the tendons) without collapsing under gravity.

7.2.5 Integrating with a Sensing Network

Since CuRLE is the realization of a key aspect of the ”fully autonomous” end of the user control spectrum, integrating the robot with sensing technology will be vital for future development. In this thesis, the ”sensing problem” was solved by assuming knowledge of the locations of the goal configurations and the exact layout of the environments *a priori*. For the robotic lamp to claim full autonomy, it must be able to process user commands (e.g. voice recognition system) and sense the environment (e.g. vision system) to avoid obstacles and locate its goal. To accomplish this, we envision at a least two vision systems: a ”global” system that acts as a **M**oniter of the **O**perating **E**nvironment (MOE) that is mounted as an ”eye-in-sky” and detects the environment and communicates with CuRLE. The second vision system is local, mounted at the end-effector of CuRLE. This would be primarily used with locating objects to pick up/place. In addition to these vision systems, we could imagine other sensors in the home as part of the ”smart home” societal trend. Whether it be other members of *home+*, like the **L**inearly **A**ctuated **R**obotic **E**nd-table **E**lement (LAREE), or other devices that been integrated with ”smart” technology, we hope that CuRLE will be fully integrated in the home to provide care and assist with aging-in-place.

Appendices

Appendix A CuRLE Robot Software: Arduino

```
#include <SPI.h>
#include <pwm_lib.h>
#include "_aux.h"

// Continuum State Variables
const float s = 1.03;
const float d = 0.022;

int32_t eCount[5];
int32_t prevCnt[] = { 0, 0, 0, 0, 0, 0 };
const int32_t eCountZero[] = { 77514, 72058, 70443, 73889, 0 };
float spooledLen[4];
float K_tendon[] = { 0.15, 0.001, 0.02, 0.15, 0.001, 0.02};

float u = 0;
float v = 0;
float w = 0;

float u_set = 0;
float v_set = 0;
float w_set = 0;

bool debugPrint = false;
bool ESTOP = true;
bool initialized = false;
bool idle = true;

// Mobile Base
float mu[3];
```

```
30 int muPtr = MU_IDLE;
31 float stpt[] = { 0, 0 };
32 float K_mobile[] = { 0.15, 0.001, 0.023, 0.025};
33
34
35 // Motors
36 MotorList motors;
37
38 // Tension Sensors
39 TensionSensor tSensor[4];
40
41 // Clock for Encoder Chips
42 // defining pwm object using pin 53, pin PB14 mapped to pin 53 on the
     DUE
43 // this object uses PWM channel 2
44 using namespace arduino_due::pwm_lib;
45 pwm<pwm_pin::PWMH2_PB14> encoderCLK;
46
47 // Gripper
48 // servo 44-> PWMH5 (PC19)
49 servo<pwm_pin::PWMH5_PC19> servo_wrist; // blue strip
50 // 25 -> u = -v,  65 -> v = 0,   95 -> u = v,   135 -> u = 0
51 int wristAngle = 65;
52 // servo 45-> PWMH6 (PC18)
53 servo<pwm_pin::PWMH6_PC18> servo_claw;
54 // 0 -> open,   90 -> closed (clear Solo)
55 int clawAngle = 0;
56
57 // Comm
58 String msg = "";
59 String cmd = "";
```

```
60
61 // Time
62 unsigned long timeNow = 0;
63 unsigned long timeLast = 0;
64 unsigned long timeLastSave = 0;
65 unsigned long timeLastPrint = 0;
66 unsigned long timeWait = 0;
67 int timeStep = 50;
68 int timeSaveInterval = 60000;
69 int timePrintInterval = 1000;
70
71 void setup() {
72   digitalWrite(RESET_PIN, HIGH);
73   pinMode(RESET_PIN, OUTPUT);
74
75   Serial.begin(9600);
76   while(!Serial);
77   Serial1.begin(9600);
78   while(!Serial1);
79
80   Serial.println("Lamp Test");
81
82   encoderCLK.start(ENC_CLK_PERIOD, ENC_CLK_DUTY);
83
84   servo_wrist.start(
85     SERVO_PWM_PERIOD, // pwm servo period
86     75000, // 1e-8 s. (1 msecs.), minimum duty value
87     225000, // 1e-8 s. (2 msecs.), maximum duty value
88     0, // minimum angle, corresponding minimum servo angle
89     140, // maximum angle, corresponding minimum servo angle
90     wristAngle // initial servo angle
```

```
  );
  servo_claw.start(
    SERVO_PWM_PERIOD, // pwm servo period
    75000, // 1e-8 s. (1 msecs.), minimum duty value
    225000, // 1e-8 s. (2 msecs.), maximum duty value
    0, // minimum angle, corresponding minimum servo angle
    140, // maximum angle, corresponding minimum servo angle
    clawAngle // initial servo angle
  );

  // Switches
  pinMode(SW_LED, OUTPUT);
  _aux::setswitch(SW_LED, OFF);

  pinMode(SW_DRV, OUTPUT);
  _aux::setswitch(SW_DRV, OFF);

  // Have to start SPI before initializing motors (for the encoders)
  SPI.begin();
  motors.init();

  // Initialize the tension sensors (analog inputs)
  analogReadResolution(12);
  tSensor[0].init(T_SENSOR1);
  tSensor[1].init(T_SENSOR2);
  tSensor[2].init(T_SENSOR3);
  tSensor[3].init(T_SENSOR4);

  // status variables
  ESTOP = true;
  initialized = false;
```

```
122
123   // request the state data from the Raspberry Pi
124   sendRasPiMsg(RPI_LOAD);
125 }
126
127 void loop() {
128   float len[4], deltaLen[4];
129   float set[4]; // drive motors use stpt[], which is global
130   static int32_t errLast[] = { 0, 0, 0, 0, 0, 0};
131
132   int tensionErr;
133   float tensionK;
134
135   static RunningSum<int32_t> errSum[6];
136   static bool once = true;
137   int32_t err[6];
138   int32_t cnt[6], deltaC[6];
139   int CPR, i, k;
140   float pwm[6];
141   const float TENDON_PWM_SAT = 100.0;
142   const float DRIVE_PWM_SAT = 75;
143   const int32_t TENDON_ERR_THRESH = 40;
144   const int32_t DRIVE_ERR_THRESH = 60;
145   const int TENSION_THRESH = 4000;
146   bool complete[] = { false, false, false, false, false, false };
147   static int completeCnt = 0;
148
149   if (once) {
150     for (int i = T_MOTOR1; i <= D_MOTOR2; ++i) {
151       errSum[i].init(10);
152     }
```

```
153     once = false;
154   }
155
156   timeNow = millis();
157
158   if (!initialized) {
159     if (timeNow - timeLast > 1000) {
160       //_aux::toggle(SW_LED);
161       timeLast = timeNow;
162     }
163   }
164   else
165   {
166
167
168
169   // should occur every 50ms
170   if ( (timeNow-timeLast) >= timeStep) {
171     // Kinematics
172     // get lengths from the current state
173     len[T_MOTOR1] = s - (v*d);
174     len[T_MOTOR3] = s + (v*d);
175     len[T_MOTOR2] = s + (u*d);
176     len[T_MOTOR4] = s - (u*d);
177
178     // update current state[u, v, w]
179     // get the current counts
180     for (i = T_MOTOR1; i <= T_MOTOR4; ++i) {
181       cnt[i] = eCount[i] + motors[i].getCount();
182       deltaC[i] = cnt[i] - prevCnt[i];
183       // this is really the delta length
```

```
184       deltaLen[i] = _aux::lengthOnSpool(motors[i].CPR(), prevCnt[i]) -
    _aux::lengthOnSpool(motors[i].CPR(), cnt[i]);
185       len[i] += deltaLen[i];
186     }
187
188     // get counts from the drive motors
189     for (i = D_MOTOR1; i <= D_MOTOR2; ++i) {
190       cnt[i] = motors[i].getCount();
191     }
192
193
194     // Set points for tendon motors
195     set[T_MOTOR1] = s - (v_set*d);
196     set[T_MOTOR3] = s + (v_set*d);
197     set[T_MOTOR2] = s + (u_set*d);
198     set[T_MOTOR4] = s - (u_set*d);
199
200
201
202     // calculate errors
203     for (i = T_MOTOR1; i <= T_MOTOR4; ++i) {
204       CPR = motors[i].CPR();
205
206       // calculate tension error (not used...)
207       tensionErr = tSensor[i].read()-TENSION_THRESH;
208       tensionErr = (tensionErr > 0) ? 0 : tensionErr;
209
210       err[i] = _aux::deltaLengthToSetCount(CPR, cnt[i], (set[i] - len[i
    ])) - cnt[i];
211
212       k = 0;
```

```
213     pwm[i] = (err[i]*K_tendon[0+k]) + (err[i] - errLast[i])/(
        static_cast<float>(timeNow-timeLast)/1000.0)*K_tendon[1+k] + errSum[
        i].add(err[i])*K_tendon[2+k];
214     pwm[i] = (pwm[i] < 0) ? max(pwm[i],-TENDON_PWM_SAT) : min(pwm[i],
        TENDON_PWM_SAT);
215    }
216
217    deltaC[D_MOTOR1] = (cnt[D_MOTOR1]-prevCnt[D_MOTOR1])-(cnt[D_MOTOR2]-
       prevCnt[D_MOTOR2]);
218    deltaC[D_MOTOR2] = (cnt[D_MOTOR2]-prevCnt[D_MOTOR2])-(cnt[D_MOTOR1]-
       prevCnt[D_MOTOR1]);
219    for (i = D_MOTOR1; i <= D_MOTOR2; ++i) {
220     err[i] = stpt[i-D_MOTOR1] - cnt[i];
221     // PID control and velocity control (i.e. try to keep them at the
        same speed)
222     pwm[i] = (err[i]*K_mobile[0]) + (err[i] - errLast[i])/(static_cast
        <float>(timeNow-timeLast)/1000.0)*K_mobile[1] + errSum[i].add(err[i
        ])*K_mobile[2] - deltaC[i]*K_mobile[3];
223     pwm[i] = (pwm[i] < 0) ? max(pwm[i],-DRIVE_PWM_SAT) : min(pwm[i],
        DRIVE_PWM_SAT);
224    }
225
226    if (timeNow - timeLastPrint > timePrintInterval) {
227     if (debugPrint) {
228       //Serial.println("spl: ["+String(spooledLen[0],4)+","+String(
          spooledLen[1],4)+","+String(spooledLen[2],4)+","+String(spooledLen
          [3],4)+"]");
229       //Serial.println("dCt: ["+String(deltaC[0])+","+String(deltaC
          [1])+","+String(deltaC[2])+","+String(deltaC[3])+"]");
230       Serial.println("dCt: ["+String(deltaC[4])+","+String(deltaC[5])+
          "]");
```

```
231         // Serial.println("dLn: ["+String(deltaLen[0],4)+","+String(
    deltaLen[1],4)+","+String(deltaLen[2],4)+","+String(deltaLen[3],4)
    +"]");
232         // Serial.println("len: ["+String(len[0],4)+","+String(len[1],4)
    +","+String(len[2],4)+","+String(len[3],4)+"]");
233         // Serial.println("set: ["+String(set[0],4)+","+String(set[1],4)
    +","+String(set[2],4)+","+String(set[3],4)+"]");
234         Serial.println("stp: ["+String(stpt[0])+","+String(stpt[1])+"]")
    ;
235         Serial.println("err: ["+String(err[0])+","+String(err[1])+","+
    String(err[2])+","+String(err[3])+","+String(err[4])+","+String(err
    [5])+"]");
236         // Serial.println("errSum: ["+String(errSum[0].sum())+","+String(
    errSum[1].sum())+","+String(errSum[2].sum())+","+String(errSum[3].
    sum())+"]");
237         Serial.println("pwm: ["+String(pwm[0],4)+","+String(pwm[1],4)+",
    "+String(pwm[2],4)+","+String(pwm[3],4)+","+String(pwm[4],4)+","+
    String(pwm[5],4)+"]");
238         // Serial.println("mCt: ["+String(motors[0].getCount())+","+
    String(motors[1].getCount())+","+String(motors[2].getCount())+","+
    String(motors[3].getCount())+"]");
239         Serial.println("mCt: ["+String(motors[4].getCount())+","+String(
    motors[5].getCount())+"]");
240         Serial.println("mu: ["+String(mu[0],4)+","+String(mu[1],4)+","+
    String(mu[2],4)+"] muPtr = "+String(muPtr));
241       }
242       timeLastPrint = timeNow;
243     }
244
245
246
```

```
247     // update
248     for (i = T_MOTOR1; i <= D_MOTOR2; ++i) {
249       if ( i <= T_MOTOR4) {
250         complete[i] = (abs(err[i]) < TENDON_ERR_THRESH);
251       }
252       else {
253         complete[i] = (abs(err[i]) < DRIVE_ERR_THRESH);
254       }
255       if (!ESTOP ) {
256           motors[i].run(static_cast<int>(pwm[i]));
257       }
258       prevCnt[i] = cnt[i];
259       errLast[i] = err[i];
260     }
261     if (_aux::checkComplete(complete, 6) && !idle) {
262       completeCnt++;
263       if (completeCnt >= 5) {
264         motors.stopall();
265         if ( advanceMu() ) {
266           sendRasPiMsg(RPI_COMPLETE);
267           _aux::blink();
268           idle = true;
269         }
270         completeCnt = 0;
271       }
272     }
273     else {
274       completeCnt = 0;
275     }
276     u = (len[T_MOTOR2] - len[T_MOTOR4]) / (2*d);
277     v = (len[T_MOTOR3] - len[T_MOTOR1]) / (2*d);
```

```
278       timeLast = timeNow;
279     }
280
281     if (timeWait > 0 && timeNow >= timeWait) {
282       timeWait = 0;
283       sendRasPiMsg(RPI_READY);
284     }
285
286     // should occur every minute (60,000 ms)
287     if (timeNow-timeLastSave >= timeSaveInterval) {
288       // save
289       sendRasPiMsg(RPI_SAVE);
290       timeLastSave = timeNow;
291     }
292     }
293 }
294
295 // return true if done with mu (action vector)
296 bool advanceMu()
297 {
298   int delta;
299   muPtr = min(muPtr+1, MU_IDLE);
300   if (muPtr == MU_IDLE) {
301     return true;
302   }
303   switch(muPtr) {
304     case MU_ROT1:
305     case MU_ROT2:
306       delta = round((mu[muPtr]) * ROBOT_RADIUS / (WHEEL_RADIUS*2*PI) *
    static_cast<float>(motors[D_MOTOR1].CPR()));
307       stpt[0] = motors[D_MOTOR1].getCount() + delta;
```

```
308         stpt[1] = motors[D_MOTOR2].getCount() + delta;
309         break;
310       case MU_TRAN:
311         delta = round((mu[muPtr]) / (WHEEL_RADIUS*2*PI) * static_cast<
      float>(motors[D_MOTOR1].CPR()));
312         stpt[0] = motors[D_MOTOR1].getCount() + delta;
313         stpt[1] = motors[D_MOTOR2].getCount() - delta;
314         break;
315       default:
316         stpt[0] = motors[D_MOTOR1].getCount();
317         stpt[1] = motors[D_MOTOR2].getCount();
318         break;
319     }
320     return false;
321 }
322
323
324 // COMM
325 void serialEvent() {
326   byte buf[512];
327   char c;
328   while( Serial.available() ) {
329     c = Serial.read();
330     msg += c;
331     if (c == ';') {
332       msg.getBytes(buf, 512);
333       Serial1.write(buf, msg.length());
334       msg = "";
335     }
336     else {
337       // for debugging: user can select a motor to interface with
```

```
338       //mid = _aux::process_char(motors,c) - 1;
339       Serial.println(_aux::countToString(motors, -1));
340       if (c == '1') {
341         motors[0].run(25);
342       }
343       else if (c == '2') {
344         motors[1].run(25);
345       }
346       else if (c == '3') {
347         motors[2].run(25);
348       }
349       else if (c == '4') {
350         motors[3].run(25);
351       }
352       else if (c == '5') {
353         motors[0].run(-25);
354       }
355       else if (c == '6') {
356         motors[1].run(-25);
357       }
358       else if (c == '7') {
359         motors[2].run(-25);
360       }
361       else if (c == '8') {
362         motors[3].run(-25);
363       }
364       else if (c == '9') {
365         motors[4].run(100);
366         motors[5].run(-100);
367       }
368       else if (c == '0') {
```

```
369         motors[4].run(-100);
370         motors[5].run(100);
371       }
372       else if (c == 'f') {
373         motors[6].run(255);
374       }
375       else if (c == 'r') {
376         motors[6].run(-255);
377       }
378       else if (c == 's') {
379         motors.stopall();
380       }
381       else if (c == 'o') {
382         _aux::setswitch(SW_LED, ON);
383       }
384       else if (c == 'p') {
385         _aux::setswitch(SW_LED, OFF);
386       }
387
388       else if (c == 't') {
389         Serial.println("TS[:] = [ "+String(tSensor[0].read())+" ,"+
    String(tSensor[1].read())+", "+String(tSensor[2].read())+", "+String
    (tSensor[3].read())+" ]");
390       }
391       else if (c == 'q') {
392         servo_claw.set_angle(90);
393       }
394       else if (c == 'a') {
395         servo_claw.set_angle(45);
396       }
397       else if (c == 'z') {
```

```
398         servo_claw.set_angle(0);
399       }
400       else if (c == 'y') {
401         wristAngle += 5;
402         Serial.println("wristAngle = " + String(wristAngle));
403       }
404       else if (c == 'h') {
405         servo_wrist.set_angle(wristAngle);
406         Serial.println("wristAngle = " + String(wristAngle));
407       }
408       else if (c == 'n') {
409         wristAngle -= 5;
410         Serial.println("wristAngle = " + String(wristAngle));
411       }
412       else if (c == 'u') {
413         clawAngle += 5;
414         Serial.println("clawAngle = " + String(clawAngle));
415       }
416       else if (c == 'j') {
417         servo_claw.set_angle(clawAngle);
418         Serial.println("clawAngle = " + String(clawAngle));
419       }
420       else if (c == 'm') {
421         clawAngle -= 5;
422         Serial.println("clawAngle = " + String(clawAngle));
423       }
424     }
425   }
426 }
427
428 void serialEvent1() {
```

```
429   char c;
430   static int wordCount = 0;
431   while( Serial1.available() ) {
432     c = Serial1.read();
433     cmd += c;
434     if (c == '/') {
435       ++wordCount;
436     }
437     if (c == ';') {
438       Serial.println("recv'd (raspi) -> " + cmd);
439       processRasPiMsg(cmd, ++wordCount);
440       cmd = "";
441       wordCount = 0;
442     }
443   }
444 }
445
446 void processRasPiMsg(String buf, int wordCount)
447 {
448   String msgWords[10];
449   String op = "";
450   String param = "";
451   String tmp = "";
452   float arg[6];
453   int enc[5];
454
455   int i, j, k, idx, c;
456
457   // split msg into words
458   j = 0; idx = 0;
459   for (i = 0; i < buf.length(); ++i) {
```

```
460     if (buf[i] == '/' || buf[i] == ';') {
461       msgWords[idx++] = buf.substring(j,i+1);
462       j = i+1;
463     }
464   }
465   for (c = 0; c < wordCount; ++c)
466   {
467     for (i = 0; i < msgWords[c].length(); ++i) {
468       if (msgWords[c][i] == ':')
469         break;
470     }
471     op = msgWords[c].substring(0,i);
472     param = msgWords[c].substring(i+1);
473 
474     if (op.compareTo("init") == 0) {
475       if (initialized) {
476         continue;
477       }
478       _aux::blink();
479       ESTOP = false;
480       initialized = true;
481     }
482     else if (op.compareTo("state") == 0) {
483       j = 0; idx = 0;
484       for (i = 0; i < param.length(); ++i) {
485         if (param[i] == ',' || param[i] == ';' || param[i] == '/') {
486           tmp = param.substring(j,i);
487           arg[idx++] = tmp.toFloat();
488           j = i+1;
489         }
490       }
```

```
491       u = arg[0];
492       v = arg[1];
493       w = arg[2];
494     }
495     else if (op.compareTo("encoder") == 0) {
496       j = 0; idx = 0;
497       for (i = 0; i < param.length(); ++i) {
498         if (param[i] == ',' || param[i] == ';' || param[i] == '/') {
499           tmp = param.substring(j,i);
500           enc[idx++] = tmp.toInt();
501           j = i+1;
502         }
503       }
504       for (int _i = T_MOTOR1; _i <= T_MOTOR4; ++_i) {
505         eCount[_i] = enc[_i];
506         spooledLen[_i] = _aux::lengthOnSpool(motors[_i].CPR(), eCount[_i
    ]);
507         prevCnt[_i] = eCount[_i];
508       }
509       eCount[4] = enc[4];
510     }
511     else if (op.compareTo("goto") == 0) {
512       j = 0; idx = 0;
513       for (i = 0; i < param.length(); ++i) {
514         if (param[i] == ',' || param[i] == ';' || param[i] == '/') {
515           tmp = param.substring(j,i);
516           arg[idx++] = tmp.toFloat();
517           j = i+1;
518         }
519       }
520       u_set = arg[0];
```

```
521       v_set = arg[1];
522       w_set = arg[2];
523       idle = false;
524     }
525     else if (op.compareTo("drive") == 0) {
526       j = 0; idx = 0;
527       for (i = 0; i < param.length(); ++i) {
528         if (param[i] == ',' || param[i] == ';' || param[i] == '/') {
529           tmp = param.substring(j,i);
530           arg[idx++] = tmp.toFloat();
531           j = i+1;
532         }
533       }
534       stpt[0] = motors[D_MOTOR1].getCount();
535       stpt[1] = motors[D_MOTOR2].getCount();
536       mu[MU_ROT1] = arg[0];
537       mu[MU_TRAN] = -arg[1];
538       mu[MU_ROT2] = arg[2];
539       muPtr = MU_PREP;
540       idle = false;
541     }
542     else if (op.compareTo("drive_gain") == 0) {
543       j = 0; idx = 0;
544       for (i = 0; i < param.length(); ++i) {
545         if (param[i] == ',' || param[i] == ';' || param[i] == '/') {
546           tmp = param.substring(j,i);
547           arg[idx++] = tmp.toFloat();
548           j = i+1;
549         }
550       }
551       K_mobile[0] = arg[0];
```

```
552       K_mobile[1] = arg[1];
553       K_mobile[2] = arg[2];
554       K_mobile[3] = arg[3];
555     }
556     else if (op.compareTo("tendon_gain") == 0) {
557       j = 0; idx = 0;
558       for (i = 0; i < param.length(); ++i) {
559         if (param[i] == ',' || param[i] == ';' || param[i] == '/') {
560           tmp = param.substring(j,i);
561           arg[idx++] = tmp.toFloat();
562           j = i+1;
563         }
564       }
565       K_tendon[0] = arg[0];
566       K_tendon[1] = arg[1];
567       K_tendon[2] = arg[2];
568       K_tendon[3] = arg[0];
569       K_tendon[4] = arg[1];
570       K_tendon[5] = arg[2];
571     }
572     else if (op.compareTo("estop") == 0) {
573       motors.stopall();
574       ESTOP = true;
575     }
576     else if (op.compareTo("resume") == 0) {
577       ESTOP = false;
578     }
579     else if (op.compareTo("status") == 0) {
580       sendRasPiMsg(RPI_STATUS);
581     }
582     else if (op.compareTo("grab") == 0) {
```

```
583       servo_claw.set_angle(90);
584     }
585     else if (op.compareTo("letgo") == 0) {
586       servo_claw.set_angle(0);
587     }
588     else if (op.compareTo("print_setpoint") == 0) {
589       sendRasPiMsg(RPI_SETPOINT);
590     }
591     else if (op.compareTo("print_state") == 0) {
592       sendRasPiMsg(RPI_STATE);
593     }
594     else if (op.compareTo("print_encoders") == 0) {
595       sendRasPiMsg(RPI_ECOUNT);
596     }
597     else if (op.compareTo("print_gain") == 0) {
598       sendRasPiMsg(RPI_GAIN);
599     }
600     else if (op.compareTo("poll") == 0) {
601       sendRasPiMsg(RPI_LOAD);
602     }
603     else if (op.compareTo("passive") == 0) {
604       _aux::setswitch(SW_DRV, ON);
605     }
606     else if (op.compareTo("active") == 0) {
607       _aux::setswitch(SW_DRV, OFF);
608     }
609     else if (op.compareTo("wait") == 0) {
610       timeWait = timeNow + param.toInt();
611     }
612     else if (op.compareTo("led_on") == 0) {
613       _aux::setswitch(SW_LED, ON);
```

```
614     }
615     else if (op.compareTo("led_off") == 0) {
616       _aux::setswitch(SW_LED, OFF);
617     }
618     else if (op.compareTo("reset") == 0) {
619       digitalWrite(RESET_PIN, LOW);
620     }
621     else if (op.compareTo("debug") == 0) {  /* debug:0;   OR   debug:1;
    */
622       if (param.toInt()) {
623         debugPrint = true;
624       }
625       else {
626         debugPrint = false;
627       }
628     }
629     else {
630       Serial.println("Invalid cmd..." + msgWords[c]);
631     }
632   }
633 }
634
635 void sendRasPiMsg(int op)
636 {
637   byte buf[512];
638   String message = "";
639   switch(op) {
640     case RPI_LOAD: // load
641       message = (!initialized) ? "load:_;" : "noop:_;";
642       break;
643     case RPI_SAVE: // save the current state and encoder values
```

```
644       message = "save:_/";
645       message += "state:"+String(u,4)+","+String(v,4)+","+String(w,4)+"/
      ";
646       message += "encoder:"+String(eCount[0]+motors[0].getCount())+",";
647       message += String(eCount[1]+motors[1].getCount())+",";
648       message += String(eCount[2]+motors[2].getCount())+",";
649       message += String(eCount[3]+motors[3].getCount())+",";
650       message += String(eCount[4]+motors[4].getCount())+";";
651       break;
652     case RPI_SETPOINT: // send the current setpoint for all
653       message = "print:_/";
654       message += "setpoint:"+String(u_set,4)+","+String(v_set,4)+","+
      String(w_set,4)+";";
655       break;
656     case RPI_STATE: // send current state
657       message = "print:_/";
658       message += "state:"+String(u,4)+","+String(v,4)+","+String(w,4)+";
      ";
659       break;
660     case RPI_ECOUNT: // send encoder counts
661       message = "print:_/";
662       message += "encoder:"+String(eCount[0]+motors[0].getCount())+",";
663       message += String(eCount[1]+motors[1].getCount())+",";
664       message += String(eCount[2]+motors[2].getCount())+",";
665       message += String(eCount[3]+motors[3].getCount())+",";
666       message += String(eCount[4]+motors[4].getCount())+";";
667       break;
668     case RPI_GAIN: // send tendon gains
669       message = "print:_/";
670       message += "tendon_gain:"+String(K_tendon[0],4)+",";
671       message += String(K_tendon[1],4)+",";
```

```
672         message += String(K_tendon[2],4)+",";
673         message += String(K_tendon[3],4)+",";
674         message += String(K_tendon[4],4)+",";
675         message += String(K_tendon[5],4)+"/";
676         message = "print:_/";
677         message += "drive_gain:"+String(K_mobile[0],4)+",";
678         message += String(K_mobile[1],4)+",";
679         message += String(K_mobile[2],4)+",";
680         message += String(K_mobile[3],4)+";";
681         break;
682       case RPI_COMPLETE:
683         message = "complete:_;";
684         break;
685       case RPI_STATUS:
686         message = "status:";
687         message += (idle) ? "idle;" : "running;";
688         break;
689       case RPI_READY:
690         message = "wait_over:__ready__;";
691         break;
692       default:
693         message = "noop:_;";
694         break;
695     }
696     message.getBytes(buf,512);
697     Serial1.write(buf, message.length());
698 }
```

Listing 1: Main CuRLE Arduino Code

```
1 //=========================HEADER
    ============================================================
```

```
/*
   Auxillary prototypes for Lamp

   AUTHOR: Zach Hawks
   DATE: March 13, 2019

   This header contains all of the auxillary functions
   for the lamp in the namespace _aux

   License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
   licenses/by-sa/3.0/)
   You're free to use this code for any venture. Attribution is greatly
   appreciated.

//=============================================================================

*/
#ifndef __LAMP_AUXILLARY__
#define __LAMP_AUXILLARY__

#include "motor_list.h"
#include "motor.h"
#include "linear_actuator.h"
#include "tension_sensor.h"
#include "lamp_def.h"
#include "running_sum.h"
#include "running_sum.cpp"

namespace _aux {
  void setswitch(int pin, int val);
  void toggle(int pin);
```

```
30   void blink();
31   bool checkComplete(bool* complete, int len);
32   String countToString(MotorList& motors, int idx);
33   int process_char(MotorList& motors, char c);
34   float getRadius(int CPR, int32_t eCount);
35   float lengthOnSpool(int CPR, int32_t eCount);
36   int32_t deltaLengthToSetCount(int CPR, int32_t eCount, float deltaLen)
       ;
37   float tLengthAbs(MotorList& motors, int idx, int32_t eCount, int32_t
       eCountZero, float s);
38   float tLength(int idx, float u, float v, float s, float d);
39 };
40
41
42 #endif /* __LAMP_AUXILLARY__ */
```

Listing 2: Auxillary header

```
1 #include "_aux.h"
2
3 float _aux::getRadius(int CPR, int32_t eCount)
4 {
5   int rev = eCount/CPR;
6   return SPOOL_RADIUS_BASE + (static_cast<float>(rev)/1000.0);
7 }
8 int32_t _aux::deltaLengthToSetCount(int CPR, int32_t eCount, float
      deltaLen)
9 {
10   // GOAL : to find the setCount value given a deltaLen and the current
       eCount
11   float setLen, boundaryLen, radius, tmpLen;
12   int32_t cnt, deltaCnt;
```

```
13
14  // get the desired length
15  setLen = lengthOnSpool(CPR, eCount) - deltaLen;
16
17  // cnt is the iteration control variable
18  cnt = eCount;
19  if (deltaLen > 0) {
20    // find the lower boundary of the current window
21    boundaryLen = lengthOnSpool(CPR, (cnt/CPR)*CPR);
22    // check if in the same "window" (radius is constant per window)
23    if (boundaryLen > setLen) {
24      // not in same window -> iterate by moving count into the next "
    window" (lower)
25      cnt = ( (cnt/CPR)*CPR ) - 1;
26    }
27    else {
28      // in same window -> so can determine exactly where setCount is
29      radius = getRadius(CPR, cnt);
30      tmpLen = lengthOnSpool(CPR, cnt);
31      deltaCnt = (tmpLen - setLen) / (2*PI*radius) * CPR;
32      return (cnt - deltaCnt);
33    }
34  }
35  else {
36    while (1) {
37      // find the upper boundary of the current window
38      boundaryLen = lengthOnSpool(CPR, ( (cnt/CPR)+1 )*CPR);
39
40      // check if in the same "window" (radius is constant per window)
41      if (boundaryLen < setLen) {
42        // not in same window -> iterate by moving count into next "
```

```
    window" (upper)
43          cnt = ( (cnt/CPR)+1 ) * CPR;
44        }
45        else {
46          // in same window -> so can determine exactly where setCount is
47          radius = getRadius(CPR, cnt);
48          tmpLen = lengthOnSpool(CPR, cnt);
49          deltaCnt = (setLen - tmpLen) / (2*PI*radius) * CPR;
50          return (cnt + deltaCnt);
51        }
52      }
53    }
54
55    return 0;
56  }
57
58  float _aux::lengthOnSpool(int CPR, int32_t eCount)
59  {
60    float len = 0;
61    int cnt = 0;
62    float rad = SPOOL_RADIUS_BASE;
63    while (cnt < eCount) {
64      if (cnt + CPR < eCount) {
65        len += 2*rad*PI;
66        cnt += CPR;
67        rad += 0.001;
68      }
69      else {
70        len += static_cast<float>(eCount - cnt) / static_cast<float>(CPR)
    * (2*rad*PI);
71        return len;
```

```
72      }
73    }
74  }
75  float _aux::tLengthAbs(MotorList& motors, int idx, int32_t eCount,
      int32_t eCountZero, float s)
76  {
77    float lenZero = lengthOnSpool(motors[idx].CPR(), eCountZero);
78    float len = lengthOnSpool(motors[idx].CPR(), eCount);
79
80    return s + (lenZero - len);
81  }
82  float _aux::tLength(int idx, float u, float v, float s, float d)
83  {
84    switch(idx) {
85      case T_MOTOR1:
86        return s - (v*d);
87      case T_MOTOR2:
88        return s + (u*d);
89      case T_MOTOR3:
90        return s + (v*d);
91      case T_MOTOR4:
92        return s - (u*d);
93      default:
94        return -1;
95    }
96  }
97
98  bool _aux::checkComplete(bool* complete, int len)
99  {
100   for (int i = 0; i < len; ++i) {
101     if (!complete[i]) {
```

```
102         return false;
103       }
104     }
105     return true;
106 }
107
108 void _aux::toggle(int pin)
109 {
110     int val = digitalRead(pin);
111     if (val) {
112       setswitch(pin, OFF);
113     }
114     else {
115       setswitch(pin, ON);
116     }
117 }
118
119 void _aux::blink()
120 {
121     setswitch(SW_LED, ON);
122     delay(100);
123     for (int i = 0; i < 5; ++i) {
124       toggle(SW_LED);
125       delay(100);
126     }
127     setswitch(SW_LED,OFF);
128 }
129
130 void _aux::setswitch(int pin, int val)
131 {
132     if (val == ON) {
```

```
133     digitalWrite(pin, HIGH);
134   }
135   else {
136     digitalWrite(pin, LOW);
137   }
138 }
139
140
141
142 int _aux::process_char(MotorList& motors, char c)
143 {
144   int i;
145   if (c >= '1' && c <= '7') {
146     i = c - '0';
147     return i;
148   }
149   return 0;
150 }
151
152 String _aux::countToString(MotorList& motors, int idx)
153 {
154   String str;
155   if (idx < 0)  {/* all */
156     str = "count[:] = [ ";
157     for (int i = 0; i < M_CNT-1; ++i) {
158       str = str + String(motors[i].getCount()) + ", ";
159     }
160     str = str + String(motors[M_CNT-1].getCount()) + " ]";
161   }
162   else {
163     str = "count[" + String(idx+1) + "] = [ " + String(motors[idx].
```

```
    getCount()) + " ]";
  }
  return str;
}
```

Listing 3: Auxillary functions

```
//=========================HEADER
    ================================================================
/*
   LS7366 Quadrature Counter Interface Library
   AUTHOR: Zach Hawks
   DATE: March 9, 2019

   This is a simple library program to read encoder counts
   collected by the LS7366 chip (no breakout board). This code
   is derived from code developed by Jason Traud.
   (https://github.com/SuperDroidRobots/Encoder-Buffer-Breakout)

   License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
    licenses/by-sa/3.0/)
   You're free to use this code for any venture. Attribution is greatly
    appreciated.

//=============================================================================

*/

#ifndef __ENCODER_CHIP_H__
#define __ENCODER_CHIP_H__

#include <inttypes.h>
```

```
22 #include <Wire.h>
23 #include "Arduino.h"
24
25 class EncoderChip {
26 public:
27   EncoderChip(int ssPin);
28   EncoderChip(const EncoderChip& ec);
29
30   EncoderChip();
31   EncoderChip& operator=(const EncoderChip&) = delete;
32
33   void init();
34   int32_t read();
35   void clear();
36
37   // debug method
38   int getpin() const { return _ssPin; }
39
40 private:
41   static int ID;
42   int _id;
43   int _ssPin;
44 };
45
46 #endif /*__ENCODER_CHIP_H__*/
```

Listing 4: Encoder Chip Class Header

```
1 #include "encoder_chip.h"
2 #include <SPI.h>
3
4 int EncoderChip::ID = 0;
```

```
5
6  EncoderChip::EncoderChip(int ssPin) : _id(ID++), _ssPin(ssPin)
7  {
8  }
9
10 EncoderChip::EncoderChip(const EncoderChip& ec) : _id(ec._id), _ssPin(ec
       ._ssPin)
11 {
12 }
13
14 EncoderChip::EncoderChip() : _id(-1), _ssPin(-1)
15 {
16 }
17
18 void EncoderChip::init()
19 {
20   if (_id < 0) {
21     return;
22   }
23   // Set slave selects as outputs
24   pinMode(_ssPin, OUTPUT);
25
26   // Raise select pins
27   // Communication begins when you drop the individual select signsl
28   digitalWrite(_ssPin, HIGH);
29
30   // Initialize encoder
31   //     Clock division factor: 0
32   //     Negative index input
33   //     free-running count mode
34   //     x4 quatrature count mode (four counts per quadrature cycle)
```

```
35   // NOTE: For more information on commands, see datasheet
36   digitalWrite(_ssPin, LOW);         // Begin SPI conversation
37   SPI.transfer(0x88);                        // Write to MDR0
38   SPI.transfer(0x03);                        // Configure to 4 byte mode
39   digitalWrite(_ssPin, HIGH);        // Terminate SPI conversation
40 }
41
42 int32_t EncoderChip::read()
43 {
44   if (_id < 0) {
45     return 0;
46   }
47   // Initialize temporary variables for SPI read
48   int32_t count_1, count_2, count_3, count_4;
49   int32_t cnt;
50
51   digitalWrite(_ssPin, LOW);       // Begin SPI conversation
52   SPI.transfer(0x60);                       // Request count
53   count_1 = SPI.transfer(0x00);             // Read highest order byte
54   count_2 = SPI.transfer(0x00);
55   count_3 = SPI.transfer(0x00);
56   count_4 = SPI.transfer(0x00);             // Read lowest order byte
57   digitalWrite(_ssPin, HIGH);      // Terminate SPI conversation
58
59   // Calculate encoder count
60   cnt = (count_1 << 8) + count_2;
61   cnt = (cnt << 8) + count_3;
62   cnt = (cnt << 8) + count_4;
63
64   return cnt;
65 }
```

```
66
67 void EncoderChip::clear()
68 {
69   if (_id < 0) {
70     return;
71   }
72     // Set encoder's data register to 0
73   digitalWrite(_ssPin, LOW);       // Begin SPI conversation
74   // Write to DTR
75   SPI.transfer(0x98);
76   // Load data
77   SPI.transfer(0x00);  // Highest order byte
78   SPI.transfer(0x00);
79   SPI.transfer(0x00);
80   SPI.transfer(0x00);  // lowest order byte
81   digitalWrite(_ssPin, HIGH);      // Terminate SPI conversation
82
83   delayMicroseconds(100);  // provides some breathing room between SPI
      conversations
84
85   // Set encoder current data register to center
86   digitalWrite(_ssPin, LOW);       // Begin SPI conversation
87   SPI.transfer(0xE0);
88   digitalWrite(_ssPin, HIGH);      // Terminate SPI conversation
89 }
```

Listing 5: Encoder Chip Class Source

```
1 //=========================HEADER
     ============================================================
2 /*
3    Definition for home+ lamp: CuRLE
```

```
4
5     AUTHOR: Zach Hawks
6     DATE: March 13, 2019
7
8     This header contains all of the definitions for the pin
9     mappings for the lamp
10
11    License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
      licenses/by-sa/3.0/)
12    You're free to use this code for any venture. Attribution is greatly
      appreciated.
13
14 */
15 //
      ================================================================================
16
17 #ifndef __LAMP__
18 #define __LAMP__
19
20 // Frequencies for PWM
21 #define ENC_CLK_PERIOD    20       // in 1e-8s ==> period=200ns, freq=5MHz
22 #define ENC_CLK_DUTY      10       // in 1e-8s ==> duty=100ns, duty_cycle
      =50%
23 #define SERVO_PWM_PERIOD 2000000 // in 1e-8s ==> period=20ms, freq=50Hz
24
25 // Other Constants
26 #define ON        1
27 #define OFF       0
28 #define M_CNT     7
29 #define SPOOL_RADIUS_BASE   (0.0125) // m
```

```
30 #define WHEEL_RADIUS        (0.042)  // m
31 #define ROBOT_RADIUS        (0.223)  // m
32
33 // Action "States"
34 #define MU_PREP    (-1)
35 #define MU_ROT1    0
36 #define MU_TRAN    1
37 #define MU_ROT2    2
38 #define MU_IDLE    3
39
40 // Opcodes for Raspberry Pi Messages
41 #define RPI_LOAD          0
42 #define RPI_SAVE          1
43 #define RPI_SETPOINT      2
44 #define RPI_STATE         3
45 #define RPI_ECOUNT        4
46 #define RPI_GAIN          5
47 #define RPI_COMPLETE      6
48 #define RPI_STATUS        7
49 #define RPI_READY         8
50
51
52 // For Motor List Indexing
53 #define MOTOR1   0
54 #define MOTOR2   1
55 #define MOTOR3   2
56 #define MOTOR4   3
57 #define MOTOR5   4
58 #define MOTOR6   5
59 #define MOTOR7   6
60
```

```
/*=====================================================*/
/*====== PINS ======*/
/*=====================================================*/

// Reset Pin
#define RESET_PIN          A0

// Tendon Motor Pins
#define T_MOTOR1           MOTOR1
#define T_MOTOR1_PWM       11
#define T_MOTOR1_DIR       29
#define T_MOTOR1_ENC       30
#define T_MOTOR1_CPR       12659

#define T_MOTOR2           MOTOR2
#define T_MOTOR2_PWM       2
#define T_MOTOR2_DIR       35
#define T_MOTOR2_ENC       32
#define T_MOTOR2_CPR       12659

#define T_MOTOR3           MOTOR3
#define T_MOTOR3_PWM       4           /*10*/
#define T_MOTOR3_DIR       39          /*37*/
#define T_MOTOR3_ENC       36          /*34*/
#define T_MOTOR3_CPR       12659
/* due to wiring, had to switch motor3 and motor4 */
#define T_MOTOR4           MOTOR4
#define T_MOTOR4_PWM       10           /*4*/
#define T_MOTOR4_DIR       37          /*39*/
#define T_MOTOR4_ENC       34          /*36*/
#define T_MOTOR4_CPR       12659
```

```
92
93  // Drive Motor Pins
94  #define D_MOTOR1          MOTOR5
95  #define D_MOTOR1_PWM      12
96  #define D_MOTOR1_DIR      31
97  #define D_MOTOR1_ENC      38
98  #define D_MOTOR1_CPR      3200
99
100 #define D_MOTOR2          MOTOR6
101 #define D_MOTOR2_PWM      5
102 #define D_MOTOR2_DIR      41
103 #define D_MOTOR2_ENC      40
104 #define D_MOTOR2_CPR      3200
105
106 // Worm Gear Motor Pins
107 #define W_MOTOR           MOTOR7
108 #define W_MOTOR_PWM       3
109 #define W_MOTOR_DIR       43
110 #define W_MOTOR_ENC       42
111 #define W_MOTOR_CPR       (6533*30)
112
113 // Linear Actuator Pins
114 #define L_MOTOR           0
115 #define L_MOTOR_PWM       13
116 #define L_MOTOR_DIR       33
117 #define L_MOTOR_IR        A5
118
119 // Gripper Servo Pin
120 #define WRIST_PIN         44
121 #define CLAW_PIN          45
122
```

```
123 // Tension Sensors
124 #define T_SENSOR1       A1
125 #define T_SENSOR2       A2
126 #define T_SENSOR3       A3   /*A2*/
127 #define T_SENSOR4       A4   /*A3*/
128
129 // Switches (Relay Control Pins)
130 #define SW_LED          50
131 #define SW_DRV          52
132 /*=======================================================*/
133
134 #endif /*__LAMP__*/
```

Listing 6: Macros

```
1 //=========================HEADER
    ================================================================
2 /*
3    Linear Actuator Library
4    AUTHOR: Zach Hawks
5    DATE: March 13, 2019
6
7    This is a simple library to interface with a linear actuator
8    controlled by a dual H-bridge (DIR and PWM).
9
10   License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
     licenses/by-sa/3.0/)
11   You're free to use this code for any venture. Attribution is greatly
     appreciated.
12
13 //=============================================================================
```

```
*/

#ifndef __LINEAR_ACTUATOR_H__
#define __LINEAR_ACTUATOR_H__

#include <inttypes.h>
#include <Wire.h>
#include "Arduino.h"

class LinearActuator {
public:
  LinearActuator(int dirPin, int pwmPin, int irPin);
  LinearActuator() = delete;
  LinearActuator(const LinearActuator&) = delete;

  LinearActuator& operator=(const LinearActuator&) = delete;

  void run(int val);
  int length();

private:
  int _dirPin;
  int _pwmPin;
  int _irPin;
  int _zeroPt;

  int _read();
};

```

```
#endif /*__LINEAR_ACTUATOR_H__*/
```

Listing 7: Linear Actuator Class Header

```
#include "linear_actuator.h"

LinearActuator::LinearActuator(int dirPin, int pwmPin, int irPin) :
  _dirPin(dirPin),
  _pwmPin(pwmPin),
  _irPin(irPin),
  _zeroPt(0.0)
{
  pinMode(_dirPin, OUTPUT);
  digitalWrite(_dirPin, LOW);

  pinMode(_pwmPin, OUTPUT);
  analogWrite(_pwmPin, 0);

  pinMode(_irPin, INPUT);
  _zeroPt = _read();
}

void LinearActuator::run(int val)
{
  //                        wind()    unwind()
  int dir = (val <= 0) ?   LOW :      HIGH;
  int pwm = abs(val);

  // direction
  digitalWrite(_dirPin, dir);

  // speed
```

```
29    analogWrite(_pwmPin, pwm);
30  }
31
32  int LinearActuator::length()
33  {
34    int tmp = _read() - _zeroPt;
35
36    // TODO: convert the raw analog "int" to a distance
37
38    return tmp;
39  }
40
41  int LinearActuator::_read()
42  {
43    int tmp, i, cnt;
44    cnt = 5;
45    tmp = 0;
46    for (i = 0; i < cnt; ++i) {
47      tmp += analogRead(_irPin);
48    }
49    return tmp / cnt;
50  }
```

Listing 8: Linear Actuator Class Source

```
1  //=========================HEADER
       ============================================================
2  /*
3     Encoded DC Gear-Motor Library
4     AUTHOR: Zach Hawks
5     DATE: March 13, 2019
6
```

```
7     This is a simple library to interface with a DC gear-motor with
8     a quadrature encoder, controlled by a dual H-bridge (DIR and PWM).
9
10    License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
      licenses/by-sa/3.0/)
11    You're free to use this code for any venture. Attribution is greatly
      appreciated.
12
13  //=============================================================================

14  */
15
16  #ifndef __MOTOR_H__
17  #define __MOTOR_H__
18
19  #include <inttypes.h>
20  #include <Wire.h>
21  #include "Arduino.h"
22  #include "encoder_chip.h"
23
24  class Motor {
25  public:
26    Motor(int dirPin, int pwmPin, int cntPerRev, int encSlaveSelectPin);
27    Motor(const Motor& m);
28
29    Motor();
30    Motor& operator=(const Motor&) = delete;
31
32    void init();
33    void run(int val);
34    void stop();
```

```
35    int32_t getCount() { return _encoder.read(); }
36    int CPR() const { return _cntPerRev; }
37    int getid() const { return _id; }
38
39    String string() const;
40
41  private:
42    static int ID;
43    int _id;
44    EncoderChip _encoder;
45    int _cntPerRev;
46    int _dirPin;
47    int _pwmPin;
48  };
49
50  #endif /*__MOTOR_H__*/
```

Listing 9: Motor Class Header

```
1  #include "motor.h"
2
3  // motor index
4  int Motor::ID = 0;
5
6  Motor::Motor(int dirPin, int pwmPin, int cntPerRev, int
       encSlaveSelectPin) :
7    _id(ID++),
8    _encoder(encSlaveSelectPin),
9    _cntPerRev(cntPerRev),
10   _dirPin(dirPin),
11   _pwmPin(pwmPin)
12 {
```

```
13 }
14
15 Motor::Motor(const Motor& m) :
16   _id(m._id),
17   _encoder(m._encoder),
18   _cntPerRev(m._cntPerRev),
19   _dirPin(m._dirPin),
20   _pwmPin(m._pwmPin)
21 {
22 }
23
24 Motor::Motor() :
25     _id(-1),
26   _encoder(),
27   _cntPerRev(1),
28   _dirPin(-1),
29   _pwmPin(-1)
30 {
31 }
32
33 void Motor::init()
34 {
35   if (_id < 0) {
36     return;
37   }
38   pinMode(_dirPin, OUTPUT);
39   digitalWrite(_dirPin, LOW);
40
41   pinMode(_pwmPin, OUTPUT);
42   analogWrite(_pwmPin, 0);
43   _encoder.init();
```

```
44    _encoder.clear();
45  }
46
47  String Motor::string() const
48  {
49    if (_id < 0) {
50      return "invalid";
51    }
52    String s = "motor["+String(_id+1)+"] [dir="+String(_dirPin);
53    s = s + ", pwm="+String(_pwmPin)+", cpr="+String(_cntPerRev)+", enc="+
       String(_encoder.getpin());
54    return s;
55  }
56
57  void Motor::stop()
58  {
59    run(0);
60  }
61
62  void Motor::run(int val)
63  {
64    if (_id < 0) {
65      return;
66    }
67    // direction
68    if (val <= 0) {
69      digitalWrite(_dirPin, LOW);   // wind()
70    }
71    else {
72      digitalWrite(_dirPin, HIGH); // unwind()
73    }
```

```
74    // speed
75    analogWrite(_pwmPin, abs(val));
76  }
```

Listing 10: Motor Class Source

```
1  //========================HEADER
       ================================================================
2  /*
3     Encoded DC Gear-Motor List Library
4     AUTHOR: Zach Hawks
5     DATE: March 15, 2019
6
7     This is a simple class to hold multiple Motor objects
8
9     License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
       licenses/by-sa/3.0/)
10    You're free to use this code for any venture. Attribution is greatly
       appreciated.
11
12 //=============================================================================
13 */
14
15 #ifndef __MOTOR_LIST_H__
16 #define __MOTOR_LIST_H__
17
18 #include <inttypes.h>
19 #include <Wire.h>
20 #include "Arduino.h"
21 #include "motor.h"
22
```

```
23 class MotorList {
24 public:
25   MotorList();
26
27   MotorList(const MotorList& ml) = delete;
28   MotorList& operator=(const MotorList&) = delete;
29
30   Motor& operator[](int i);
31   void init();
32   void stopall();
33
34 private:
35   Motor _invalid;
36   Motor _tMotor1;
37   Motor _tMotor2;
38   Motor _tMotor3;
39   Motor _tMotor4;
40   Motor _dMotor1;
41   Motor _dMotor2;
42   Motor _wMotor;
43   int _length;
44 };
45
46 #endif /*__MOTOR_H__*/
```

Listing 11: Motor List Class Header

```
1 #include "motor_list.h"
2 #include "lamp_def.h"
3
4 MotorList::MotorList() :
5   _invalid(),
```

```
6     _tMotor1(T_MOTOR1_DIR, T_MOTOR1_PWM, T_MOTOR1_CPR, T_MOTOR1_ENC),
7     _tMotor2(T_MOTOR2_DIR, T_MOTOR2_PWM, T_MOTOR2_CPR, T_MOTOR2_ENC),
8     _tMotor3(T_MOTOR3_DIR, T_MOTOR3_PWM, T_MOTOR3_CPR, T_MOTOR3_ENC),
9     _tMotor4(T_MOTOR4_DIR, T_MOTOR4_PWM, T_MOTOR4_CPR, T_MOTOR4_ENC),
10    _dMotor1(D_MOTOR1_DIR, D_MOTOR1_PWM, D_MOTOR1_CPR, D_MOTOR1_ENC),
11    _dMotor2(D_MOTOR2_DIR, D_MOTOR2_PWM, D_MOTOR2_CPR, D_MOTOR2_ENC),
12    _wMotor(W_MOTOR_DIR, W_MOTOR_PWM, W_MOTOR_CPR, W_MOTOR_ENC),
13    _length(M_CNT)
14  {
15
16  }
17
18  void MotorList::init()
19  {
20    _tMotor1.init();
21    _tMotor2.init();
22    _tMotor3.init();
23    _tMotor4.init();
24    _dMotor1.init();
25    _dMotor2.init();
26    _wMotor.init();
27  }
28
29  Motor& MotorList::operator[](int i)
30  {
31    switch(i) {
32      case 0:
33        return _tMotor1;
34      case 1:
35        return _tMotor2;
36      case 2:
```

```
37        return _tMotor3;
38      case 3:
39        return _tMotor4;
40      case 4:
41        return _dMotor1;
42      case 5:
43        return _dMotor2;
44      case 6:
45        return _wMotor;
46      default:
47        return _invalid;
48    }
49 }
50
51 void MotorList::stopall()
52 {
53   _tMotor1.stop();
54   _tMotor2.stop();
55   _tMotor3.stop();
56   _tMotor4.stop();
57   _dMotor1.stop();
58   _dMotor2.stop();
59   _wMotor.stop();
60 }
```

Listing 12: Motor List Class Source

```
1 //=========================HEADER
      ============================================================
2 /*
3    Library for running sum/average
4    AUTHOR: Zach Hawks
```

```
5     DATE: March 13, 2019
6
7     This is a simple class/library that keeps a running sum/average
8
9     License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
      licenses/by-sa/3.0/)
10    You're free to use this code for any venture. Attribution is greatly
      appreciated.
11
12 //=============================================================================
13 */
14
15 #ifndef __RUNNING_SUM_H__
16 #define __RUNNING_SUM_H__
17
18 #include <inttypes.h>
19 #include <Wire.h>
20 #include "Arduino.h"
21
22 template <class T>
23 class RunningSum {
24 public:
25   RunningSum();
26   RunningSum( int qty );
27   ~RunningSum();
28
29   RunningSum& operator=(const RunningSum& rs) = delete;
30   RunningSum(const RunningSum<T>& rs) = delete;
31
32   void init( int qty );
```

```
33   T add( T item );
34   T sum() const { return _sum; }
35   int capacity() const { return _size; }
36   int size() const { return _cnt; }
37   void clear();
38   float avg() const;
39
40 private:
41   int _size;
42   T* _queue;
43   int _cnt;
44   int _ptr;
45   T _sum;
46 };
47
48 #endif /*__RUNNING_SUM_H__*/
```

Listing 13: Running Sum Class Header

```
1  #include "running_sum.h"
2
3  template <class T>
4  RunningSum<T>::RunningSum() : _size(-1), _queue(nullptr), _cnt(-1), _ptr
       (-1), _sum(0)
5  {
6
7  }
8  template <class T>
9  RunningSum<T>::RunningSum( int qty ) : _size(qty), _queue(nullptr), _cnt
       (0), _ptr(0), _sum(0)
10 {
11   _queue = (T*)calloc(_size, sizeof(T));
```

```
12 }
13 template <class T>
14 RunningSum<T>::~RunningSum()
15 {
16   if (_size != -1) {
17     free(_queue);
18   }
19 }
20
21 template <class T>
22 void RunningSum<T>::init( int qty )
23 {
24   _size = qty;
25   _queue = (T*)calloc(_size, sizeof(T));
26   _cnt = 0;
27   _ptr = 0;
28   _sum = 0;
29 }
30 template <class T>
31 T RunningSum<T>::add( T item )
32 {
33   if ( _size <= 0) {
34     return _sum;
35   }
36   _sum = _sum - _queue[_ptr] + item;
37   _queue[_ptr] = item;
38
39   _ptr = (_ptr+1)%_size;
40   _cnt = max(_cnt+1,_size);
41
42   return _sum;
```

```
}

template <class T>
void RunningSum<T>::clear()
{
  if (_size < 0) {
    return;
  }
  memset(_queue, 0, _size*sizeof(T));
  _cnt = 0;
  _ptr = 0;
  _sum = 0;
}

template <class T>
float RunningSum<T>::avg() const
{
  if (_cnt <= 0) {
    return 0;
  }
  return (static_cast<float>(_sum) / static_cast<float>(_cnt));
}
```

Listing 14: Running Sum Class Source

```
//=========================HEADER
    ============================================================
/*
   Linear Spring Loaded Potentiometer (Tension Sensor) Interface Library
   AUTHOR: Zach Hawks
   DATE: March 13, 2019

```

```
7     This is a simple library program to read "tension" from
8     a spring loaded linear potentiometer.
9
10    License: CCAv3.0 Attribution-ShareAlike (http://creativecommons.org/
      licenses/by-sa/3.0/)
11    You're free to use this code for any venture. Attribution is greatly
      appreciated.
12
13  //=============================================================================
14  */
15
16  #ifndef __TENSION_SENSOR_H__
17  #define __TENSION_SENSOR_H__
18
19  #include <inttypes.h>
20  #include <Wire.h>
21  #include "Arduino.h"
22
23  class TensionSensor {
24  public:
25    TensionSensor();
26    TensionSensor(int pin);
27    TensionSensor(const TensionSensor&) = delete;
28
29    TensionSensor& operator=(const TensionSensor&) = delete;
30
31    int init(int pin);
32    int init();
33    int read();
34
```

```
35 private:
36   int _pin;
37 };
38
39 #endif /*__TENSION_SENSOR_H__*/
```

Listing 15: Tension Sensor Class Header

```
1  #include "tension_sensor.h"
2
3  TensionSensor::TensionSensor() : _pin(-1)
4  {
5    //pinMode(_pin, INPUT);
6  }
7
8  TensionSensor::TensionSensor(int pin) : _pin(pin)
9  {
10   //pinMode(_pin, INPUT);
11 }
12
13 int TensionSensor::init(int pin)
14 {
15   _pin = pin;
16   pinMode(_pin, INPUT);
17 }
18
19 int TensionSensor::init()
20 {
21   if (_pin == -1) {
22     return -1;
23   }
24   pinMode(_pin, INPUT);
```

```
}

int TensionSensor::read()
{
  int i, tmp, cnt;
  cnt = 5;
  tmp = 0;
  for (i = 0; i < cnt; ++i) {
    tmp += analogRead(_pin);
  }
  return tmp / cnt;
}
```

Listing 16: Tension Sensor Class Source

Appendix B CuRLE Robot Software: Raspberry Pi

```
import signal
import serial
import time
import sys
import socket
import select
import json
import math

global _serial, _socket, Kgain

def cleanup(sig, fram):
    _socket.close()
    _serial.close()
```

```
15      with open("/home/pi/lamp/gain.json", "w") as f:
16          json.dump(Kgain, f, indent=4)
17      print("exiting...")
18      sys.exit(0)
19
20  def reply_arduino( msg ):
21      word = msg.split('/')
22      if (word[0] == "save:_"):
23          _string = word[1]
24          _string = _string[:-1] + ";" + word[2]
25          save_state(_string)
26          #
27      elif (word[0] == "print:_"):
28          print("arduino -> ", word[1])
29
30      elif (word[0] == "load:_;"):
31          state = load_state()
32          text = replyForLoad(state, Kgain)
33          if (text[-1] != ';'):
34              text = text + ';'
35          _serial.write(text.encode('utf-8'))
36
37      elif (word[0] == "complete:_;"):
38          print("setpoint reached")
39
40      elif (word[0] == "noop:_;"):
41          print("arduino goofed...")
42
43
44  def relay_arduino( cmd ):
45      if (cmd[-1] != ';'):
```

```
46          cmd = cmd + ';'
47      tmp = cmd[:-1].split(':')
48      if tmp[0] == "tendon_gain":
49          K = tmp[1].split(',')
50          i = 0
51          for k in K:
52              Kgain["K_tendon"][i] = float(k)
53              if (len(K) == 3):
54                  Kgain["K_tendon"][i+3] = float(k)
55              i = i+1
56      elif tmp[0] == "drive_gain":
57          K = tmp[1].split(',')
58          i = 0
59          for k in K:
60              Kgain["K_drive"][i] = float(k)
61              i = i + 1
62
63      print('sending -> ', cmd)
64      _serial.write(cmd.encode('utf-8'))
65
66  def load_state():
67      with open("/home/pi/lamp/lamp.json", "r") as f:
68          data = f.read()
69      #print(data)
70      o = json.loads(data)
71      return o
72
73  def save_state(state):
74      cmds = state.split(';')
75      if (len(cmds) != 3):
76          print("has to be 2 cmds...")
```

```
77          return
78      # "state" : { "u": _u, "v": _v, "w": _w, "raw": [_u, _v, _w] }
79      tmp = cmds[0].split(':')
80      if (tmp[0] != 'state'):
81          print("state: command not present...")
82          return
83      raw = tmp[1].split(',')
84      u = float(raw[0])
85      v = float(raw[1])
86      w = float(raw[2])
87      raw = [u,v,w]
88
89      # "spool_rad" : [_r1, _r2, _r3, _r4]
90      tmp = cmds[1].split(':')
91      if (tmp[0] != 'encoder'):
92          print("encoder: command not present...")
93          return
94      cnts = tmp[1].split(',')
95      enc = []
96      for c in cnts:
97          enc.append(int(c))
98
99      jstring = json.dumps({"state" : { "u": u, "v": v, "w": w, "raw": raw
        },"encoder" : enc},indent=4)
100     #print(jstring)
101     with open("/home/pi/lamp/lamp.json", "w") as f:
102         json.dump({"state" : { "u": u, "v": v, "w": w, "raw": raw },"
    encoder" : enc}, f,indent=4)
103
104     return
105
```

```
106 def load_gain():
107     with open("/home/pi/lamp/gain.json", "r") as f:
108         data = f.read()
109     #print(data)
110     o = json.loads(data)
111     return o
112 
113 def replyForLoad(data, gain):
114     state = data["state"]
115     raw = state["raw"]
116     _string = "state:"+"%0.4f"%state["u"]+","
117     _string += "%0.4f"%state["v"]+","
118     _string += "%0.4f"%state["w"]+"/"
119 
120     _string1 = "encoder:"
121     for r in data["encoder"]:
122         _string1 = _string1 + str(r)+","
123     _string1 = _string1[:-1] + '/'
124 
125     _string2 = "tendon_gain:"
126     for k in gain["K_tendon"]:
127         _string2 = _string2 + str(k)+","
128     _string2 = _string2[:-1] + '/'
129 
130     _string3 = "drive_gain:"
131     for k in gain["K_drive"]:
132         _string3 = _string3 + str(k)+","
133     _string3 = _string3[:-1] + '/'
134 
135     _string4 = "goto:"+"%0.4f"%state["u"]+","
136     _string4 += "%0.4f"%state["v"]+","
```

```
137     _string4 += "%0.4f"%state["w"]+"/"
138
139     _string5 = "init:_;"
140
141     return _string+_string1+_string2+_string3+_string4+_string5
142
143
144 signal.signal(signal.SIGINT, cleanup)
145
146 TIMEOUT = 3
147 # Setup serial comm
148 _serial = serial.Serial(port='/dev/ttyUSB0', baudrate=9600, timeout=
    TIMEOUT)
149 msg = b''
150
151 # Setup wireless comm
152 PORT = 16996
153 _socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
154 _socket.setblocking(0)
155 _socket.bind(('',PORT))
156 _socket.listen(2)
157
158
159 Kgain = load_gain()
160
161 read_list = [_socket, sys.stdin]
162 #read_list = [_socket]
163 connected = False
164 while True:
165     readable, writable, errored = select.select(read_list, [], [],
    TIMEOUT)
```

```
166     for s in readable:
167         # new connection available
168         if s is _socket:
169             _client, addr = _socket.accept()
170             read_list.append(_client)
171             connected = True
172         # existing connection has data available
173         elif s is sys.stdin:
174             userIn = sys.stdin.readlines()
175             for line in userIn:
176                 print("echo -> ", line[:-1])
177                 if line == '\n':
178                     line = ';;'
179                 relay_arduino(line[:-1])
180         else:
181             cmd = s.recv(1024)
182             print("recv'd (client) -> ", cmd)
183             if (cmd):
184                 relay_arduino(cmd.decode("utf-8"))
185             else:
186                 s.close()
187                 read_list.remove(s)
188                 connected = False
189 
190     # Read data from Arduino
191     while (_serial.in_waiting):
192         c = _serial.read(1)
193         msg += c
194         if (c == b';'):
195             print("recv'd -> (arduino)", msg.decode("utf-8"))
196             if (connected):
```

```
197                     _client.sendall(msg)
198                 else:
199                     reply_arduino(msg.decode("utf-8"))
200                 msg = b''
```

Listing 17: Python script that controls the robot running on Raspberry Pi

Appendix C CuRLE Robot Software: Central Computer

```
1  import socket
2  import sys
3  import select
4  import signal
5
6  def cleanup(sigint, frame):
7      _socket.close()
8      print 'exiting...'
9      sys.exit(0)
10
11 signal.signal(signal.SIGINT, cleanup)
12
13 keepGoing = True
14 HOST = '198.21.183.61'
15 PORT = 16996
16 TIMEOUT = 5
17
18 _socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19 _socket.settimeout(TIMEOUT)
20
21 try:
22     _socket.connect((HOST,PORT))
```

```
23      read_list = [_socket]
24      _socket.sendall('init;'.encode("utf-8"))
25      while keepGoing:
26          shouldReply = False
27          # read replies
28          readable, writable, errored = select.select(read_list, [], [],
    TIMEOUT)
29          for s in readable:
30              msg = s.recv(1024)
31              print "recv'd (raspi) -> " , msg
32              shouldReply = True
33
34          if (shouldReply):
35              text = raw_input('reply -> ')
36              if (text[0] == '/'):
37                  keepGoing = False
38              if (text[-1] != ';'):
39                  text = text + ';'
40              _socket.sendall(text.encode("utf-8"))
41
42  except Exception as e:
43      print 'exception ->', e
44  finally:
45      _socket.close()
```

Listing 18: Python script to communicate RRT output to CuRLE

Appendix D Motion Planning Software

```
#ifndef __CONFIG_SPACE_H__
#define __CONFIG_SPACE_H__

#include "RobotState.h"
#include <vector>

class ConfigSpace{
public:
  ConfigSpace(std::string fname, int d);
  ~ConfigSpace();

  int collisionPoints() const;
  bool collidedState(RobotState &st);
  bool collidedEdge(RobotState &st1, RobotState &st2);

private:
  const double _pi = 3.1415926535897;
  int dim;
  int *size;
  int total_size;
  uint8_t *cspace;

  ConfigSpace() = delete;
  ConfigSpace(const ConfigSpace &cs) = delete;
  ConfigSpace& operator=(const ConfigSpace &cs) = delete;
};


```

```
#endif // !__CONFIG_SPACE_H__
```

Listing 19: ConfigSpace Class header

```
#include "ConfigSpace.h"
#include <iostream>
#include <string>
#include <fstream>

ConfigSpace::ConfigSpace(std::string fname, int d) : dim(2), size(new
    int[dim]), total_size(1), cspace(nullptr)
{
  int _dim;
  FILE *fptr;
  char buf[256];
  fopen_s(&fptr, fname.c_str(), "rb");
  if (fptr == nullptr) {
    std::cout << "Could not open file: " << fname << std::endl;
    return;
  }
  fgets(buf, 256, fptr);
  sscanf_s(buf, "%d %d %d", &_dim, size + 0, size + 1);
  for (int i = 0; i < dim; ++i) {
    total_size *= size[i];
  }
  cspace = new uint8_t[total_size];
  fread(cspace, 1, total_size, fptr);
  fclose(fptr);
}

ConfigSpace::~ConfigSpace()
{
```

```
28    delete[] cspace;
29    delete[] size;
30  }
31
32  bool ConfigSpace::collidedState(RobotState &st)
33  {
34    // since C-Space is only 2-dimensional, only worry about first 2 state
        variables (X = [x1, x2, ...., xn]')
35    double x1, x2;
36    int d0, d1;    // d0 -> "row",  d1 -> "col"
37    st.get(0, x1);
38    st.get(1, x2);
39    // theta is stored in x1, and this is the "column" of the "image"
40    //d1 = static_cast<int>((x1 + _pi) * 100); // resolution is 0.01
41    d1 = (int)std::round1(x1 * 100) + 314;
42    // u is stored in x2, and this is the "row" of the "image"
43    //d0 = static_cast<int>((x2 + _pi) * 100); // resolution is 0.01
44    d0 = (int)std::round1(x2 * 100) + 314;
45
46    // size[1] -> COLS, size[0] -> ROWS
47    return (cspace[d0*size[1] + d1] == 1);
48  }
49
50  bool ConfigSpace::collidedEdge(RobotState &st1, RobotState &st2)
51  {
52    // Sample along the edge from both directions
53    const double DELTA = 0.001; // radians (~3 degrees)
54    double dist = st1.euclidDist(st2);
55    double cnt = 0.0;
56    // get the unit vector
57    RobotState v = ((st1) - (st2));
```

```
58    RobotState n = v / v.magnitude();
59
60    RobotState sample1 = (st1) - n*DELTA;
61    RobotState sample2 = (st2) + n*DELTA;
62    while (cnt <= dist / 2) {
63      if (collidedState(sample1) || collidedState(sample2)) {
64        return true;
65      }
66      sample1 = sample1 - n*DELTA;
67      sample2 = sample2 + n*DELTA;
68      cnt += DELTA;
69    }
70    return false;
71  }
72
73  int ConfigSpace::collisionPoints() const
74  {
75    if (cspace == nullptr) {
76      return -1;
77    }
78    int cnt = 0;
79    for (int i = 0; i < total_size; ++i) {
80      if (cspace[i] == 1) {
81        cnt++;
82      }
83    }
84    return cnt;
85  }
```

Listing 20: ConfigSpace Class source

```
1  #include "MotionPlanner.h"
```

```
#include "RobotCom.h"
#include "StopWatch.h"

std::ostream& operator<<(std::ostream& out, const RobotState& s);
void printActionPath(std::vector<RobotAction> actions, bool toFile);

int main(int argc, char* argv[]) {
  MotionPlanner& planner = MotionPlanner::getInstance();
  bool unitTesting = false;
  int seed = 101;
  //std::string configFileName("../RRTPlanner/config/config2a");
  //std::string configFileName("../RRTPlanner/config/config2b");
  std::string configFileName("../RRTPlanner/config/config1a");
  //std::string configFileName("../RRTPlanner/config/config1b");
  //std::string configFileName("../RRTPlanner/config/config1c");
  StopWatch<std::chrono::milliseconds> stopwatch;
  RobotCom comm(1011, "192.168.0.111", 50);
  std::vector<RobotAction> actions;
  std::ofstream csvfile, datfile;
  std::ifstream unittestfile;
  std::stringstream strm;
  int bytes, minNodes, lookForGoalAfter, nodeLimit;
  double delta, theta, thetaOpt, maxDist, minDist;
  bool success, run;
  MotionPlanner::RRTtype type;
  std::string output, buffer, fname;

  bytes = 0;
  theta = thetaOpt = delta = 0.0;
  if (unitTesting) {
    auto inc = [](MotionPlanner::RRTtype t) {
```

```
33      if (t == MotionPlanner::RRTtype::VERTEX) { return MotionPlanner::
    RRTtype::EXT_VERTEX; }
34      if (t == MotionPlanner::RRTtype::EXT_VERTEX)    { return
    MotionPlanner::RRTtype::EDGE; }
35      return MotionPlanner::RRTtype::VERTEX;
36     };
37     datfile.open("analysis/experiment_data.csv", std::ios::out | std::
    ios::app);
38     if (!datfile.is_open()) { std::cout << "ERROR!" << std::endl; }
39     else {
40
41       datfile << "seed,type,success,vlimit,goal_look,min_nodes,num_nodes
    ,num_misses,t_RRT,t_A*,pathNodes,pathDist,thetaDist,thetaOpt\n";
42       run = true;
43       type = MotionPlanner::RRTtype::VERTEX;
44       while (run) {
45         lookForGoalAfter = 50;
46         minNodes = 75;
47         nodeLimit = 150;
48         maxDist = 30.48;
49         minDist = 15.24;
50         // 20 loops
51         for (seed = 1111; seed < 1331; seed += 11) {
52           planner.configure(configFileName);
53           stopwatch.start();
54           success = planner.runRRT(type, maxDist, minDist,
    lookForGoalAfter, nodeLimit, minNodes);
55           stopwatch.stop();
56           auto rrtTime = stopwatch.readback();
57           stopwatch.save(std::string("rrt_time"));
58           if (success) {
```

```
59              stopwatch.start();
60              actions = planner.runAstar();
61              stopwatch.stop();
62              stopwatch.save(std::string("a*_time"));
63              //theta = getTotalRotation(actions);
64              planner.optimizeActions(actions);
65              //thetaOpt = getTotalRotation(actions);
66              //delta = getTotalTranslation(actions);
67            }
68            datfile << seed << "," << type << "," << success << "," <<
    nodeLimit << ",";
69            datfile << lookForGoalAfter << ',' << minNodes << ',';
70            datfile << planner.getNumV() << "," << planner.
    getNumMissedNodes() << ",";
71            datfile << rrtTime.count() << ",";
72            datfile << stopwatch.readback().count() << ",";
73            datfile << actions.size() << "," << delta << "," << theta << "
    ," << thetaOpt << std::endl;
74            planner.resetGraph();
75            actions.clear();
76
77          }
78          type = inc(type);
79          run = !(type == MotionPlanner::RRTtype::VERTEX);
80        }
81        datfile.close();
82      }
83
84    }
85    else {
86      auto radians = [](double deg)->double { const double _pi =
```

```
    3.14159265; return (deg / 180.0 * _pi); };
87   planner.configure(configFileName);
88
89   stopwatch.start();
90   planner.runRRT(MotionPlanner::RRTtype::EXT_VERTEX, radians(40),
    radians(10), 25, 100, 30);
91   stopwatch.stop();
92   stopwatch.save(std::string("rrt_time"));
93
94
95   stopwatch.start();
96   actions = planner.runAstar();
97   stopwatch.stop();
98   stopwatch.save(std::string("a*_time"));
99
100  planner.optimizeActions(actions);
101
102  output = planner.buildcsv();
103  strm.str(std::string()); strm.clear();
104  strm << "analysis/path.csv";
105  fname = strm.str();
106  csvfile.open(fname.c_str(), std::ios::out | std::ios::trunc);
107  if (!csvfile.is_open())
108    std::cout << "ERROR!" << std::endl;
109  else {
110    csvfile << output << std::endl;
111    csvfile.close();
112  }
113  /*csvfile.open("analysis/experiment_log.csv", std::ios::out | std::
    ios::app);
114  if (!csvfile.is_open()) { std::cout << "ERROR!" << std::endl; }
```

```
115     else {
116       csvfile << "Experiment,seed," << seed << ",configFile," <<
      configFileName << std::endl;
117       csvfile << "type,maxDistThresh,minDistThresh,nodesBeforeGoal,
      nodeLimit,minNodes" << std::endl;
118       csvfile << planner.getRRTdescriptor() << std::endl;
119       csvfile << "nodes," << planner.getNumV();
120       csvfile << ",misses," << planner.getNumMissedNodes() << std::endl;
121       csvfile << "time[ms],RRT," << stopwatch.recall(std::string("
      rrt_time")).count();
122       csvfile << ",A*," << stopwatch.recall(std::string("a*_time")).
      count() << std::endl;
123       csvfile << "file," << fname << std::endl << std::endl;
124       csvfile.close();
125     }*/
126     strm.str(std::string()); strm.clear();
127
128     //printActionPath(actions, true);
129     //comm.connectToRobot();
130
131     // send to the robot
132     if (comm.isConnected()) {
133       //strm << "init:0\n";
134       //bytes = comm.sendMsg(strm.str());
135       //if (bytes < 0) { std::cout << "error sending on init" << std::
      endl; getchar();  return 1; }
136       //strm.str(std::string()); strm.clear();
137       ////bytes = comm.recvMsg(buffer);
138       ////std::cout << buffer << std::endl;
139       //std::size_t i = 0;
140       //std::chrono::milliseconds tnow = stopwatch.readclock();
```

```
141       //std::chrono::milliseconds alarm = tnow + std::chrono::
    milliseconds(500);
142       //while (tnow < alarm) { tnow = stopwatch.readclock(); }
143       //while (bytes > 0 && i < actions.size()) {
144       //  // compose
145       //  tnow = stopwatch.readclock();
146       //  if (tnow > alarm) {
147       //    strm << "action:" << actions[i].getRotation1() << ",";
148       //    strm << actions[i].getTranslation() << ",";
149       //    strm << actions[i].getRotation2() << "\n";
150       //    std::cout << strm.str();
151       //    bytes = comm.sendMsg(strm.str());
152       //    if (bytes < 0) { std::cout << "error sending on index " << i
    << std::endl; break; }
153       //    buffer = (std::string());
154       //    bytes = comm.recvMsg(buffer);
155       //    if (bytes >= 0)
156       //      std::cout << buffer << " [" << bytes << "]" << std::endl;
157       //    strm.str(std::string()); strm.clear();
158       //    ++i;
159       //    alarm = tnow + std::chrono::milliseconds(1000);
160       //  }
161       //}
162       //comm.closeConnection();
163     }
164     else {
165       std::cout << "comm error" << std::endl;
166     }
167   }
168   std::cout << "complete..." << std::endl;
169   getchar();
```

```
170   return 0;
171 }
172
173 double getTotalTranslation(std::vector<RobotAction> actions) {
174   double delta = 0.0;
175   for (std::size_t i = 0; i < actions.size(); ++i) {
176     //delta += actions[i].getTranslation();
177   }
178   return delta;
179 }
180 double getTotalRotation(std::vector<RobotAction> actions) {
181   double theta = 0.0;
182   for (std::size_t i = 0; i < actions.size(); ++i) {
183     //theta += std::fabs(actions[i].getRotation1()) + std::fabs(actions[
      i].getRotation2());
184   }
185   return theta;
186 }
187
188 // print action vector
189 void printActionPath(std::vector<RobotAction> actions, bool toFile) {
190   std::stringstream strm;
191   std::ofstream file;
192   if (toFile) {
193     file.open("analysis/action_list.csv");
194     if (!file.is_open()) {
195       std::cout << "error opening file...using std::cout instead" << std
      ::endl;
196       toFile = false;
197     }
198   }
```

```
199   strm.str(std::string()); strm.clear();
200   if (toFile) {
201     for (std::size_t i = 0; i < actions.size(); ++i) {
202       //strm << actions[i].getRotation1() << ",";
203       //strm << actions[i].getTranslation() << ",";
204       //strm << actions[i].getRotation2() << "\n";
205       //file << strm.str();
206       //strm.str(std::string()); strm.clear();
207     }
208   }
209   else {
210     for (std::size_t i = 0; i < actions.size(); ++i) {
211       //strm << "action:" << actions[i].getRotation1() << ",";
212       //strm << actions[i].getTranslation() << ",";
213       //strm << actions[i].getRotation2() << "\n";
214       //std::cout << strm.str();
215       //strm.str(std::string()); strm.clear();
216     }
217   }
218   if (toFile) {
219     file.close();
220   }
221 }
```

Listing 21: Main Execution Function

```
1 #ifndef __MOTION_PLANNER_H__
2 #define __MOTION_PLANNER_H__
3 #include <boost/config.hpp>
4 #include <fstream>
5 #include <iostream>
6 #include <string>
```

```
7  #include <sstream>
8  #include <random>
9  #include <functional>
10 #include <utility>
11 #include <deque>
12 #include <chrono>
13
14 #include <boost/graph/adjacency_list.hpp>
15 #include <boost/graph/graph_utility.hpp>
16 #include <boost/property_map/property_map.hpp>
17 #include <boost/tuple/tuple.hpp>
18
19 #include "RobotAction.h"
20 #include "ConfigSpace.h"
21 #include "RobotState.h"
22 #include "PriorityQueue.h"
23
24 class RobotAction;
25 class ConfigSpace;
26 class RobotState;
27 class PrioityQueue;
28 class MotionPlanner;
29
30 namespace RRT {
31   const int DIM = 3;
32   struct VertexProperties {
33     std::size_t index;
34     RobotState *state;
35     double heuristic;
36     double cost;
37   };
```

```
38    struct EdgeProperties {
39      double dist;
40      RobotAction *action;
41    };
42    typedef boost::adjacency_list<boost::vecS, boost::listS, boost::
        directedS, VertexProperties, EdgeProperties> Graph;
43    typedef boost::graph_traits<Graph>::vertex_descriptor Vertex;
44    typedef boost::graph_traits<Graph>::edge_descriptor Edge;
45    typedef boost::graph_traits<Graph>::vertex_iterator V_iter;
46    typedef boost::graph_traits<Graph>::out_edge_iterator E_iter;
47    typedef std::map<std::pair<std::size_t, std::size_t>, RobotState>
        EdgeMeasured;
48  };
49
50
51
52  class MotionPlanner { /* Meyers Singleton Class */
53  public:
54    // Functor
55    class LowerCost {
56    public:
57      bool operator()(RRT::Vertex v1, RRT::Vertex v2) {
58        double tmp = MotionPlanner::getInstance().getCost(v1);
59        double tmp2 = MotionPlanner::getInstance().getCost(v2);
60        return (tmp <= tmp2);
61      }
62    };
63
64    enum RRTtype { VERTEX=0, EXT_VERTEX, EDGE };
65    static MotionPlanner& getInstance();
66    ~MotionPlanner();
```

```
67    void configure(std::string configfile);
68    std::string buildcsv();
69
70    MotionPlanner(const MotionPlanner&) = delete;
71    MotionPlanner& operator=(const MotionPlanner&) = delete;
72
73    double getCost(RRT::Vertex v) { if (configured) { return v_cost[v]; }
       else { return 0.0; } }
74    int getNumV() const { return numv; }
75    int getNumMissedNodes() const { return missedNodes; }
76    std::string getRRTdescriptor() const { return descriptor; }
77
78    bool runRRT(RRTtype type, double maxDistThresh, double minDistThresh,
79                int numBeforeLookForGoal, int vLimit, int minNodes);
80    void resetGraph();
81    std::vector<RobotAction> runAstar(/*params*/);
82    void optimizeActions(std::vector<RobotAction>& actions);
83
84  private:
85    MotionPlanner();
86
87    // Scenario parameters
88    RobotState startState;
89    RobotState goalState;
90    std::size_t goalId;
91    ConfigSpace *cSpace;
92
93    // Random generator params
94    std::random_device rd;
95    std::mt19937 gen;
96    std::uniform_real_distribution<> disTh;
```

```
97  std::uniform_real_distribution<> disU;
98  std::uniform_real_distribution<> disV;
99
100 // Booleans (state vars)
101 bool goalIsConnected;
102 bool failedToConnectToGoal;
103 bool inserted;
104
105 // Graph variables
106 RRT::Graph rrt;
107 std::string descriptor;
108 int numv;
109 int missedNodes;
110 bool configured;
111 RRT::Vertex vptr, vptr2, vstart;
112 RRT::Vertex src, tgt;
113 RRT::Edge eptr, eptr2, estart;
114 boost::property_map<RRT::Graph, std::size_t RRT::VertexProperties
    ::*>::type v_id;
115 boost::property_map<RRT::Graph, RobotState* RRT::VertexProperties
    ::*>::type v_state;
116 boost::property_map<RRT::Graph, double RRT::VertexProperties::*>::type
     v_heur;
117 boost::property_map<RRT::Graph, double RRT::VertexProperties::*>::type
     v_cost;
118 boost::property_map<RRT::Graph, double RRT::EdgeProperties::*>::type
    e_dist;
119 boost::property_map<RRT::Graph, RobotAction* RRT::EdgeProperties::*>::
    type e_action;
120 RRT::V_iter vi, viend;
121 RRT::E_iter ei, eiend;
```

```
122
123   // A* variables
124   bool* closedList;
125   PriorityQueue<RRT::Vertex, std::deque<RRT::Vertex>, LowerCost>
        openList;
126   PriorityQueue<RRT::Vertex, std::deque<RRT::Vertex>, LowerCost>::
        const_iterator pqPtr;
127   std::vector<RRT::Edge> path;
128   std::vector<RRT::Edge> pathOptimal;
129   std::vector<RRT::Edge>::const_iterator pathPtr;
130
131 // Methods
132 private:
133   double extVertDelta(double min_dist, double thresh);
134   friend class LowerCost;
135 };
136
137 #endif // !__MOTION_PLANNER_H__
```

Listing 22: MotionPlanner Class header

```
1  #include "MotionPlanner.h"
2
3  // Meyers Singleton
4  MotionPlanner& MotionPlanner::getInstance() {
5    static MotionPlanner instance;
6    return instance;
7  }
8  //disTh(-std::_Pi, std::_Pi),disU(-std::_Pi, std::_Pi),disV(-std::_Pi,
       std::_Pi),
9  MotionPlanner::MotionPlanner() :
10   startState(RRT::DIM),goalState(RRT::DIM),
```

```
11    goalId( -1 ),
12    cSpace(nullptr),
13    rd(), gen(),
14    disTh(-314,314),disU(-314, 314),disV(-314, 314),
15    configured(false),
16    goalIsConnected(false),
17    failedToConnectToGoal(false),
18    inserted(false),
19    rrt(), descriptor(), numv(0), missedNodes(0),
20    vptr(), vptr2(), vstart(),
21    src(), tgt(),
22    eptr(), eptr2(), estart(),
23    v_id(get(&RRT::VertexProperties::index, rrt)),
24    v_state(get(&RRT::VertexProperties::state, rrt)),
25    v_heur(get(&RRT::VertexProperties::heuristic, rrt)),
26    v_cost(get(&RRT::VertexProperties::cost, rrt)),
27    e_dist(get(&RRT::EdgeProperties::dist, rrt)),
28    e_action(get(&RRT::EdgeProperties::action, rrt)),
29    vi(), viend(),
30    ei(), eiend(),
31    closedList(nullptr),
32    openList(), pqPtr(),
33    path(), pathOptimal(), pathPtr()
34  {
35  }
36
37  MotionPlanner::~MotionPlanner() {
38    // Clean up
39    if (cSpace) {
40      delete cSpace;
41    }
```

```
42    if (closedList) {
43      delete[] closedList;
44    }
45    for (boost::tie(vi, viend) = boost::vertices(rrt); vi != viend; ++vi)
       {
46      for (boost::tie(ei, eiend) = boost::out_edges(*vi, rrt); ei != eiend
       ; ++ei) {
47        // check that edge not already deleted (v_id[0] will have edge
       0->3, so at v_id[3], edge already deleted)
48        if (e_action[*ei]) {
49          delete (e_action[*ei]);
50          e_action[*ei] = nullptr;
51        }
52      }
53      delete v_state[*vi];
54    }
55  }
56
57
58  bool MotionPlanner::runRRT( RRTtype type, double maxDistThresh, double
       minDistThresh,
59                              int numBeforeLookForGoal, int vLimit, int
       minNodes) {
60    auto discrete = [](double d)->double { int i = (int)d; return (double)
       i / 100.0; };
61    RobotState qr(3); // random node
62    RobotState qGoalLine(3);
63    RobotState vEdge(3);
64    bool collided, edgeUsed, usingGoal, goalLineConnected;
65    int iterations, indexToUse;
66    double min_dist;
```

```
67  double omega;
68
69  // create the descriptor
70  std::stringstream strm; strm.str(std::string()); strm.clear();
71  if (type == VERTEX)       { strm << "VERTEX";      }
72  else if (type == EDGE)    { strm << "EDGE";        }
73  else                      { strm << "EXT_VERTEX"; }
74  strm << "," << maxDistThresh << "," << minDistThresh << ",";
75  strm << numBeforeLookForGoal << "," << vLimit << "," << minNodes;
76  descriptor = strm.str();
77
78  // Add the start node to the graph
79  vstart = boost::add_vertex(rrt);
80  v_id[vstart] = 0;
81  v_state[vstart] = new RobotState(startState);
82  v_heur[vstart] = startState.dist(goalState);
83  v_cost[vstart] = 0.0;
84
85  goalLineConnected = false;
86
87  numv = 1;
88  iterations = missedNodes = 0;
89  while (numv <= vLimit) {
90    ++iterations;
91    if (iterations >= 3000000) return false;
92    // generate new random node
93    if (numv % numBeforeLookForGoal == 1 && !goalIsConnected) {
94      if (failedToConnectToGoal) {
95        double st[] = { discrete(disTh(gen)), discrete(disU(gen)),
    discrete(0) };
96        //double st[] = { disTh(gen), disU(gen), disV(gen) };
```

```
97          qr = RobotState(3, st);
98          usingGoal = false;
99          indexToUse = -1;
100       }
101       else {
102         qr = goalState;
103         usingGoal = true;
104         indexToUse = 0; // theta = 0, u = 1, v = 2
105       }
106     }
107     else {
108       double st[] = { discrete(disTh(gen)), discrete(disU(gen)),
    discrete(0) };
109       //double st[] = { disTh(gen), disU(gen), disV(gen) };
110       qr = RobotState(3, st);
111       failedToConnectToGoal = false;
112       usingGoal = false;
113       indexToUse = -1;
114     }
115     // check for qr collisions
116     collided = cSpace->collidedState(qr);
117     if (collided) { ++missedNodes; continue; }
118
119     // find closest vertex
120     min_dist = -1;
121     vptr2 = vstart;
122     edgeUsed = false;
123     for (boost::tie(vi, viend) = boost::vertices(rrt); vi != viend; ++vi
    )
124     {
125       double distanceV = qr.dist(v_state[*vi], indexToUse);
```

```
126       if (distanceV < min_dist || min_dist < 0) {
127         min_dist = distanceV;
128         vptr2 = *vi;
129         edgeUsed = false;
130       }
131       if (type == RRTtype::EDGE) {
132         std::pair<std::size_t, std::size_t> e1, e2;
133         RRT::EdgeMeasured emap;
134         RRT::EdgeMeasured::iterator pos;
135         // find closest edge
136         for (boost::tie(ei, eiend) = boost::out_edges(*vi, rrt); ei !=
    eiend; ++ei) {
137           // grab the edge (defined by the 2 vertex ids)
138           e1 = std::make_pair(v_id[boost::source(*ei, rrt)], v_id[boost
    ::target(*ei, rrt)]);
139           // grab its "mirror" ( <1,0> is same edge as <0,1> )
140           e2 = std::make_pair(e1.second, e1.first);
141
142           // check if the edge has already been used
143           if (emap.find(e1) == emap.end()) {
144             // check if the "mirror" of the edge has been used
145             if (emap.find(e2) == emap.end()) {
146               // insert the edge into the map
147               boost::tie(pos, inserted) = emap.insert(std::make_pair(e1,
     RobotState()));
148               // calculate the distance (normal intersect)
149               pos->second = qr.intersect(v_state[boost::source(*ei, rrt)
    ], v_state[boost::target(*ei, rrt)]);
150               double distanceE = qr.dist(pos->second);
151               if (pos->second.onTheLine(v_state[boost::source(*ei, rrt)
    ], v_state[boost::target(*ei, rrt)]))
```

```
152             {
153               if (distanceE < min_dist || min_dist < 0) {
154                 min_dist = distanceE;
155                 edgeUsed = true;
156                 vEdge = pos->second;
157                 eptr2 = *ei;
158               }
159             }
160           }
161           // means the mirror was found (edge used)
162         } // means the edge was found (edge used)
163
164       } // end looping for edges for node(i)
165     } /* RRTtype == EDGE */
166
167   } // end find closest distance for all nodes
168
169
170     // check if the new edge will pass through an obstacle (between *
    v_state[vptr2] and qr)
171   if (type == RRTtype::EDGE && edgeUsed) {
172     collided = cSpace->collidedEdge(vEdge, qr);
173   }
174   else {
175     // modifiy the desired node to the one that is one the goalLine
176     if (usingGoal && min_dist >= 0.005 && !goalLineConnected) {
177       qGoalLine = RobotState(*(v_state[vptr2]));
178       goalState.get(0, omega);
179       qGoalLine.set(0, omega);
180       usingGoal = false;
181       goalLineConnected = true;
```

```
182         qr = qGoalLine;
183       }
184       collided = cSpace->collidedEdge(*(v_state[vptr2]), qr);
185     }
186
187     if (collided) {
188       if (qr == goalState) {
189         // set flag to know try to select more random nodes
190         // otherwise will continuously try (and fail) to connect to the
    goal node
191         failedToConnectToGoal = true;
192       }
193       ++missedNodes;
194       continue;
195     }
196     if (usingGoal && (goalLineConnected || min_dist <= 0.001)) {
197       min_dist = goalState.dist(qGoalLine, -1);
198     }
199
200     // add qr to the graph if greater than mininum threshold distance
201     // if qr is the goal node, we don't care if it's "too" close
202     // this accounts for the case where we randomly picked a node that
    is basically the goal location
203     if (min_dist <= minDistThresh && !(qr == goalState)) { ++missedNodes
    ; continue; }
204
205     // make sure that if using edges, the new edges aren't less than the
     minimum distance
206     if (type == RRTtype::EDGE && edgeUsed && !(qr == goalState)) {
207       src = boost::source(eptr2, rrt);
208       tgt = boost::target(eptr2, rrt);
```

```
209       if ((v_state[src]->dist(vEdge) <= minDistThresh ||
210         v_state[tgt]->dist(vEdge) <= minDistThresh))
211       {
212         ++missedNodes; continue;
213       }
214     }
215     // if we aren't using edges but the new distance is "far", let's
    break it up into mulitple nodes/edges
216     // this can be faster for checking "closest" node, rather than using
     edges (at higher resource cost)
217     if (type == RRTtype::EXT_VERTEX && min_dist >= maxDistThresh) {
218       // Try to keep new distances less than 1ft (30.5 cm)
219       RobotState q = qr - (*v_state[vptr2]);
220       RobotState n = q / q.magnitude();
221       double delta = extVertDelta(min_dist, maxDistThresh);
222       double d = 0;
223       while (d < (min_dist - delta)) {
224         vptr = boost::add_vertex(rrt);
225         v_id[vptr] = numv++;
226         v_state[vptr] = new RobotState((*v_state[vptr2]) + n*delta);
227         v_heur[vptr] = v_state[vptr]->dist(goalState);
228         boost::tie(eptr, inserted) = boost::add_edge(vptr2, vptr, rrt);
229         if (inserted) {
230           e_dist[eptr] = delta;
231           e_action[eptr] = new RobotAction(v_state[vptr2], v_state[vptr
    ]);
232         }
233         vptr2 = vptr;
234         d += delta;
235       }
236       min_dist = delta;
```

```
237     } /* end RRTtype::EXT_VERTEX */
238
239     if (qr == goalState) {
240       // at this point, the goal can now be connected
241       // the loop will most likely still run, since having more nodes in
    the graph
242       // helps plan paths
243       goalIsConnected = true;
244       goalId = numv;
245     }
246     // add new node to the tree
247     vptr = boost::add_vertex(rrt);
248     v_id[vptr] = numv;
249     v_state[vptr] = new RobotState(qr);
250     v_heur[vptr] = qr.dist(goalState);
251
252     // check if edge or vertex used
253     if (type == RRTtype::EDGE && edgeUsed) {
254       ++numv;
255       vptr2 = boost::add_vertex(rrt);
256       v_id[vptr2] = numv;
257       v_state[vptr2] = new RobotState(vEdge);
258       v_heur[vptr2] = vEdge.dist(goalState);
259       // grab vertex end points
260       src = boost::source(eptr2, rrt);
261       tgt = boost::target(eptr2, rrt);
262       // remove old edge (and delete the action)
263       delete e_action[eptr2];
264       e_action[eptr2] = nullptr;
265       boost::remove_edge(eptr2, rrt);
266       // add new edges
```

```
267       boost::tie(eptr, inserted) = boost::add_edge(src, vptr2, rrt);
268       if (inserted) {
269         e_dist[eptr] = v_state[src]->dist(v_state[vptr2]);
270         e_action[eptr] = new RobotAction(v_state[src], v_state[vptr2]);
271       }
272       boost::tie(eptr, inserted) = boost::add_edge(vptr2, tgt, rrt);
273       if (inserted) {
274         e_dist[eptr] = v_state[vptr2]->dist(v_state[tgt]);
275         e_action[eptr] = new RobotAction(v_state[vptr2], v_state[tgt]);
276       }
277       boost::tie(eptr, inserted) = boost::add_edge(vptr2, vptr, rrt);
278       if (inserted) {
279         e_dist[eptr] = min_dist;
280         e_action[eptr] = new RobotAction(v_state[vptr2], v_state[vptr]);
281       }
282       // increment vLimit, since we want to add 10 new nodes, not
    including ones I added into edges
283       ++vLimit;
284     }
285     else {
286       boost::tie(eptr, inserted) = boost::add_edge(vptr2, vptr, rrt);
287       if (inserted) {
288         e_dist[eptr] = min_dist;
289         e_action[eptr] = new RobotAction(v_state[vptr2], v_state[vptr]);
290       }
291       else {
292         std::cout << "error" << std::endl;
293       }
294     }
295     ++numv;
296     if (goalIsConnected && numv >= minNodes) {
```

```
297         break;
298       } /* quit the loop */
299     }
300     return goalIsConnected;
301 }
302 
303 void MotionPlanner::resetGraph() {
304     goalId = -1;
305     goalIsConnected = false;
306     failedToConnectToGoal = false;
307     // Delete all nodes and edges (along with their properties)
308     int i = numv-1;
309     //while (i >= 0) {
310     //   for (boost::tie(ei, eiend) = boost::out_edges(boost::vertex(i, rrt
        ), rrt); ei != eiend; ++ei) {
311     //     if (e_action[*ei]) {
312     //       delete (e_action[*ei]);
313     //       e_action[*ei] = nullptr;
314     //     }
315     //   }
316     //   delete v_state[boost::vertex(i, rrt)];
317     //   v_state[boost::vertex(i, rrt)] = nullptr;
318     //   //boost::clear_vertex(boost::vertex(i, rrt), rrt);
319     //   //boost::remove_vertex(boost::vertex(i, rrt), rrt);
320     //   --i;
321     //}
322     rrt.clear();
323 }
324 
325 std::vector<RobotAction> MotionPlanner::runAstar() {
326     std::vector<RobotAction> actions;
```

```
327   pathOptimal.clear();
328   path.clear();
329   path.reserve(numv);
330   if (goalIsConnected) {
331     // initialize the closedList to empty
332     if (!closedList) { closedList = new bool[numv]; }
333     memset(closedList, 0, numv);
334
335     // initialize the openList with the start node
336     openList.push(vstart);
337     bool keepPlanning = true;
338     while (keepPlanning) {
339       vptr = openList.top();
340       openList.pop();
341       // add node to the closed list
342       closedList[v_id[vptr]] = true;
343       // for all neighbors
344       for (boost::tie(ei, eiend) = boost::out_edges(vptr, rrt); ei !=
    eiend; ++ei) {
345         tgt = boost::target(*ei, rrt);
346         std::size_t smacky = v_id[tgt];
347         // check if neighbor (tgt) is in the closed list
348         if (!closedList[v_id[tgt]]) {
349           // find cost of the neighbor
350           // f(n')   =   (    g(n)     +          g(n')
      ) +     h(n')
351           double cost = (v_cost[vptr] + v_state[vptr]->dist(v_state[tgt
    ])) + v_heur[tgt];
352
353           // check if tgt is currently in the openList
354           pqPtr = openList.find(tgt);
```

```
355         if (pqPtr == openList.end()) {
356           v_cost[tgt] = cost;
357           openList.push(tgt);
358           // add edge to the path
359           path.push_back(*ei);
360         }
361         else if (cost < v_cost[*pqPtr]) {
362           // delete the old edge in the path, since we found a better
    one
363           RRT::E_iter ei2, eiend2;
364           for (boost::tie(ei2, eiend2) = boost::out_edges(*pqPtr, rrt)
    ; ei2 != eiend2; ++ei2) {
365             pathPtr = std::find(path.begin(), path.end(), *ei2);
366             if (pathPtr != path.end()) {
367               path.erase(pathPtr);
368               break;
369             }
370           }
371           // update the cost & add new edge to the path
372           v_cost[*pqPtr] = cost;
373           path.push_back(*ei);
374         }
375       }
376     }
377     RobotState q = *v_state[openList.top()];
378     int id = v_id[openList.top()];
379     keepPlanning = !(*v_state[openList.top()] == goalState);
380   }
381
382   // Grab optimal path from the
383   int pathLen = 0;
```

```
384     std::size_t srcId, tgtId;
385     for (std::size_t i = path.size() - 1; i >= 0 && i < path.size(); --i
        ) {
386       srcId = (pathLen == 0) ? goalId : v_id[boost::source(pathOptimal[
        pathLen - 1], rrt)];
387       tgtId = v_id[boost::target(path[i], rrt)];
388       if (srcId == tgtId) {
389         pathOptimal.push_back(path[i]);
390         actions.push_back(*e_action[path[i]]);
391         ++pathLen;
392       }
393     }
394   }
395   openList.clear();
396   //while (!openList.empty())
397   //  openList.pop();
398   return actions;
399 }
400
401 void MotionPlanner::optimizeActions(std::vector<RobotAction>& actions) {
402   auto min_angle = [](double delta)->double {
403     if ((delta) > std::_Pi) return (delta - (2 * std::_Pi));
404     if ((delta) < -std::_Pi) return ((delta) + (2 * std::_Pi));
405     return (delta);
406   };
407   // reverse the path
408   RobotAction tmp;
409   std::size_t i = 0;
410   std::size_t j = actions.size() - 1;
411
412   while (i <= j) {
```

```
413       tmp = actions[i];
414       actions[i] = actions[j];
415       actions[j] = tmp;
416       if (j == 0) {
417         break;
418       }
419       ++i;
420       --j;
421     }
422
423     // optimize by removing the second rotation action
424     // assumptions: start node is first, goal node is last
425     /*for (i = 0; i < actions.size()-1; ++i) {
426       delta = actions[i].getRotation2() + actions[i + 1].getRotation1();
427       if (fabs(delta) <= epsilon) {
428         delta = 0.0;
429       }
430       actions[i].setRotation2(min_angle(delta));
431       actions[i + 1].setRotation1(0.0);
432     }*/
433
434 }
435
436 /* Config File Format
437   (1) seed
438   (2) dim
439   (3) c_space.conf
440   (4) start
441   (5) goal_cnt (2)
442   (6) goal_1
443   (7) goal_2
```

```
444 */
445 void MotionPlanner::configure(std::string configfile) {
446   FILE *fptr;
447   char buf[256];
448   char cspaceFname[80];
449   int seed;
450   int cspace_dim;
451   double st[3];
452   int goal_cnt;
453
454   fopen_s(&fptr, configfile.c_str(), "r");
455   if (fptr == NULL) {
456     std::cout << "Error! couldn't open file " << configfile  << std::
      endl;
457     return;
458   }
459   // seed
460   fgets(buf, 256, fptr);
461   sscanf_s(buf, "%d", &seed);
462   // dimension of cspace
463   fgets(buf, 256, fptr);
464   sscanf_s(buf, "%d", &cspace_dim);
465   // cspace file name
466   fgets(buf, 256, fptr);
467   sscanf_s(buf, "%s", cspaceFname, 80);
468   // start state
469   fgets(buf, 256, fptr);
470   sscanf_s(buf, "%lf %lf %lf", st + 0, st + 1, st + 2);
471   startState = RobotState(3, st);
472   // number of goals
473   fgets(buf, 256, fptr);
```

```
474   sscanf_s(buf, "%d", &goal_cnt);
475   // goal state: 1
476   fgets(buf, 256, fptr);
477   sscanf_s(buf, "%lf %lf %lf", st + 0, st + 1, st + 2);
478   goalState = RobotState(3, st);
479   // goal state: 2
480   //fgets(buf, 256, fptr);
481   //sscanf_s(buf, "%lf %lf %lf", st + 0, st + 1, st + 2);
482   //goalState2 = RobotState(dim, st);
483   fclose(fptr);
484
485   // Set up the random number generator (re-seed it)
486   if (seed <= -1) { gen.seed(rd());   }
487   else            { gen.seed(seed); }
488
489   if (cSpace) {
490     delete cSpace;
491     cSpace = nullptr;
492   }
493   cSpace = new ConfigSpace(std::string(cspaceFname), cspace_dim);
494
495   configured = true;
496 }
497
498 std::string MotionPlanner::buildcsv() {
499   std::stringstream csvfile;
500   csvfile.str(std::string());
501   csvfile.clear();
502   double x[3];
503   // Output the tree
504   csvfile << "Id, state.th, state.u, state.v, e_dist, e_id," << std::
```

```
    endl;
505   for (boost::tie(vi, viend) = boost::vertices(rrt); vi != viend; ++vi)
    {
506     v_state[*vi]->get(0, x[0]);
507     v_state[*vi]->get(1, x[1]);
508     v_state[*vi]->get(2, x[2]);
509     csvfile << v_id[*vi] << " , " << x[0] << " , " << x[1] << " , " << x
    [2] << " , ";
510     for (boost::tie(ei, eiend) = boost::out_edges(*vi, rrt); ei != eiend
    ; ++ei) {
511       csvfile << e_dist[*ei] << " , " << v_id[boost::target(*ei, rrt)]
    << " , ";
512     }
513     csvfile << std::endl;
514   }
515   // Output the path (in nodes)
516   for (std::size_t i = pathOptimal.size() - 1; i >= 0 && i < pathOptimal
    .size(); --i) {
517     csvfile << v_id[boost::source(pathOptimal[i], rrt)] << " , ";
518   }
519   if (pathOptimal.size()) {
520     csvfile << v_id[boost::target(pathOptimal[0], rrt)] << std::endl;
521   }
522   // Output the path (in actions)
523   // TODO
524
525   return csvfile.str();
526 }
527
528 // Private methods
529 double MotionPlanner::extVertDelta(double min_dist, double thresh) {
```

```
  double delta;
  if (min_dist >= 5 * thresh)      { delta = min_dist / 6; }
  else if (min_dist >= 4 * thresh) { delta = min_dist / 5; }
  else if (min_dist >= 3 * thresh) { delta = min_dist / 4; }
  else if (min_dist >= 2 * thresh) { delta = min_dist / 3; }
  else { delta = min_dist / 2; }
  return delta;
}
```

Listing 23: MotionPlanner Class source

```
#ifndef __PRIORITY_QUEUE_H__
#define __PRIORITY_QUEUE_H__

#include <queue>
template<
  class T,
  class Container = std::vector<T>,
  class Compare = std::less<typename Container::value_type>
> class PriorityQueue : public std::priority_queue<T, Container, Compare
    >
{
public:
  typedef typename
    std::priority_queue<
    T,
    Container,
    Compare>::container_type::const_iterator const_iterator;

  const_iterator find(const T&val) const {
    auto first = this->c.cbegin();
    auto last = this->c.cend();
```

```
21      while (first != last) {
22        if (*first == val) return first;
23        ++first;
24      }
25      return last;
26    }
27    const_iterator end() const {
28      return this->c.cend();
29    }
30    void clear() {
31      while (!this->empty()) {
32        this->pop();
33      }
34    }
35  };
36
37  #endif/*__PRIORITY_QUEUE_H__*/
```

Listing 24: PriorityQueue Class header

```
1  #ifndef __ROBOT_ACTION_H__
2  #define __ROBOT_ACTION_H__
3
4  #include <random>
5  #include "RobotState.h"
6  #include <algorithm>
7
8  class RobotAction {
9  public:
10   RobotAction();
11   RobotAction(int d);
12   RobotAction(int d, double *act);
```

```
13    RobotAction(const RobotAction& act);
14    RobotAction(const RobotState* s1, const RobotState* s2);
15
16    ~RobotAction();
17
18    // Operator Overloads
19    RobotAction& operator=(const RobotAction& act)
20    {
21      if (dim == act.dim) {
22        memcpy(action, act.action, dim * sizeof(double));
23      }
24      return (*this);
25    }
26
27    int get(int idx, double &val) const;
28    int set(int idx, double  val);
29
30  private:
31    int dim;
32    double *action;
33  };
34  #endif // !__ROBOT_ACTION_H__
```

Listing 25: RobotAction Class header

```
1  #include "RobotAction.h"
2  #include <random>
3  #include <algorithm>
4  #include <functional>
5
6  RobotAction::RobotAction() : dim(0), action(nullptr)
7  {
```

```
8
9  }
10 RobotAction::RobotAction(int d) : dim(d), action(new double[dim])
11 {
12   std::memset(action, 0, dim * sizeof(double));
13 }
14 RobotAction::RobotAction(int d, double *act) : dim(d), action(new double
       [dim])
15 {
16   std::memcpy(action, act, dim*sizeof(double));
17 }
18 RobotAction::RobotAction(const RobotAction& act) : dim(act.dim), action(
       new double[dim])
19 {
20   std::memcpy(action, act.action, dim * sizeof(double));
21 }
22
23 RobotAction::RobotAction(const RobotState* s1, const RobotState* s2) :
       RobotAction(s1->dimension()) {
24   const double epsilon = 0.0001;
25   double src, dst;
26   auto min_angle = [](double dst, double src)->double {
27     if ((dst - src) > std::_Pi) return ((dst - src) - (2 * std::_Pi));
28     if ((dst - src) < -std::_Pi) return ((dst - src) + (2 * std::_Pi));
29     if ((dst - src) == std::_Pi || (dst - src) == -std::_Pi) return (0);
30     return (dst - src);
31   };
32   for (int i = 0; i < dim; ++i) {
33     (void)s1->get(i, src);
34     (void)s2->get(i, dst);
35     action[i] = min_angle(dst, src);
```

```
  }
}

RobotAction::~RobotAction()
{
  if (action != nullptr) {
    delete[] action;
  }
}
```

Listing 26: RobotAction Class source

```
#ifndef __ROBOT_COM_H__
#define __ROBOT_COM_H__
#include <winsock.h>
#include <string>

class RobotCom {
public:
  RobotCom(int port, std::string ip, std::size_t len) :
    _socket(), connected(false), portNo(port), ipAddr(ip),
    msgLen(len), buffer(new char[msgLen]) { }

  ~RobotCom() {
    if (connected) { closeConnection(); }
    delete[] buffer;
  }
  // unused (so restrict)
  RobotCom() = delete;
  RobotCom(const RobotCom&) = delete;
  RobotCom& operator=(const RobotCom&) = delete;

```

```
21
22    bool isConnected() const { return connected; }
23    int getPortNumber() const { return portNo; }
24    std::string getIpAddr() const { return ipAddr; }
25
26    bool connectToRobot();
27    void closeConnection();
28    int sendMsg(const std::string msg);
29    int recvMsg(std::string& msg);
30
31  private:
32    SOCKET _socket;
33    bool connected;
34    int portNo;
35    std::string ipAddr;
36    const int msgLen;
37    char* buffer;
38  };
39
40  #endif // !__ROBOT_COM_H__
```

Listing 27: RobotCom Class header

```
1  #include "RobotCom.h"
2
3  bool RobotCom::connectToRobot() {
4    // start up Winsock...
5    WSADATA wsadata;
6
7    int error = WSAStartup(0x0202, &wsadata);
8
9    if (error) {
```

```
10      connected = false;
11      return false;
12    }
13    // verify version
14    if (wsadata.wVersion != 0x0202) {
15      WSACleanup();
16      connected = false;
17      return false;
18    }
19
20    SOCKADDR_IN target;
21    target.sin_family = AF_INET;
22    target.sin_port = htons(portNo);
23    target.sin_addr.s_addr = inet_addr(ipAddr.c_str());
24
25    _socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);  // create socket
26    if (_socket == INVALID_SOCKET) {
27      connected = false;
28      return false;
29    }
30
31    // try connecting...
32    connected = !(connect(_socket, (SOCKADDR*)&target, sizeof(target)) ==
       SOCKET_ERROR);
33    return connected;
34  }
35
36  void RobotCom::closeConnection() {
37    if (_socket) {
38      closesocket(_socket);
39    }
```

```
40    WSACleanup();
41  }
42
43  int RobotCom::sendMsg(const std::string msg) {
44    int bytes = -1;
45    strcpy_s(buffer, msgLen, msg.c_str());
46    if (connected) {
47      bytes = send(_socket, buffer, strlen(buffer), 0);
48      if (bytes == SOCKET_ERROR)
49        return WSAGetLastError();
50    }
51    return bytes;
52  }
53
54  int RobotCom::recvMsg(std::string& msg) {
55    int bytes = recv(_socket, buffer, msgLen, 0);
56    if (bytes < 0)
57      return WSAGetLastError();
58
59    // return by reference
60    msg = (buffer);
61    return bytes;
62  }
```

Listing 28: RobotCom Class source

```
1  #ifndef __ROBOT_STATE_H__
2  #define __ROBOT_STATE_H__
3
4  #include <iostream>
5
6  class RobotState {
```

```
public:
  RobotState();
  RobotState(int d);
  RobotState(int d, double *st);
  RobotState(const RobotState& st);
  ~RobotState();

  // Operator Overloads
  RobotState& operator=(const RobotState& st)
  {
    if (dim == st.dim) {
      memcpy(state, st.state, dim * sizeof(double));
    }
    return (*this);
  }

  RobotState operator-(const RobotState& st) const;
  RobotState operator+(const RobotState& st) const;
  RobotState operator*(double a) const;
  RobotState operator/(double a) const;
  bool operator==(const RobotState& st) const;

  // Get/Set functions
  int get(int idx, double &val) const;
  int set(int idx, double  val);
  int dimension() const { return dim;  }

  // Other operations
  double dot(const RobotState& st) const;
  double magnitude() const;

```

```
38   // distance function
39   RobotState intersect(const RobotState* st1, const RobotState* st2)
       const;
40   bool onTheLine(const RobotState* st1, const RobotState* st2) const;
41   // distance function
42   double dist(const RobotState* st, int indexToUse = -1) const;
43   double dist(const RobotState st, int indexToUse = -1) const;
44   double euclidDist(const RobotState st) const;
45
46 private:
47   const double _pi = 3.1415926535897;
48   int dim;
49   double *state;
50   double min_angle(double dst, double src) const;
51
52   // implementation specific
53   int weights(double *a) const;
54 };
55
56 #endif // !__ROBOT_STATE_H__
```

Listing 29: RobotState Class header

```
1 #include "RobotState.h"
2 #include <algorithm>
3
4 // Constructors
5 RobotState::RobotState() : dim(2), state(new double[dim])
6 {
7   std::memset(state, 0, dim * sizeof(double));
8 }
9 RobotState::RobotState(int d) : dim(d), state(new double[dim])
```

```
10 {
11   std::memset(state, 0, dim * sizeof(double));
12 }
13 RobotState::RobotState(int d, double *st) : dim(d), state(new double[dim
       ])
14 {
15   std::memcpy(state, st, dim * sizeof(double));
16 }
17 RobotState::RobotState(const RobotState& st) : dim(st.dim), state(new
       double[dim])
18 {
19   std::memcpy(state, st.state, dim * sizeof(double));
20 }
21
22 // Destructor
23 RobotState::~RobotState()
24 {
25   if (state != nullptr) {
26     delete[] state;
27   }
28 }
29
30 int RobotState::get(int idx, double &val) const
31 {
32   if (idx < 0 || idx >= dim) {
33     return -1;
34   }
35   val = state[idx];
36   return 0;
37 }
38
```

```
39 int RobotState::set(int idx, double val)
40 {
41   if (idx < 0 || idx >= dim) {
42     return -1;
43   }
44   state[idx] = val;
45   return 0;
46 }
47
48 // Operator Overloads
49 bool RobotState::operator==(const RobotState& st) const
50 {
51   if (dim != st.dim) {
52     return false;
53   }
54   for (int i = 0; i < dim; ++i) {
55     if (state[i] != st.state[i]) {
56       return false;
57     }
58   }
59   return true;
60 }
61
62 RobotState RobotState::operator-(const RobotState& st) const
63 {
64   RobotState rs(dim);
65   for (int i = 0; i < dim; ++i) {
66     rs.state[i] = this->state[i] - st.state[i];
67   }
68   return rs;
69 }
```

```
70 RobotState RobotState::operator+(const RobotState& st) const
71 {
72   RobotState rs(dim);
73   for (int i = 0; i < dim; ++i) {
74     rs.state[i] = this->state[i] + st.state[i];
75   }
76   return rs;
77 }
78 RobotState RobotState::operator*(double a) const
79 {
80   RobotState rs(dim);
81   for (int i = 0; i < dim; ++i) {
82     rs.state[i] = this->state[i] * a;
83   }
84   return rs;
85 }
86 RobotState RobotState::operator/(double a) const
87 {
88   RobotState rs(dim);
89   for (int i = 0; i < dim; ++i) {
90     rs.state[i] = this->state[i] / a;
91   }
92   return rs;
93 }
94
95 // Other operations
96 double RobotState::dot(const RobotState& st) const
97 {
98   double result = 0.0;
99   for (int i = 0; i < dim; ++i) {
100    result += this->state[i] * st.state[i];
```

```
101   }
102   return result;
103 }
104
105 double RobotState::magnitude() const
106 {
107   double result = 0.0;
108   for (int i = 0; i < dim; ++i) {
109     result += this->state[i] * this->state[i];
110   }
111   return std::sqrt(result);
112 }
113
114 // return the the node that is the projection onto the line segment
      defined by 2 node parameters
115 RobotState RobotState::intersect(const RobotState* st1, const RobotState
      * st2) const
116 {
117   return RobotState(dim);
118 }
119 bool RobotState::onTheLine(const RobotState* st1, const RobotState* st2)
       const
120 {
121   return false;
122 }
123
124 // distance function
125 double RobotState::dist(const RobotState *st, int indexToUse) const
126 {
127   auto SQR = [](double x)->double { return x*x;  };
128   // implementation specific
```

```
129   double *a = new double[dim];
130   double tmp;
131   // set the values of a (implementation specific)
132   (void)weights(a);
133   double d = 0;
134   if (indexToUse >= 0) {
135     (void)st->get(indexToUse, tmp);
136     d = a[indexToUse] * SQR(state[indexToUse] - tmp);
137   }
138   else {
139     for (int i = 0; i < dim; ++i) {
140       (void)st->get(i, tmp);
141       d += a[i] * SQR(state[i] - tmp);
142     }
143   }
144
145   delete[] a;
146   return std::sqrt(d);
147 }
148 double RobotState::dist(const RobotState st, int indexToUse) const
149 {
150   auto SQR = [](double x)->double { return x*x;  };
151   // implementation specific
152   double *a = new double[dim];
153   double tmp;
154   // set the values of a (implementation specific)
155   (void)weights(a);
156   double d = 0;
157   if (indexToUse >= 0) {
158     (void)st.get(indexToUse, tmp);
159     d = a[indexToUse] * SQR(state[indexToUse] - tmp);
```

```
160   }
161   else {
162     for (int i = 0; i < dim; ++i) {
163       (void)st.get(i, tmp);
164       d += a[i] * SQR(state[i] - tmp);
165     }
166   }
167 
168   delete[] a;
169   return std::sqrt(d);
170 }
171 double RobotState::euclidDist(const RobotState st) const
172 {
173   auto SQR = [](double x)->double { return x*x;  };
174   double tmp, d;
175   d = 0;
176   for (int i = 0; i < dim; ++i) {
177     (void)st.get(i, tmp);
178     d += SQR(state[i] - tmp);
179   }
180   return std::sqrt(d);
181 }
182 double RobotState::min_angle(double dst, double src) const
183 {
184   if ((dst - src) > _pi) return ((dst - src) - (2 * _pi));
185   if ((dst - src) < -_pi) return ((dst - src) + (2 * _pi));
186   return (dst - src);
187 }
188 
189 int RobotState::weights(double *a) const
190 {
```

```
191     // dim=3;
192     a[0] = 0.3;
193     a[1] = 0.7;
194     a[2] = 0.0;
195     return 1;
196 }
```

Listing 30: RobotState Class source

Appendix E Continuum Element Simulation Software

```
function [L] = allen_lengths( X )
    d = 0.0438;
    alpha = d*sqrt(3)/2;
    A = [ 0,  -2,   2/(d*sqrt(3)), 0;
          1,   1,   2/(d*sqrt(3)), 0;
         -1,   1,   2/(d*sqrt(3)), 0;
          0,   0,              0, 1
        ];
    inv(A)
    l = (alpha*A*X);
    l1 = l(1); l2 = l(2); l3 = l(3);
end
```

Listing 31: Function to retrieve tendon lengths from configuration variables

```
function col = check_collision(arm, off, gripper, all_obs)
col = false;
X = 1; Y = 2; Z = 3;
rad = 0.025;
[r,~] = size(arm);
c = max(size(all_obs));
% for each obstacle
for j = 1:c
    obs = all_obs{j};
    % check gripper
    for g = 1:3
        grp = gripper{g};
        col = is_inside(grp.p, obs);
        if (col)
            return;
```

```
        end
        col = is_inside(obs.p, grp);
        if (col)
            return;
        end
    end
    % check along the backbone, so inflate by backbone radius
    obs.w = obs.w + 2*rad;
    obs.l = obs.l + 2*rad;
    obs.h = obs.h + 2*rad;
    col = is_inside(arm + [zeros(r,1), zeros(r,1), off*ones(r,1)], obs);
    if (col)
        return;
    end
end
end
```

Listing 32: Function to check for collisions

```
function execute_timestep_ind(obj, ~)

S = 1; U = 2; V = 3; W = 4;
data = get(obj, 'UserData');
data.t = data.t + data.tau;
data.t;
% data has the following properties
% t       :
% tau     :
% state   : [s, u, v, w]
% setpt   : [s, u ,v, w]
% K       : [kp, kd, ki]
% ravg    : [ras, rau, rav, raw]
```

```
% samples :
% elast   : [es, eu, ev, ew]
s_0 = 0.95; d = 0.022;
u = data.state(U); v = data.state(V); s = data.state(S);
l = [ s_0 + (-2)*d*v ;
      s_0 + ( 2)*d*u ;
      s_0 + ( 2)*d*v ;
      s_0 + (-2)*d*u
    ];
u = data.setpt(U); v = data.setpt(V); s = data.setpt(S);
l_set = [ s_0 + (-2)*d*v ;
      s_0 + ( 2)*d*u ;
      s_0 + ( 2)*d*v ;
      s_0 + (-2)*d*u
    ];
out = zeros(1,4);
err = l_set - l;
for i = 1:4
    data.ravg(i) = (data.ravg(i)*data.samples+err(i))/(data.samples+1);
    out(i) = err(i) * data.K(1) + ...
             (err(i) - data.elast(i))/data.t * data.K(2) + ...
             (data.ravg(i)) * data.K(3);
    % saturate
    if (out(i) > 12)
        out(i) = 12;
    elseif (out(i) < -12)
        out(i) = -12;
    end
end

%% simulate
```

```
l_next = 1 + (out)'/12*(32/60)*data.t*0.44*pi;

state_next = (1/(4*d)) * [-1, 0, 1, 0; 0, 1, 0, -1; d, d, d, d] * l_next
    ;

data.state = [state_next(3), state_next(1), state_next(2), data.state(W)
    ];
data.samples = data.samples + 1;
% if (data.samples > 20)
%     data.samples = 20;
% end

obj.UserData = data;
```

Listing 33: Function that executes a time-step in simulation

```
clear; clc;

W = 1; U = 2; V = 3;
v = 0.000001; s  = 1.0;
idx = 1;
step = 0.01;
points = [];
start = tic;
for w = -pi:step:pi
    for u = -pi:step:pi
        collision = lamp_sim(s, u, v, w);
        [r,~] = size(collision);
        if ( r > 0 )
            points(idx, :) = collision;
            idx = idx + 1;
        end
```

```
    end
end
t = toc(start)
% figure(1);
% hold on
% axis([-pi pi -pi pi -pi pi]);
% xlabel('Omega')
% ylabel('U')
% [r,c] = size(points);
% for i = 1:r
% plot3(points(i,W), points(i,U), points(i, V), 'ko', 'MarkerSize', 1, '
    MarkerFaceColor', [0 0 0]);
% end
```

Listing 34: Script that explores the space to generate the occupancy map

```
clear;clc;

%load('cspace_shelf_grp.mat');
load('cspace_shelf_grp2.mat');
cspace = uint8(zeros(629,629,629));
[len,~] = size(points);

for i=1:len
    r = int32((points(i,1) + pi) * 100)+1;
    c = int32((points(i,2) + pi) * 100)+1;
    d = int32((points(i,3) + pi) * 0) + 1;      % v should always be 0
    (1)

    cspace(r,c,d) = 1;
end

```

```
img = uint8(cspace(:,:,1)*255);
imshow(img);

fid = fopen('cspace2.conf','w');
fprintf(fid, '2 629 629\n');
fwrite(fid, uint8(cspace(:,:,1)));
fclose(fid);

fid = fopen('cspace2.ppm','w');
fprintf(fid, 'P5 629 629 255\n');
fwrite(fid, img);
fclose(fid);
```

Listing 35: Script that generates the occupancy map file

```
function b = is_inside( points, obj )
xyz = [false false false];
X = 1; Y = 2; Z = 3;
upperb = zeros(3);
lowerb = zeros(3);

for i = X:Z
    upperb(i) = max(obj.p(:,i));
    lowerb(i) = min(obj.p(:,i));
end

[r,~] = size(points);
for i = 1:r
    for j = X:Z
        xyz(j) = (points(i,j) >= lowerb(j) && points(i,j) <= upperb(j));
    end
    b = xyz(X) && xyz(Y) && xyz(Z);
```

```
18      if (b)
19          break;
20      end
21  end
22
23
24  end
```

Listing 36: Function that checks if a point is inside a polygon

```
1  function varargout = KinematicGui(varargin)
2  % KINEMATICGUI MATLAB code for KinematicGui.fig
3  %      KINEMATICGUI, by itself, creates a new KINEMATICGUI or raises the
       existing
4  %      singleton*.
5  %
6  %      H = KINEMATICGUI returns the handle to a new KINEMATICGUI or the
       handle to
7  %      the existing singleton*.
8  %
9  %      KINEMATICGUI('CALLBACK',hObject,eventData,handles,...) calls the
       local
10 %      function named CALLBACK in KINEMATICGUI.M with the given input
       arguments.
11 %
12 %      KINEMATICGUI('Property','Value',...) creates a new KINEMATICGUI
       or raises the
13 %      existing singleton*.  Starting from the left, property value
       pairs are
14 %      applied to the GUI before KinematicGui_OpeningFcn gets called.
       An
15 %      unrecognized property name or invalid value makes property
```

```
    application
%       stop.  All inputs are passed to KinematicGui_OpeningFcn via
    varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
    one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help KinematicGui

% Last Modified by GUIDE v2.5 13-Feb-2019 13:58:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @KinematicGui_OpeningFcn, ...
                   'gui_OutputFcn',  @KinematicGui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

```
44 % End initialization code - DO NOT EDIT
45
46 % --- Executes just before KinematicGui is made visible.
47 function KinematicGui_OpeningFcn(hObject, eventdata, handles, varargin)
48 % This function has no output args, see OutputFcn.
49 % hObject    handle to figure
50 % eventdata  reserved - to be defined in a future version of MATLAB
51 % handles    structure with handles and user data (see GUIDATA)
52 % varargin   command line arguments to KinematicGui (see VARARGIN)
53
54 % Choose default command line output for KinematicGui
55 handles.output = hObject;
56 handles.S = 1;
57 handles.U = 2;
58 handles.V = 3;
59 handles.W = 4;
60 handles.X = 6;
61 handles.Y = 7;
62 handles.Z = 8;
63
64 handles.UV = 3;
65 handles.UW = 2;
66 handles.VW = 1;
67
68 handles.state = [1.03, 0.0001, 0.0001, 0];
69 handles.rigid = 0;
70
71 handles.delta = 2 * (pi / 180);
72 handles.max_angle = 4*pi;
73 handles.min_angle = -4*pi;
74
```

```
75 handles.cpoints = zeros(1,4);
76 handles.collided = false;
77
78 handles.sim = false;
79 handles.node = false;
80 % Update handles structure
81 guidata(hObject, handles);
82
83 set(handles.axesLamp, 'Units', 'pixels', 'Position',
       [-1275,-1250,3000,3000]);
84 set(handles.axesCspace3, 'Units', 'pixels', 'Position', [-850, -1250,
       3000, 3000]);
85 set(handles.axesCspace, 'Units', 'pixels', 'Position', [-475, -1250,
       3000, 3000]);
86
87
88 % decent values....
89 %set(handles.axesLamp, 'Units', 'pixels', 'Position',
       [-1150,-1200,3000,3000]);
90 %set(handles.axesCspace, 'Units', 'pixels', 'Position', [-450, -950,
       3000, 3000]);
91 %set(handles.axesCspace3, 'Units', 'pixels', 'Position', [-450, -1300,
       3000, 3000]);
92
93 axes(handles.axesLamp);
94 rotate3d(handles.axesLamp, 'on');
95
96 axes(handles.axesCspace);
97
98 axes(handles.axesCspace3);
99 %rotate3d(handles.axesCspace3, 'on');
```

```
100
101 view(handles.axesCspace,0,-90)
102 view(handles.axesCspace3,30,22)
103 view(handles.axesLamp,-46,22)
104
105 if (handles.rigid == 0)
106     Lamp_Sim_GUI(1.03, 0.0001, 0.0001, 0, handles);
107 else
108     rigidLamp_Sim_GUI(1.03, 0.0001, 0.0001, 0, handles);
109 end
110 handles.configTable.Data(handles.S) = {1.03};
111 handles.configTable.Data(handles.U) = {0.0001};
112 handles.configTable.Data(handles.V) = {0.0001};
113 handles.configTable.Data(handles.W) = {0.0000};
114
115 updatePlot(handles);
116
117 %set(handles.rotThVal,'string',0.0000)
118 %set(handles.rotUVal,'string',0.0001)
119 %set(handles.rotVVal,'string',0.0001)
120
121
122 % UIWAIT makes KinematicGui wait for user response (see UIRESUME)
123 % uiwait(handles.figure1);
124
125
126 % --- Outputs from this function are returned to the command line.
127 function varargout = KinematicGui_OutputFcn(hObject, eventdata, handles)
128 % varargout  cell array for returning output args (see VARARGOUT);
129 % hObject    handle to figure
130 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
131 % handles    structure with handles and user data (see GUIDATA)
132
133 % Get default command line output from handles structure
134 varargout{1} = handles.output;
135
136 % --- Executes on button press in viewPoint.
137 function viewPoint_Callback(hObject, eventdata, handles)
138 % hObject    handle to viewPoint (see GCBO)
139 % eventdata  reserved - to be defined in a future version of MATLAB
140 % handles    structure with handles and user data (see GUIDATA)
141 v_en=get(handles.viewPoint,'Value');
142 [az, el] = view;
143 if v_en == 1
144    omega = handles.configTable.Data{handles.W};
145    view_az = 180*(omega)/(pi)-30;
146    view(handles.axesLamp,view_az,30);
147    %rotate3d off;
148 else
149     rotate3d(handles.axesLamp);
150 end
151 % Hint: get(hObject,'Value') returns toggle state of viewPoint
152
153
154 function assignValuesAllen(handles)
155
156 s = handles.configTable.Data{handles.S};
157 u = handles.configTable.Data{handles.U};
158 v = handles.configTable.Data{handles.V};
159 w = handles.configTable.Data{handles.W};
160
161 cla(handles.axesLamp)
```

```
%handles.collided = Lamp_Sim_GUI(s, u, v, w, handles);
if (handles.rigid == 0)
    handles.collided = Lamp_Sim_GUI(s, u, v, w, handles);
else
    handles.collided = rigidLamp_Sim_GUI(s, u, v, w, handles);
end
handles.collided = false;
updatePlot(handles);



function updatePlot(handles)
u = handles.configTable.Data{handles.U};
v = handles.configTable.Data{handles.V};
w = handles.configTable.Data{handles.W};
i = getDrawPlane(handles);

% 3D Plot
cla(handles.axesCspace3)
axis(handles.axesCspace3, 18*[-1, 1, -1, 1, -1, 1], 'square') % set axis
    to square
plot_torus(u, v, w, handles.axesCspace3);
%plot_donut(handles.axesCspace3, u, v, w, i);
axis(handles.axesCspace3, 18*[-1, 1, -1, 1, -1, 1], 'square') % set axis
    to square
set(handles.axesCspace3, 'XTick', [], 'YTick', [], 'ZTick', []);
% 2D Plot
fig = handles.axesCspace;
if (~handles.sim)
    cla(fig)
end
```

```
191 hold(fig,'on')
192 grid(fig, 'on')
193 axis(fig,1.75*[-pi pi -pi pi -pi pi], 'square')
194 if (i == handles.UW)
195     axis(fig,1.75*[-pi pi -pi pi -pi pi], 'square')
196     if (handles.sim)
197         if (handles.node)
198             plot3(fig, w, u, v, 'ko', 'MarkerSize', 3, 'MarkerFaceColor'
    , [1.0, 1.0, 0.0]);
199         else
200             plot3(fig, w, u, v, 'ko', 'MarkerSize', 1, 'MarkerFaceColor'
    , [0.0, 0.0, 0.0]);
201         end
202     else
203     plot3(fig, w, u, v, 'ko', 'MarkerSize', 3, 'MarkerFaceColor', [0.0,
    0.0, 0.0]);
204     end
205     xlabel(fig,'\omega')
206     ylabel(fig,'u')
207 elseif (i == handles.UV)
208     axis(fig,1.75*[-pi pi -pi pi -pi pi], 'square')
209     if (handles.sim)
210         if (handles.node)
211             plot3(fig, v, u, w, 'ko', 'MarkerSize', 3, 'MarkerFaceColor'
    , [1.0, 1.0, 0.0]);
212         else
213             plot3(fig, v, u, w, 'ko', 'MarkerSize', 1, 'MarkerFaceColor'
    , [0.0, 0.0, 0.0]);
214         end
215     else
216     plot3(fig, v, u, w, 'ko', 'MarkerSize', 3, 'MarkerFaceColor', [0.0,
```

```
    0.0, 0.0]);
    end
    %plot3(fig, v, u, w, 'ko', 'MarkerSize', 3, 'MarkerFaceColor', [0.0,
     0.0, 0.0]);
    xlabel(fig,'v')
    ylabel(fig,'u')
else
    axis(fig,1.75*[-pi pi -pi pi -pi pi], 'square')
    if (handles.sim)
        if (handles.node)
            plot3(fig, w, v, u, 'ko', 'MarkerSize', 3, 'MarkerFaceColor'
    , [1.0, 1.0, 0.0]);
        else
            plot3(fig, w, v, u, 'ko', 'MarkerSize', 1, 'MarkerFaceColor'
    , [0.0, 0.0, 0.0]);
        end
    else
    plot3(fig, w, v, u, 'ko', 'MarkerSize', 3, 'MarkerFaceColor', [0.0,
    0.0, 0.0]);
    end
    %plot3(fig, w, v, u, 'ko', 'MarkerSize', 3, 'MarkerFaceColor', [0.0,
     0.0, 0.0]);
    xlabel(fig,'\omega')
    ylabel(fig,'v')
end
hold(fig, 'off')


function idx = getDrawPlane(handles)
if (handles.uibuttongroup1.SelectedObject == handles.uwPlaneButton)
    idx = handles.UW;
```

```
242 elseif (handles.uibuttongroup1.SelectedObject == handles.uvPlaneButton)
243     idx = handles.UV;
244 else
245     idx = handles.VW;
246 end
247
248 % --- Executes on button press in plotButton.
249 function plotButton_Callback(hObject, eventdata, handles)
250 % hObject    handle to plotButton (see GCBO)
251 % eventdata  reserved - to be defined in a future version of MATLAB
252 % handles    structure with handles and user data (see GUIDATA)
253 [r, c] = size(handles.cpoints);
254 u = handles.configTable.Data{handles.U};
255 v = handles.configTable.Data{handles.V};
256 w = handles.configTable.Data{handles.W};
257 c = handles.collided;
258 handles.cpoints(r+1,:) = [w, u, v, c];
259 guidata(hObject, handles)
260
261
262 % --- Executes on button press in saveButton.
263 function saveButton_Callback(hObject, eventdata, handles)
264 % hObject    handle to saveButton (see GCBO)
265 % eventdata  reserved - to be defined in a future version of MATLAB
266 % handles    structure with handles and user data (see GUIDATA
267
268
269 % --- Executes when entered data in editable cell(s) in configTable.
270 function configTable_CellEditCallback(hObject, eventdata, handles)
271 % hObject    handle to configTable (see GCBO)
272 % eventdata  structure with the following fields (see MATLAB.UI.CONTROL.
```

```
    TABLE)
% Indices: row and column indices of the cell(s) edited
% PreviousData: previous data for the cell(s) edited
% EditData: string(s) entered by the user
% NewData: EditData or its converted form set on the Data property.
    Empty if Data was not changed
% Error: error string when failed to convert EditData to appropriate
    value for Data
% handles    structure with handles and user data (see GUIDATA)
C = eventdata.Indices;
c = C(2);
data = eventdata.NewData;
if (not(isempty(data)))
if (c == handles.S)
    %
elseif (c == handles.U)
    if (data > handles.max_angle)
        data = handles.max_angle;
    elseif (data < handles.min_angle)
        data = handles.min_angle;
    end
    handles.configTable.Data(c) = {data};
elseif (c == handles.V)
    if (data > handles.max_angle)
        data = handles.max_angle;
    elseif (data < handles.min_angle)
        data = handles.min_angle;
    end
    handles.configTable.Data(c) = {data};
elseif (c == handles.W)
%     if (data > pi)
```

```
301 %         data = pi;
302 %     elseif (data < -pi)
303 %         data = -pi;
304 %     end
305     data = mod(data, 2*pi);
306     handles.configTable.Data(c) = {data};
307 end
308 assignValuesAllen(handles)
309 end
310
311
312 % --- Executes on button press in sEditToggle.
313 function sEditToggle_Callback(hObject, eventdata, handles)
314 % hObject    handle to sEditToggle (see GCBO)
315 % eventdata  reserved - to be defined in a future version of MATLAB
316 % handles    structure with handles and user data (see GUIDATA)
317 v = get(hObject, 'Value');
318 handles.configTable.ColumnEditable(handles.S) = v;
319 if (v == 0)
320     handles.s_minus.Enable = 'off';
321     handles.s_plus.Enable = 'off';
322 else
323     handles.s_minus.Enable = 'on';
324     handles.s_plus.Enable = 'on';
325 end
326 % Hint: get(hObject,'Value') returns toggle state of sEditToggle
327
328
329 % --- Executes on button press in uEditToggle.
330 function uEditToggle_Callback(hObject, eventdata, handles)
331 % hObject    handle to uEditToggle (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
v = get(hObject, 'Value');
handles.configTable.ColumnEditable(handles.U) = v;
if (v == 0)
    handles.u_minus.Enable = 'off';
    handles.u_plus.Enable = 'off';
else
    handles.u_minus.Enable = 'on';
    handles.u_plus.Enable = 'on';
end
% Hint: get(hObject,'Value') returns toggle state of uEditToggle


% --- Executes on button press in vEditToggle.
function vEditToggle_Callback(hObject, eventdata, handles)
% hObject    handle to vEditToggle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
v = get(hObject, 'Value');
handles.configTable.ColumnEditable(handles.V) = v;
if (v == 0)
    handles.v_minus.Enable = 'off';
    handles.v_plus.Enable = 'off';
else
    handles.v_minus.Enable = 'on';
    handles.v_plus.Enable = 'on';
end
% Hint: get(hObject,'Value') returns toggle state of vEditToggle


```

```
363 % --- Executes on button press in wEditToggle.
364 function wEditToggle_Callback(hObject, eventdata, handles)
365 % hObject    handle to wEditToggle (see GCBO)
366 % eventdata  reserved - to be defined in a future version of MATLAB
367 % handles    structure with handles and user data (see GUIDATA)
368 v = get(hObject, 'Value');
369 handles.configTable.ColumnEditable(handles.W) = v;
370 if (v == 0)
371     handles.w_minus.Enable = 'off';
372     handles.w_plus.Enable = 'off';
373 else
374     handles.w_minus.Enable = 'on';
375     handles.w_plus.Enable = 'on';
376 end
377 % Hint: get(hObject,'Value') returns toggle state of wEditToggle
378
379
380 % --- Executes during object creation, after setting all properties.
381 function configTable_CreateFcn(hObject, eventdata, handles)
382 % hObject    handle to configTable (see GCBO)
383 % eventdata  reserved - to be defined in a future version of MATLAB
384 % handles    empty - handles not created until after all CreateFcns
    called
385 set(hObject, 'Data', cell(1,8));
386 set(hObject, 'ColumnName', {'S', 'U', 'V', 'Omega (W)', '', 'X', 'Y', 'Z
    '});
387
388 % --- Executes on button press in s_plus.
389 function s_plus_Callback(hObject, eventdata, handles)
390 % hObject    handle to s_plus (see GCBO)
391 % eventdata  reserved - to be defined in a future version of MATLAB
```

```
392 % handles    structure with handles and user data (see GUIDATA)
393 s = handles.configTable.Data{handles.S};
394 handles.configTable.Data(handles.S) = {s + 0.5};
395 assignValuesAllen(handles)
396
397 % --- Executes on button press in u_plus.
398 function u_plus_Callback(hObject, eventdata, handles)
399 % hObject    handle to u_plus (see GCBO)
400 % eventdata  reserved - to be defined in a future version of MATLAB
401 % handles    structure with handles and user data (see GUIDATA)
402 u = handles.configTable.Data{handles.U} + handles.delta;
403 if (u > handles.max_angle)
404   u = handles.max_angle;
405 end
406 handles.configTable.Data(handles.U) = {u};
407 assignValuesAllen(handles)
408
409 % --- Executes on button press in v_plus.
410 function v_plus_Callback(hObject, eventdata, handles)
411 % hObject    handle to v_plus (see GCBO)
412 % eventdata  reserved - to be defined in a future version of MATLAB
413 % handles    structure with handles and user data (see GUIDATA)
414 v = handles.configTable.Data{handles.V} + handles.delta;
415 if (v > handles.max_angle)
416   v = handles.max_angle;
417 end
418 handles.configTable.Data(handles.V) = {v};
419 assignValuesAllen(handles)
420
421 % --- Executes on button press in w_plus.
422 function w_plus_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to w_plus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
w = handles.configTable.Data{handles.W} + handles.delta;
w = mod(w, 2*pi);
handles.configTable.Data(handles.W) = {w};
assignValuesAllen(handles)

% --- Executes on button press in s_minus.
function s_minus_Callback(hObject, eventdata, handles)
% hObject    handle to s_minus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
s = handles.configTable.Data{handles.S};
handles.configTable.Data(handles.S) = {s - 0.05};
assignValuesAllen(handles)

% --- Executes on button press in u_minus.
function u_minus_Callback(hObject, eventdata, handles)
% hObject    handle to u_minus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
u = handles.configTable.Data{handles.U} - handles.delta;
if (u < handles.min_angle)
  u = handles.min_angle;
end
handles.configTable.Data(handles.U) = {u};
assignValuesAllen(handles)


% --- Executes on button press in v_minus.
```

```
454 function v_minus_Callback(hObject, eventdata, handles)
455 % hObject    handle to v_minus (see GCBO)
456 % eventdata  reserved - to be defined in a future version of MATLAB
457 % handles    structure with handles and user data (see GUIDATA)
458 v = handles.configTable.Data{handles.V} - handles.delta;
459 if (v < handles.min_angle)
460   v = handles.min_angle;
461 end
462 handles.configTable.Data(handles.V) = {v};
463 assignValuesAllen(handles)
464
465 % --- Executes on button press in w_minus.
466 function w_minus_Callback(hObject, eventdata, handles)
467 % hObject    handle to w_minus (see GCBO)
468 % eventdata  reserved - to be defined in a future version of MATLAB
469 % handles    structure with handles and user data (see GUIDATA)
470 w = handles.configTable.Data{handles.W} - handles.delta;
471 w = mod(w, 2*pi);
472 handles.configTable.Data(handles.W) = {w};
473 assignValuesAllen(handles)
474
475 % --- Executes during object creation, after setting all properties.
476 function axesLamp_CreateFcn(hObject, eventdata, handles)
477 % hObject    handle to axesLamp (see GCBO)
478 % eventdata  reserved - to be defined in a future version of MATLAB
479 % handles    empty - handles not created until after all CreateFcns
      called
480
481 % Hint: place code in OpeningFcn to populate axesLamp
482
483
```

```
484 % --- Executes when selected object is changed in uibuttongroup1.
485 function uibuttongroup1_SelectionChangedFcn(hObject, eventdata, handles)
486 % hObject    handle to the selected object in uibuttongroup1
487 % eventdata  reserved - to be defined in a future version of MATLAB
488 % handles    structure with handles and user data (see GUIDATA)
489 updatePlot(handles);
490
491
492 % --- Executes on button press in continuumSelectButton.
493 function continuumSelectButton_Callback(hObject, eventdata, handles)
494 % hObject    handle to continuumSelectButton (see GCBO)
495 % eventdata  reserved - to be defined in a future version of MATLAB
496 % handles    structure with handles and user data (see GUIDATA)
497
498 % Hint: get(hObject,'Value') returns toggle state of
    continuumSelectButton
499
500
501 % --- Executes when selected object is changed in uibuttongroupRobot.
502 function uibuttongroupRobot_SelectionChangedFcn(hObject, eventdata,
    handles)
503 % hObject    handle to the selected object in uibuttongroupRobot
504 % eventdata  reserved - to be defined in a future version of MATLAB
505 % handles    structure with handles and user data (see GUIDATA)
506 if (hObject == handles.continuumSelectButton)
507     handles.rigid = 0;
508 else
509     handles.rigid = 1;
510 end
511 assignValuesAllen(handles)
512 guidata(hObject, handles)
```

```
513
514
515 % --- Executes on button press in sim_enable.
516 function sim_enable_Callback(hObject, eventdata, handles)
517 % hObject    handle to sim_enable (see GCBO)
518 % eventdata  reserved - to be defined in a future version of MATLAB
519 % handles    structure with handles and user data (see GUIDATA)
520
521 % Hint: get(hObject,'Value') returns toggle state of sim_enable
522 val = get(hObject, 'Value');
523 if (val)
524     % enable the "load sim file" button
525     set(handles.file_button, 'Enable', 'on');
526
527     handles.sim = true;
528
529     % disable "interactive mode"
530     set(handles.sEditToggle, 'Value', 0);
531     set(handles.uEditToggle, 'Value', 0);
532     set(handles.vEditToggle, 'Value', 0);
533     set(handles.wEditToggle, 'Value', 0);
534
535     set(handles.sEditToggle, 'Enable', 'off');
536     set(handles.uEditToggle, 'Enable', 'off');
537     set(handles.vEditToggle, 'Enable', 'off');
538     set(handles.wEditToggle, 'Enable', 'off');
539
540     set(handles.s_minus, 'Enable', 'off');
541     set(handles.s_plus, 'Enable', 'off');
542     handles.configTable.ColumnEditable(handles.S) = 0;
543     set(handles.u_minus, 'Enable', 'off');
```

```
544     set(handles.u_plus, 'Enable', 'off');
545     handles.configTable.ColumnEditable(handles.U) = 0;
546     set(handles.v_minus, 'Enable', 'off');
547     set(handles.v_plus, 'Enable', 'off');
548     handles.configTable.ColumnEditable(handles.V) = 0;
549     set(handles.w_minus, 'Enable', 'off');
550     set(handles.w_plus, 'Enable', 'off');
551     handles.configTable.ColumnEditable(handles.W) = 0;
552 else
553     % disable the "load sim file" button
554     set(handles.file_button, 'Enable', 'of');
555 
556     handles.sim = false;
557 
558     % enable "interactive mode"
559     set(handles.sEditToggle, 'Enable', 'on');
560     set(handles.uEditToggle, 'Enable', 'on');
561     set(handles.vEditToggle, 'Enable', 'on');
562     set(handles.wEditToggle, 'Enable', 'on');
563 
564     set(handles.sEditToggle, 'Value', 0);
565     set(handles.uEditToggle, 'Value', 1);
566     set(handles.vEditToggle, 'Value', 1);
567     set(handles.wEditToggle, 'Value', 1);
568 
569     set(handles.s_minus, 'Enable', 'off');
570     set(handles.s_plus, 'Enable', 'off');
571     handles.configTable.ColumnEditable(handles.S) = 0;
572     set(handles.u_minus, 'Enable', 'on');
573     set(handles.u_plus, 'Enable', 'on');
574     handles.configTable.ColumnEditable(handles.U) = 1;
```

```
575     set(handles.v_minus, 'Enable', 'on');
576     set(handles.v_plus, 'Enable', 'on');
577     handles.configTable.ColumnEditable(handles.V) = 1;
578     set(handles.w_minus, 'Enable', 'on');
579     set(handles.w_plus, 'Enable', 'on');
580     handles.configTable.ColumnEditable(handles.W) = 1;
581 end
582
583 guidata(hObject, handles)
584
585 % --- Executes on button press in file_button.
586 function file_button_Callback(hObject, eventdata, handles_in)
587 % hObject    handle to file_button (see GCBO)
588 % eventdata  reserved - to be defined in a future version of MATLAB
589 % handles_in    structure with handles and user data (see GUIDATA)
590 handles = guidata(hObject);
591
592 % get the path
593 [file, path] = uigetfile('simfiles/*.csv');
594 handles.rawpath = csvread([path, file], 1, 0);
595 [r,~] = size(handles.rawpath);
596 path_len = handles.rawpath(r,1);
597 path_idx = handles.rawpath(r,2:(path_len+1)) + ones(1,path_len);
598 handles.path = zeros(path_len, 4);
599 for i = 1:path_len
600     handles.path(i,:) = [handles.state(1), handles.rawpath(path_idx(i)
    ,3), ...
601         handles.rawpath(path_idx(i),4), handles.rawpath(path_idx(i),2)];
602 end
603
604 % User Data for the Timer Object
```

```
605 data.t = 0;
606 data.tau = 0.01;
607 data.state   = handles.state;
608 data.path    = handles.path;
609 data.path_idx = 1;
610 data.path_len = path_len;
611 data.K       = [100, 0, 0; 200, 0, 0];
612 data.ravg    = zeros(5,20);
613 data.samples = 1;
614 data.elast   = [0, 0, 0, 0, 0];
615 data.gui = hObject.Parent;
616 data.handles = handles;
617
618 xtimer = timer;
619 xtimer.TimerFcn = @execute_timestep;
620 xtimer.StopFcn = @cleanup_timer;
621 xtimer.ExecutionMode = 'fixedSpacing';
622 xtimer.Period = 0.01;
623 xtimer.StartDelay = 3;
624 xtimer.Tag = 'exe_timer';
625 xtimer.UserData = data;
626 guidata(hObject, handles);
627
628 %function
629 start(xtimer);
630
631
632 function execute_timestep(obj, ~)
633 S = 1; U = 2; V = 3; W = 4;
634 epsilon = 0.01;
635 data = get(obj, 'UserData');
```

```
636 data.t = data.t + data.tau;
637 data.t;
638 setpt = data.path(data.path_idx,:);
639 % data has the following properties
640 % t        :
641 % tau      :
642 % state    : [s, u, v, w]
643 % K        : [kp, kd, ki]
644 % ravg     : [ [], [], [], [] ]
645 % samples  :
646 % elast    : [el1, el2, el3, el4]
647 s_0 = 0.95; d = 0.022;
648 u = data.state(U); v = data.state(V); s = data.state(S);
649 l = [0, -2*d, 1; 2*d, 0, 1; 0, 2*d, 1; -2*d, 0, 1] *[u;v;s];
650 u = setpt(U); v = setpt(V); s = setpt(S);
651 l_set = [0, -2*d, 1; 2*d, 0, 1; 0, 2*d, 1; -2*d, 0, 1] *[u;v;s];
652 out = zeros(1,5);
653 err = [l_set - l; setpt(W)-data.state(W)];
654 for i = 1:5
655     if (data.samples < 20)
656         data.ravg(i,data.samples) = err(i);
657         data.samples = data.samples + 1;
658     else
659         data.ravg(i,:) = [data.ravg(i,2:20), err(i)];
660     end
661     if (i < 5) % tendons
662     out(i) = err(i) * data.K(1,1) + ...
663             (err(i) - data.elast(i))/data.t * data.K(1,2) + ...
664             (sum(data.ravg(i))/data.samples) * data.K(1,3);
665     else % turntable
666     out(i) = err(i) * data.K(2,1) + ...
```

```
667             (err(i) - data.elast(i))/data.t * data.K(2,2) + ...
668             (sum(data.ravg(i))/data.samples) * data.K(2,3);
669     end
670     % saturate
671     if (out(i) > 12)
672         out(i) = 12;
673     elseif (out(i) < -12)
674         out(i) = -12;
675     end
676 end
677
678 %% simulate
679 % tendon lengths
680 l_next = l + (out(1:4))'/12*(32/60)*data.tau*0.44*pi;
681
682 state_next = (1/(4*d)) * [0, 1, 0, -1; -1, 0, 1, 0; d, d, d, d] * l_next
    ;
683
684 % turntable   (pi/9) rad/s at 12V
685 w_next = data.state(W) + (out(5)/12)*(pi/9)*data.tau*15;
686
687 data.state = [state_next(3), state_next(1), state_next(2), w_next];
688
689 % update the GUI
690 handles = guidata(data.gui);
691 handles.state = data.state;
692 handles.configTable.Data(S) = {handles.state(S)};
693 handles.configTable.Data(U) = {handles.state(U)};
694 handles.configTable.Data(V) = {handles.state(V)};
695 handles.configTable.Data(W) = {handles.state(W)};
696
```

```
697
698
699 % increment the path index and stop if reached end of path
700 delta = abs(data.state - setpt);
701 b = (delta < epsilon*ones(1,4));
702 if (b)
703     data.path_idx = data.path_idx + 1;
704     handles.node = true;
705     if (data.path_idx > data.path_len)
706         stop(obj)
707     end
708 else
709     handles.node = false;
710 end
711
712 guidata(data.gui, handles);
713 assignValuesAllen(handles);
714
715 obj.UserData = data;
716
717
718 function cleanup_timer(obj, ~)
719 delete(obj);
```

Listing 37: Kinematic GUI Source

```
1 function collided = Lamp_Sim_GUI(s, u, v, w, handles)
2 num_spheres = 29;
3 big_rad = 30;
4 lil_rad = 15;
5 point_a = zeros(num_spheres,3);
6
```

```
%Rotation matrix (by theta)
T = [ cos(w), -sin(w),  0,  0;
      sin(w),  cos(w),  0,  0;
           0,       0,  1,  0;
           0,       0,  0,  1
    ];
for i=1:num_spheres

    scale = i/num_spheres;

    H_a=T*tranMatrixA(u*scale,v*scale,s*scale);

%     point_a(1,1,i)= H_a(1,4);
%     point_a(1,2,i)= H_a(2,4);
%     point_a(1,3,i)= H_a(3,4);
    point_a(i,1)= H_a(1,4);
    point_a(i,2)= H_a(2,4);
    point_a(i,3)= H_a(3,4);

end
fig = handles.axesLamp;

% update position of end effector
handles.configTable.Data(handles.X) = {H_a(1,4)};
handles.configTable.Data(handles.Y) = {H_a(2,4)};
handles.configTable.Data(handles.Z) = {H_a(3,4)};

hold(fig, 'on')
grid(fig, 'on')
axis(fig,[-1 1 -1 1 0 2])
xlabel(fig,'x')
```

```
38 ylabel(fig,'y')
39
40
41 R = [ cos(w), -sin(w),  0;
42        sin(w),  cos(w),  0;
43             0,       0,  1;
44     ];
45 %% plot base
46 ext = 0.12; %read_la
47 %% "read in"  points for objects
48 %points;
49 %h ; o ; la ; point_base ; point_base_la ; point_arm_la ; point_orient;
50 [offset, point_base, point_orient,...
51    point_grp1, point_grp2, point_grp3, ...
52    point_base_la, point_arm_la, ...
53    point_shelf1, point_cup1, point_cup2, point_cup3] = points(ext,
     point_a(num_spheres,:) ,H_a);
54 % [offset, point_base, point_orient,...
55 %     point_grp1, point_grp2, point_grp3, ...
56 %     point_base_la, point_arm_la, ...
57 %     point_shelf1,point_shelf2, point_cup1, point_cup2, point_cup3] =
     points2(ext, point_a(num_spheres,:) ,H_a);
58 % Lamp box/base
59 V = zeros(8,3);
60 for i=1:8
61     V(i,:) = (R*point_base.p(i,:)')';
62 end
63 F = [1,2,4,3; 5,6,8,7; 1,2,6,5; 2,4,8,6; 3,4,8,7; 1,3,7,5];
64 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
     , [0 0 0]);
65 %p = patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], '
```

```
    EdgeColor', [0 0 0], ...
%     'FaceVertexAlphaData', 0.2, 'FaceAlpha', 'flat');
% Lamp orientation
for i=1:4
    V(i,:) = (R*point_orient(i,:)')';
end
plot3(fig,V(1:2,1),V(1:2,2),V(1:2,3),'k-','LineWidth',2);
plot3(fig,V(2:3,1),V(2:3,2),V(2:3,3),'k-','LineWidth',1.5);
plot3(fig,V(2:2:4,1),V(2:2:4,2),V(2:2:4,3),'k-','LineWidth',1.5);
% Gripper
for i=1:8
    V(i,:) = (eye(3)*point_grp1.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
    , [0 0 0]);
%patch(fig,'Faces', F, 'Vertices', W(:,1:3), 'FaceColor', [1.0 0.8 0.8],
     'EdgeColor', [0 0 0]);
for i=1:8
    V(i,:) = (eye(3)*point_grp2.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
    , [0 0 0]);
%patch(fig,'Faces', F, 'Vertices', W(:,1:3), 'FaceColor', [1.0 0.8 0.8],
     'EdgeColor', [0 0 0]);
for i=1:8
    V(i,:) = (eye(3)*point_grp3.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
    , [0 0 0]);
%patch(fig,'Faces', F, 'Vertices', W(:,1:3), 'FaceColor', [1.0 0.8 0.8],
     'EdgeColor', [0 0 0]);
```

```
90 % LA base
91 for i=1:8
92     V(i,:) = (R*point_base_la.p(i,:)')';
93 end
94 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.25 0.25 0.25], '
    EdgeColor', [0 0 0]);
95 % LA arm
96 for i=1:8
97     V(i,:) = (R*point_arm_la.p(i,:)')';
98 end
99 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.5 0.5 0.5], '
    EdgeColor', [0 0 0]);
100 % shelf1
101 for i=1:8
102     V(i,:) = (eye(3,3)*point_shelf1.p(i,:)')';
103 end
104 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.4 0.4 0.4], '
    EdgeColor', [0 0 0]);
105 % shelf2
106 % for i=1:8
107 %     V(i,:) = (eye(3,3)*point_shelf2.p(i,:)')';
108 % end
109 % patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.4 0.4 0.4], '
    EdgeColor', [0 0 0]);
110 %cup1
111 for i=1:8
112     V(i,:) = (eye(3,3)*point_cup1.p(i,:)')';
113 end
114 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.8 0.2 0.2], '
    EdgeColor', [0 0 0]);
115 %cup2
```

```
116 for i=1:8
117     V(i,:) = (eye(3,3)*point_cup2.p(i,:)')';
118 end
119 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.8 0.8 0.8], '
    EdgeColor', [0 0 0]);
120 %cup3
121 for i=1:8
122     V(i,:) = (eye(3,3)*point_cup3.p(i,:)')';
123 end
124 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.8 0.2 0.2], '
    EdgeColor', [0 0 0]);
125
126 % Plot the continuum arm
127 for k=1:num_spheres-1
128     if (mod(k,4)==1)
129         rad = big_rad;
130     else
131         rad = lil_rad;
132     end
133     plot3(fig,point_a(k,1),point_a(k,2),point_a(k,3)+offset,'.k','
    MarkerSize',rad);
134 end
135
136 %% C-Space
137 % fig = handles.axesCspace;
138 % hold(fig,'on')
139 % grid(fig, 'on')
140 % axis(fig,[-2*pi 2*pi -2*pi 2*pi -2*pi 2*pi])
141 % xlabel(fig,'Omega')
142 % ylabel(fig,'U')
143
```

```
144 %% check for collisions
145 c_obs = {point_shelf1, point_cup1, point_cup2, point_cup3};
146 collided = check_collision(point_a, offset, {point_grp1, point_grp2,
    point_grp3}, c_obs);
147
148 % plotting c-space
149 % if (~collided)
150 %     plot3(fig, w, u, v, 'ko', ...
151 %         'MarkerSize', 3, 'MarkerFaceColor', [0.0, 0.0, 0.0]);
152 % else
153 %     plot3(fig, w, u, v, 'rx', ...
154 %         'LineWidth', 2);
155 % end
156 % [r, ~] = size(handles.cpoints);
157 % for i = 2:r
158 %     if (handles.cpoints(i,4))
159 %         plot3(fig, handles.cpoints(i,1), handles.cpoints(i,2), handles
    .cpoints(i,3) ,...
160 %             'x', 'LineWidth', 2, 'LineColor', [0.2, 0.2, 0.8]);
161 %     else
162 %         plot3(fig, handles.cpoints(i,1), handles.cpoints(i,2), handles
    .cpoints(i,3) ,...
163 %             'o', 'MarkerSize', 3, 'MarkerFaceColor', [0.2, 0.2, 0.8]);
164 %     end
165 % end
166 end
```

Listing 38: Main continuum simulation file of the Kinematic GUI

```
1 function plot_torus(u,v,w, fig)
2
3 R = 3*sqrt(2)*pi;
```

```
r = sqrt(2)*pi;
theta = (0:0.01:2*pi)';
phi = 0:0.01:2*pi;
len = length(phi);
X = (R + r*cos(theta))*cos(phi);
Y = (R + r*cos(theta))*sin(phi);
Z = r*sin(theta)*ones(1,len);

s = surface(fig, X,Y,Z);
set(s, 'FaceColor',[0, 0 ,1]);
set(s, 'FaceAlpha',0.2);
set(s, 'FaceLighting', 'gouraud');
set(s, 'EdgeColor', 'none');
set(s, 'EdgeAlpha',0.1);

%X = (R + r*cos(theta))*cos(phi);
%Y = (R + r*cos(theta))*sin(phi);
%Z = r*sin(theta)*ones(1,len);

% end plane
endx = (R + r*cos(theta'))*cos(-pi);
endy = (R + r*cos(theta'))*sin(-pi);

psi = w;
x = (R + r*cos(theta'))*cos(psi);
y = (R + r*cos(theta'))*sin(psi);
z = r*sin(theta');

patch(fig, 'XData',endx,'YData', endy, 'ZData',z, 'FaceColor', [0 0 0],
    'EdgeColor', [0 0 0], 'FaceAlpha', 1.0);
patch(fig, 'XData',x,'YData', y, 'ZData',z, 'FaceColor', [1 0 0], '
```

```
    EdgeColor', [0 0 0], 'FaceAlpha', 0.7);
%patch('XData',R*cos(phi),'YData', R*sin(phi), 'ZData',zeros(1,len), '
    FaceColor', [0 0 1], 'EdgeColor', [0 0 0], 'FaceAlpha', 0);
hold on
m = sqrt(u^2 + v^2);
th = atan2(v,u);
cx = R*cos(psi); cy = R*sin(psi); cz = 0;
xx = (R + m*cos(th))*cos(psi); yy = (R + m*cos(th))*sin(psi); zz = m*sin
    (th);

% edge of the slice
cxx = (R + r*cos(pi))*cos(psi);
cyy = (R + r*cos(pi))*sin(psi);
czz = r*sin(pi);

% center of slice
plot3(fig, cx, cy, cz, 'k.', 'MarkerSize', 12);

% configuration c = [u v w]
plot3(fig, xx, yy, zz, 'k.', 'MarkerSize', 16);

% \omega = 0 axis for reference
plot3(fig, [0,R-r],[0,0],[0,0], 'k--', 'LineWidth', 1.3);

% \omega line
plot3(fig, [0,cxx],[0,cyy],[0,czz], 'k--', 'LineWidth', 1.3);

% u
plot3(fig, [cx,xx],[cy,yy],[0,0], 'k-');

% v
```

```
plot3(fig, [xx,xx],[yy,yy],[0,zz], 'k-');
end
```

Listing 39: Function to plot the c-space of CuRLE

```
function [ht, point_base, point_orient,...
    point_grp1, point_grp2, point_grp3,...
    point_base_la, point_arm_la, ...
    point_shelf, point_cup1, point_cup2, point_cup3] = points(ext,
   arm_end, Ha)
%% extra vars
T = [
    -0.5, -0.5, -0.5;
    -0.5,  0.5, -0.5;
     0.5, -0.5, -0.5;
     0.5,  0.5, -0.5;
    -0.5, -0.5,  0.5;
    -0.5,  0.5,  0.5;
     0.5, -0.5,  0.5;
     0.5,  0.5,  0.5;
];
h_ = 0.242; o = 0.055; la = 0.252;
ht = h_ + o + la + ext;
%% lamp base
w = 0.495; l = 0.495; h = h_;
c = [0, 0, o+h_/2];
point_base.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_base.w = w;
point_base.l = l;
point_base.h = h;
point_base.c = c;
%% lamp arrow (orientation)
```

```
point_orient = [
        0,         0, o+h_;
        0, -1/2+0.1, o+h_;
     -w/8,      -1/8, o+h_;
      w/8,      -1/8, o+h_;
];
%% gripper1
w = 0.088; l = 0.02; h = 0.01;
%c = [arm_end(1), arm_end(2), arm_end(3)+h/2+ht];
c = [0, 0, 0];
c_rot = (Ha*[c';0]);
p = T*diag([w l h]);
for i = 1:8
    temp = (Ha*[p(i,:)';0])';
    point_grp1.p(i,:) = temp(1,1:3) + arm_end + c_rot(1:3,1)' + [0, 0,
   ht];
end
point_grp1.w = w;
point_grp1.l = l;
point_grp1.h = h;
point_grp1.c = c;
%% gripper2
w = 0.02; l = 0.01; h = 0.10;
%c = [arm_end(1)+0.033+w/2, arm_end(2), arm_end(3)-0.01+h/2+ht];
c = [0.044+w/2, 0, h/2];
c_rot = (Ha*[c';0]);
%p = T*diag([w l h]) + ones(8,3)*diag(c_rot(1:3,:));
p = T*diag([w l h]);
for i = 1:8
    temp = (Ha*[p(i,:)';0])';
    point_grp2.p(i,:) = temp(1,1:3)+ arm_end + c_rot(1:3,1)' + [0, 0, ht
```

```
    ];
end
point_grp2.w = w;
point_grp2.l = l;
point_grp2.h = h;
point_grp2.c = c;
%% gripper3
w = 0.02; l = 0.01; h = 0.10;
%c = [arm_end(1)-0.033-w/2, arm_end(2), arm_end(3)-0.01+h/2+ht];
c = [-0.044-w/2, 0, h/2];
c_rot = (Ha*[c';0]);
%p = T*diag([w l h]) + ones(8,3)*diag(c_rot(1:3,:));
p = T*diag([w l h]);
for i = 1:8
    temp = (Ha*[p(i,:)';0])';
    point_grp3.p(i,:) = temp(1,1:3)+ arm_end + c_rot(1:3,1)' + [0, 0, ht
    ];
end
point_grp3.w = w;
point_grp3.l = l;
point_grp3.h = h;
point_grp3.c = c;
%% la_base
w = 0.035; l = 0.035; h = la;
c = [0,0,o+h_+la/2];
point_base_la.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_base_la.w = w;
point_base_la.l = l;
point_base_la.h = h;
point_base_la.c = c;
%% la_arm
```

```
w = 0.0175; l = 0.0175; h = ext;
c = [0, 0, o+h_+la+ext/2];
point_arm_la.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_arm_la.w = w;
point_arm_la.l = l;
point_arm_la.h = h;
point_arm_la.c = c;
%% shelf
w = 0.26; l = 0.66; h = 0.045;
c = [0.75, 0, 1.14];
point_shelf.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_shelf.w = w;
point_shelf.l = l;
point_shelf.h = h;
point_shelf.c = c;

%% cup1
w = 0.07; l = 0.07; h = 0.20;
c = [0.75, -0.17, 1.265];
point_cup1.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_cup1.w = w;
point_cup1.l = l;
point_cup1.h = h;
point_cup1.c = c;

%% cup2
w = 0.07; l = 0.07; h = 0.11;
c = [0.75, 0, 1.22];
point_cup2.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_cup2.w = w;
point_cup2.l = l;
```

```
point_cup2.h = h;
point_cup2.c = c;

%% cup3
w = 0.06; l = 0.06; h = 0.17;
c = [0.75, 0.26, 1.25];
point_cup3.p = T*diag([w l h]) + ones(8,3)*diag(c);
point_cup3.w = w;
point_cup3.l = l;
point_cup3.h = h;
point_cup3.c = c;

end
```

Listing 40: Function that returns all the vertex points of the simulation environment

```
function collided = rigidLamp_Sim_GUI(s, u, v, w, handles)
num_spheres = 29;
big_rad = 30;
lil_rad = 15;
point_a = zeros(num_spheres,3);
% rigidDiameter = 8; % big Balls
rigidDiameter = 5;    % small Balls

%Rotation matrix (by theta)
T = [ cos(w), -sin(w),  0,  0;
      sin(w),  cos(w),  0,  0;
           0,       0,  1,  0;
           0,       0,  0,  1
    ];


```

```
% Mod by 2*pi to "achieve full c-space"
%theta = sqrt( (mod(u,2*pi)/2)^2 + (mod(v,2*pi)/2)^2 );
%phi = atan2(-mod(v,2*pi),-mod(u,2*pi));

% Acutal equation for chords
theta = sqrt( (u/2)^2 + (v/2)^2 );
phi = atan2(-v,-u);
%equation of chord length
%L=2*(arcLength/angleOfArc)*sin(angleOfArc/2);
L = 2 * ( s / (theta*2) ) * sin( (theta*2) /2 );
L = s;

PHI = [ cos(phi), -sin(phi),  0,  0;
        sin(phi),  cos(phi),  0,  0;
               0,         0,  1,  0;
               0,         0,  0,  1];

THETA = [ 1,           0,           0, 0;
          0,  cos(theta),  sin(theta), 0;
          0, -sin(theta),  cos(theta), 0;
          0,           0,           0, 1];

S = [ 1, 0, 0, 0;
      0, 1, 0, 0;
      0, 0, 1, L;
      0, 0, 0, 1];

for i=1:num_spheres

    scale = i/num_spheres;

```

```
48      H_a = T*tranMatrixA(u*scale,v*scale,s*scale);
49
50  %       point_a(1,1,i)= H_a(1,4);
51  %       point_a(1,2,i)= H_a(2,4);
52  %       point_a(1,3,i)= H_a(3,4);
53      point_a(i,1)= H_a(1,4);
54      point_a(i,2)= H_a(2,4);
55      point_a(i,3)= H_a(3,4);
56
57  end
58
59  %H_a for rigidLink chord
60  H_a = T*PHI*THETA*S*THETA*PHI;
61  endPoint=transpose(squeeze(H_a(1:3,4)));
62
63  fig = handles.axesLamp;
64
65  % update position of end effector
66  handles.configTable.Data(handles.X) = {H_a(1,4)};
67  handles.configTable.Data(handles.Y) = {H_a(2,4)};
68  handles.configTable.Data(handles.Z) = {H_a(3,4)};
69
70  hold(fig, 'on')
71  grid(fig, 'on')
72  axis(fig,[-1 1 -1 1 0 2])
73  xlabel(fig,'x')
74  ylabel(fig,'y')
75
76
77  R = [ cos(w), -sin(w),  0;
78        sin(w),  cos(w),  0;
```

```
79              0,          0,   1;
80      ];
81 %% plot base
82 ext = 0.12; %read_la
83 %% "read in"  points for objects
84 %points;
85 %h ; o ; la ; point_base ; point_base_la ; point_arm_la ; point_orient;
86 % [offset, point_base, point_orient,...
87 %      point_grp1, point_grp2, point_grp3, ...
88 %      point_base_la, point_arm_la, ...
89 %      point_shelf, point_cup1, point_cup2, point_cup3] = points(ext,
       point_a(num_spheres,:) ,H_a);
90
91 [offset, point_base, point_orient,...
92      point_grp1, point_grp2, point_grp3, ...
93      point_base_la, point_arm_la, ...
94      point_shelf, point_cup1, point_cup2, point_cup3] = points(ext,
       endPoint ,H_a);
95
96 % Lamp box/base
97 V = zeros(8,3);
98 for i=1:8
99     V(i,:) = (R*point_base.p(i,:)')';
100 end
101 F = [1,2,4,3; 5,6,8,7; 1,2,6,5; 2,4,8,6; 3,4,8,7; 1,3,7,5];
102 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
       , [0 0 0]);
103 % Lamp orientation
104 for i=1:4
105     V(i,:) = (R*point_orient(i,:)')';
106 end
```

```
107 plot3(fig,V(1:2,1),V(1:2,2),V(1:2,3),'k-','LineWidth',2);
108 plot3(fig,V(2:3,1),V(2:3,2),V(2:3,3),'k-','LineWidth',1.5);
109 plot3(fig,V(2:2:4,1),V(2:2:4,2),V(2:2:4,3),'k-','LineWidth',1.5);
110 % Gripper
111 for i=1:8
112     V(i,:) = (eye(3)*point_grp1.p(i,:)')';
113 end
114 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
    , [0 0 0]);
115 %patch(fig,'Faces', F, 'Vertices', W(:,1:3), 'FaceColor', [1.0 0.8 0.8],
     'EdgeColor', [0 0 0]);
116 for i=1:8
117     V(i,:) = (eye(3)*point_grp2.p(i,:)')';
118 end
119 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
    , [0 0 0]);
120 %patch(fig,'Faces', F, 'Vertices', W(:,1:3), 'FaceColor', [1.0 0.8 0.8],
     'EdgeColor', [0 0 0]);
121 for i=1:8
122     V(i,:) = (eye(3)*point_grp3.p(i,:)')';
123 end
124 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [1.0 0 0], 'EdgeColor'
    , [0 0 0]);
125 %patch(fig,'Faces', F, 'Vertices', W(:,1:3), 'FaceColor', [1.0 0.8 0.8],
     'EdgeColor', [0 0 0]);
126 % LA base
127 for i=1:8
128     V(i,:) = (R*point_base_la.p(i,:)')';
129 end
130 patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.25 0.25 0.25], '
    EdgeColor', [0 0 0]);
```

```
% LA arm
for i=1:8
    V(i,:) = (R*point_arm_la.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.5 0.5 0.5], '
    EdgeColor', [0 0 0]);
% shelf
for i=1:8
    V(i,:) = (eye(3,3)*point_shelf.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.4 0.4 0.4], '
    EdgeColor', [0 0 0]);
for i=1:8
    V(i,:) = (eye(3,3)*point_cup1.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.8 0.2 0.2], '
    EdgeColor', [0 0 0]);
for i=1:8
    V(i,:) = (eye(3,3)*point_cup2.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.8 0.8 0.2], '
    EdgeColor', [0 0 0]);
for i=1:8
    V(i,:) = (eye(3,3)*point_cup3.p(i,:)')';
end
patch(fig,'Faces', F, 'Vertices', V, 'FaceColor', [0.8 0.2 0.2], '
    EdgeColor', [0 0 0]);

% Plot the continuum arm
% for k=1:num_spheres-1
%     if (mod(k,4)==1)
```

```
157 %          rad = big_rad;
158 %      else
159 %          rad = lil_rad;
160 %      end
161 %      plot3(fig,point_a(k,1),point_a(k,2),point_a(k,3)+offset,'.k','
    MarkerSize',rad);
162 % end
163
164 %Plot the rigid link arm
165 plot3(fig,[0 endPoint(1)],[0 endPoint(2)],[offset endPoint(3)+offset],'-
    k','LineWidth',rigidDiameter)
166
167 %% C-Space
168 fig = handles.axesCspace;
169 hold(fig,'on')
170 grid(fig, 'on')
171 axis(fig,[-pi pi -pi pi -pi pi])
172 xlabel(fig,'Omega')
173 ylabel(fig,'U')
174
175 %% check for collisionss
176 %c_obs = {point_shelf, point_cup1, point_cup2, point_cup3};
177 % collided = check_collision(point_a, offset, {point_grp1, point_grp2,
    point_grp3}, c_obs);
178 collided=false;
179
180 % if (~collided)
181 %      plot3(fig, th, u, v, 'ko', ...
182 %          'MarkerSize', 3, 'MarkerFaceColor', [0.0, 0.0, 0.0]);
183 % else
184 %      plot3(fig, th, u, v, 'rx', ...
```

```
%          'LineWidth', 2);
% end
% [r, ~] = size(handles.cpoints);
% for i = 2:r
%     if (handles.cpoints(i,4))
%         plot3(fig, handles.cpoints(i,1), handles.cpoints(i,2), handles
    .cpoints(i,3) ,...
%             'x', 'LineWidth', 2, 'LineColor', [0.2, 0.2, 0.8]);
%     else
%         plot3(fig, handles.cpoints(i,1), handles.cpoints(i,2), handles
    .cpoints(i,3) ,...
%             'o', 'MarkerSize', 3, 'MarkerFaceColor', [0.2, 0.2, 0.8]);
%     end
% end
end
```

Listing 41: Main rigid-link simulation file of the Kinematic GUI

```
function R = rot_mat(th, u, v)
c1 = cos(u);  s1 = sin(u);
c2 = cos(v);  s2 = sin(v);
c3 = cos(th); s3 = sin(th);
Rz = [c3, -s3, 0; s3, c3, 0; 0, 0, 1];
Ry = [c2, 0, s2; 0, 1, 0; -s2, 0, c2];
Rx = [1, 0, 0; 0, c1, -s1; 0, s1, c1];
R = Rx*Ry*Rz;
end
```

Listing 42: Function to generate a rotation matrix

```
function v_prime = rotate(v, R)
v_prime = zeros(size(v));
[r,~] = size(v);
```

```
4
5 for i = 1:r
6   v_prime(i,:) = (R * v(i,:)')';
7 end
8 end
```

Listing 43: Function to rotate a point by matrix

```
1 function Ha = tranMatrixA(u,v,s)
2
3 theta = sqrt(u^2 + v^2);
4 gam = (cos(theta)-1)/(theta^2);
5 zet = sin(theta)/theta;
6
7 Ha = [ gam*(v^2)+1 , -gam*v*u      ,  zet*v       , -gam*s*v;
8       -gam*u*v      ,  gam*(u^2)+1 , -zet*u       ,  gam*s*u;
9       -zet*v        ,  zet*u       ,  cos(theta)  ,  zet*s  ;
10       0            ,  0           ,  0           ,  1      ];
11
12
13 end
```

Listing 44: Function that returns the transformation matrix of the continuum element

```
1 function plot_graph(fig, data, goal, task)
2 figure(fig);
3 limits = [-pi pi -pi pi];
4 %goal = [1.57, 1.76, 0];
5 goalIsConnected = false;
6 goalId = -1;
7 id = data(:,1);
8 state.th = data(:,2);
9 state.u = data(:,3);
```

```
state.v = data(:,4);



epsilon = 0.0001;

if (task == 1)
    load('C:\Users\zhawks\Documents\roboticsLab\matlab\gui_sim\
    cspace_shelf_grp.mat', 'points');
elseif (task == 2)
    load('C:\Users\zhawks\Documents\roboticsLab\matlab\gui_sim_thesis\
    cspace_shelf_grp2.mat', 'points');
end
[r,~] = size(points);
cspace = [points(:,1:2), zeros(r,1)];

% determine edges
[r,c] = size(data);

% index of the path is the last row
idx_p = r;

% preset all edges to -1
r = r-1;
edges = ones(r,c-4)*(-1);

% determine which id is the goal id
for i = 1:r
    delta = [abs(state.th(i) - goal(1));abs(state.u(i) - goal(2));abs(
    state.v(i) - goal(3))];
    if (delta(1) <= epsilon &&...
```

```
38          delta(2) <= epsilon &&...
39          delta(3) <= epsilon )
40         goalIsConnected = true;
41         goalId = id(i) + 1;
42     end
43 end
44
45 opt_path = data(idx_p,:);
46
47 for i=1:r
48     j = 5;
49     while (j < c && data(i,j) ~= 0)
50         edges(i,j-4) = data(i,j);
51         edges(i,j+1-4) = data(i, j+1);
52         j = j + 2;
53     end
54 end
55 hold on
56
57 MarkerSize = 4;
58 % plot the start state
59 plot(state.th(1) , state.u(1), 'go', ...
60     'MarkerSize',MarkerSize, ...
61     'MarkerEdgeColor','k', ...
62     'MarkerFaceColor',[0,1.0,0]);
63 % plot all the other states
64 plot(state.th(2:r), state.u(2:r), 'bo', ...
65     'MarkerSize',MarkerSize, ...
66     'MarkerEdgeColor','k', ...
67     'MarkerFaceColor',[0.5,0.5,0.5]);
68 % plot the goal state
```

```
plot(goal(1), goal(2), 'ro', ...
    'MarkerSize',MarkerSize, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,0,0]);

% plot first part of path & c-obs (for legend)
plot(state.th(opt_path(1)+1) , state.u(opt_path(1)+1), 'go', ...
    'MarkerSize',MarkerSize, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,1.0,0]);
plot(cspace(1,1), cspace(1,2), 'k.');

% plot edges
[er, ec] = size(edges);
for i = 1:er
    pt1 = [state.th(i),state.u(i)];
    for j = 2:2:ec
        val = edges(i,j);
        if val ~= -1 && val < r
            pt2 = [state.th(val+1),state.u(val+1)];
            plot([pt1(1), pt2(1)], [pt1(2), pt2(2)],'k-');
        end
    end
end

plot(state.th(2:r), state.u(2:r), 'bo', ...
    'MarkerSize',MarkerSize, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[0.5,0.5,0.5]);

% plot path
```

```
for i=1:length(opt_path)
    plot(state.th(opt_path(i)+1) , state.u(opt_path(i)+1), 'go', ...
    'MarkerSize',MarkerSize, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,1.0,0]);
end


% re-plot the start
plot(state.th(1) , state.u(1), 'go', ...
    'MarkerSize',MarkerSize, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[0,1.0,0]);
% re-plot the goal
plot(goal(1), goal(2), 'ro', ...
    'MarkerSize',MarkerSize, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,0,0]);

% plot the cspace
plot(cspace(:,1), cspace(:,2), 'k.');

if (goalIsConnected)
    title(sprintf('RRT: %d nodes - Goal Id %d',r,goalId));
else
    title(sprintf('RRT: %d nodes',r))
end

xlabel('\omega [rad]')
ylabel('u [-]')
legend('Start', 'Nodes', 'Goal' ,'Path', 'C_{obs}', 'Location', '
```

```
    eastoutside');

axis(limits)
hold off
end
```

Listing 45: Function that plots the continuum RRT

```
clear;clc;
%% generic (for rapid testing...)
%Start = [1.57, 1.76, 0];
Goal = [-1.57, -1.76, 0];
data = csvread('../path.csv', 1,0);
plot_graph(9,data, Goal, 1);

%%% Scenario 2 (which comes first)
%% (a)
% Start = [0, 0, 0];
% Goal = [0, -1.47, 0];
% data = csvread('../path_arm2a.csv', 1,0);
% plot_graph(1,data, Goal, 2);
%% (b)
% Start = [0, -1.47, 0];
% Goal = [0, -0.79, 0];
% data = csvread('../path_arm2b.csv', 1,0);
% plot_graph(2,data, Goal, 2);
%
%%% Scenario 1 (which comes second)
% %(a)
% Start = [0, -0.79, 0];
% Goal = [-1.57, -1.76, 0];
% data = csvread('../path_arm1a.csv', 1,0);
```

```
% plot_graph(3,data, Goal, 1);
% %(b)
% Start = [-1.57, -1.76, 0];
% Goal = [-1.57, -0.79, 0];
% data = csvread('../path_arm1b.csv', 1,0);
% plot_graph(4,data, Goal, 1);
% %(c)
% Start = [-1.57, -0.79, 0];
% Goal = [0, 0, 0];
% data = csvread('../path_arm1c.csv', 1,0);
% plot_graph(5,data, Goal, 1);
```

Listing 46: Script that executes the continuum RRT visualization

Appendix F Mobile Base Simulation Software

```
function M = animateRRTpath(fig, scenario, data, Goal, dosave, fname)
if nargin < 6
    fname = 'rrtPath.mp4';
elseif nargin < 5
    dosave = false;
    fname = 'blank.mp4';
end
[r,c] = size(data);
idx_p = r;
r = r -1;
figHandle = figure(fig);
frameCount = 1;
vertex = 1;
set(figHandle, 'units', 'inches', 'pos', [0 0 15 11.5]);

```

```
%% prep graph
cnv = 12*2.54;
xmax = 15*cnv; xmin = -1*cnv;
ymax = 11*cnv; ymin = -1*cnv;
id = data(:,1);
state.x = data(:,2);
state.y = data(:,3);
edges = zeros(r,c-4);
for i = 1:r
    for j = 1:c-4
        edges(i,j) = -1;
    end
end
for i=1:r
    j = 5;
    while (j < c && data(i,j) ~= 0)
        edges(i,j-4) = data(i,j);
        edges(i,j+1-4) = data(i, j+1);
        j = j + 2;
    end
end
if scenario == 1
build_config_obstacles;
elseif scenario == 2
build_config_obstacles2;
elseif scenario == 0
build_config_obstaclesNone;
end
opt_path = data(idx_p,:);
%% boundary vertices
%v = [  0,0; 14,0; 14,10; 0,10 ]*12*2.54  + [1,1;-1,1;-1,-1;1,-1]*35.56;
```

```
47
48 %% loop
49 for i=1:r
50     figure(figHandle);
51     xlim([xmin xmax]);
52     ylim([ymin ymax]);
53
54     hold on
55     % plot obstacles
56     for u=1:numObs
57         fill(Cobs(u).vert(:,1),Cobs(u).vert(:,2),[1.0,0.5,0.5]);
58     end
59     for v=(1+numObs):(1+numObs*2)
60         for j=1:4
61             x(:,:) = Cobs(v).line(j,1,:);
62             y(:,:) = Cobs(v).line(j,2,:);
63             plot(x,y, 'k--');
64         end
65     end
66     % plot boundary
67     for j=1:4
68             x(:,:) = Cobs(v+1).line(j,1,:);
69             y(:,:) = Cobs(v+1).line(j,2,:);
70             plot(x,y, 'k-', 'LineWidth',2);
71     end
72     % plot the edges
73     [er, ec] = size(edges);
74     for m = 1:i
75         pt1 = [state.x(m),state.y(m)];
76         for n = 2:2:ec
77             val = edges(m,n);
```

```
78            if val ~= -1 && val <= i
79                pt2 = [state.x(val+1),state.y(val+1)];
80                plot([pt1(1), pt2(1)], [pt1(2), pt2(2)],'k-');
81            end
82        end
83    end
84
85    % plot the start
86    plot(state.x(1) , state.y(1), 'go', ...
87    'MarkerSize',8, ...
88    'MarkerEdgeColor','k', ...
89    'MarkerFaceColor',[0,1,0]);
90
91    % plot the rest up to this point
92    for j=2:i
93        plot(state.x(j) , state.y(j), 'bo', ...
94        'MarkerSize',8, ...
95        'MarkerEdgeColor','k', ...
96        'MarkerFaceColor',[0,0,1]);
97    end
98
99    % plot the goal
100   plot(Goal(1) , Goal(2), 'go', ...
101   'MarkerSize',8, ...
102   'MarkerEdgeColor','k', ...
103   'MarkerFaceColor',[1,0,1]);
104
105   hold off
106   M(frameCount) = getframe;
107   clf(figHandle);
108   frameCount = frameCount + 1;
```

```
109 end
110 % plot path
111  figure(figHandle);
112 xlim([xmin xmax]);
113 ylim([ymin ymax]);
114 hold on
115     % plot obstacles
116     for u=1:numObs
117         fill(Cobs(u).vert(:,1),Cobs(u).vert(:,2),[1.0,0.5,0.5]);
118     end
119     for v=(1+numObs):(1+numObs*2)
120         for j=1:4
121             x(:,:) = Cobs(v).line(j,1,:);
122             y(:,:) = Cobs(v).line(j,2,:);
123             plot(x,y, 'k--');
124         end
125     end
126     % plot boundary
127     for j=1:4
128             x(:,:) = Cobs(v+1).line(j,1,:);
129             y(:,:) = Cobs(v+1).line(j,2,:);
130             plot(x,y, 'k-', 'LineWidth',2);
131     end
132     % plot the edges
133     [er, ec] = size(edges);
134     for m = 1:r
135         pt1 = [state.x(m),state.y(m)];
136         for n = 2:2:ec
137             val = edges(m,n);
138             if val ~= -1 && val <= r
139                 pt2 = [state.x(val+1),state.y(val+1)];
```

```
                plot([pt1(1), pt2(1)], [pt1(2), pt2(2)],'k-');
            end
        end
    end

    % plot the rest up to this point
    for j=2:i
        plot(state.x(j) , state.y(j), 'bo', ...
        'MarkerSize',8, ...
        'MarkerEdgeColor','k', ...
        'MarkerFaceColor',[0,0,1]);
    end
    % plot the start
    plot(state.x(1) , state.y(1), 'go', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[0,1,0]);

    % plot the goal
    plot(Goal(1) , Goal(2), 'go', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1,0,1]);

for i=2:length(opt_path)-1
    plot(state.x(opt_path(i)+1) , state.y(opt_path(i)+1), 'go', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1,1,0]);
end
hold off
```

```
M(frameCount) = getframe;
clf(figHandle);
frameCount = frameCount + 1;

figure(figHandle)
xlim([xmin xmax]);
ylim([ymin ymax]);
xlabel('X [cm]')
ylabel('Y [cm]')
title('RRT Growth')
movie(M,1);

if dosave
    vid = VideoWriter(fname, 'MPEG-4');
    vid.FrameRate = 15;
    open(vid)
    writeVideo(vid,M)
    close(vid)
end

end
```

Listing 47: Function that animates the growth of the mobile base RRT

```
numObs = 2;
t_cnv = 12*2.54;
t_agent = 35.56;
Cobs(1).vert = [ 4,0;
                 5,0;
                 5,2;
                 4,2 ]*t_cnv;
v = Cobs(1).vert;
```

```
Cobs(1).line(1,:,:) = [ v(1,1),v(2,1) ; v(1,2),v(2,2) ];
Cobs(1).line(2,:,:) = [ v(2,1),v(3,1) ; v(2,2),v(3,2) ];
Cobs(1).line(3,:,:) = [ v(3,1),v(4,1) ; v(3,2),v(4,2) ];
Cobs(1).line(4,:,:) = [ v(4,1),v(1,1) ; v(4,2),v(1,2) ];

Cobs(2).vert = [  7,6;
                  9,6;
                  9,10;
                   7,10 ]*t_cnv;
v = Cobs(2).vert;
Cobs(2).line(1,:,:) = [ v(1,1),v(2,1) ; v(1,2),v(2,2) ];
Cobs(2).line(2,:,:) = [ v(2,1),v(3,1) ; v(2,2),v(3,2) ];
Cobs(2).line(3,:,:) = [ v(3,1),v(4,1) ; v(3,2),v(4,2) ];
Cobs(2).line(4,:,:) = [ v(4,1),v(1,1) ; v(4,2),v(1,2) ];

Cobs(3).vert = Cobs(1).vert + [ -t_agent,0; t_agent,0; t_agent,t_agent;
    -t_agent,t_agent];
v = Cobs(3).vert;
Cobs(3).line(1,:,:) = [ v(1,1),v(2,1) ; v(1,2),v(2,2) ];
Cobs(3).line(2,:,:) = [ v(2,1),v(3,1) ; v(2,2),v(3,2) ];
Cobs(3).line(3,:,:) = [ v(3,1),v(4,1) ; v(3,2),v(4,2) ];
Cobs(3).line(4,:,:) = [ v(4,1),v(1,1) ; v(4,2),v(1,2) ];

Cobs(4).vert = Cobs(2).vert + [ -t_agent,-t_agent; t_agent,-t_agent;
    t_agent,0; -t_agent,0];
v = Cobs(4).vert;
Cobs(4).line(1,:,:) = [ v(1,1),v(2,1) ; v(1,2),v(2,2) ];
Cobs(4).line(2,:,:) = [ v(2,1),v(3,1) ; v(2,2),v(3,2) ];
Cobs(4).line(3,:,:) = [ v(3,1),v(4,1) ; v(3,2),v(4,2) ];
Cobs(4).line(4,:,:) = [ v(4,1),v(1,1) ; v(4,2),v(1,2) ];

```

```
Cobs(5).vert = [   0,0;
                  14,0;
                  14,10;
                   0,10 ]*t_cnv;
Cobs(5).vert = Cobs(5).vert + [ t_agent,t_agent; -t_agent,t_agent; -
    t_agent,-t_agent; t_agent,-t_agent];
v = Cobs(5).vert;
Cobs(5).line(1,:,:) = [ v(1,1),v(2,1) ; v(1,2),v(2,2) ];
Cobs(5).line(2,:,:) = [ v(2,1),v(3,1) ; v(2,2),v(3,2) ];
Cobs(5).line(3,:,:) = [ v(3,1),v(4,1) ; v(3,2),v(4,2) ];
Cobs(5).line(4,:,:) = [ v(4,1),v(1,1) ; v(4,2),v(1,2) ];

Cobs(6).vert = Cobs(5).vert - [ t_agent,t_agent; -t_agent,t_agent; -
    t_agent,-t_agent; t_agent,-t_agent];
v = Cobs(6).vert;
Cobs(6).line(1,:,:) = [ v(1,1),v(2,1) ; v(1,2),v(2,2) ];
Cobs(6).line(2,:,:) = [ v(2,1),v(3,1) ; v(2,2),v(3,2) ];
Cobs(6).line(3,:,:) = [ v(3,1),v(4,1) ; v(3,2),v(4,2) ];
Cobs(6).line(4,:,:) = [ v(4,1),v(1,1) ; v(4,2),v(1,2) ];
```

Listing 48: Script that ”builds” the configuration obstacle structures

```
function scenario = mobile_config(fname)
data = csvread(fname, 0, 1);
[r,c] = size(data);
scenario.x_lim = data(1,1:2);
scenario.y_lim = data(2,1:2);
scenario.agent_rad = data(3,1);
delta = scenario.agent_rad*[
    -1,-1,1,-1,1,1,-1,1;
    -1,-1,1,-1,1,1,-1,1;
    -1,-1,1,-1,1,1,-1,1;
```

```
    -1,-1,1,-1,1,1,-1,1];
scenario.obs = data(4:7,2:c);
scenario.cobs = scenario.obs+delta;

scenario.nodes = data(8:r,2:4);
end
```

Listing 49: Function that loads the mobile base configuration file

```
clear;clc;
scenario = mobile_config('C:\Users\zhawks\Documents\c++\RRTplanner1-0\
    RRTplanner\_config.csv');
%Actions = csvread('action_list.csv');

data = csvread('../mobilepath.csv', 1,0);
scenario.startId = 2;
scenario.goalId = 3;
mobile_plot_graph(5,data,scenario);

data1 = csvread('../mobilepath1.csv', 1,0);
scenario.startId = 1;
scenario.goalId = 2;
mobile_plot_graph(1,data1,scenario);

data2 = csvread('../mobilepath2.csv', 1,0);
scenario.startId = 2;
scenario.goalId = 3;
mobile_plot_graph(2,data2,scenario);

data3 = csvread('../mobilepath3.csv', 1,0);
scenario.startId = 3;
scenario.goalId = 4;
```

```
mobile_plot_graph(3,data3,scenario);
%M = animateRRTpath(1,scenario,data,Goal,true, 'ani__seed_101_nodes_100.
    mp4');

%[Trajectory,error] = robotTrajectory(0.05,127,Start,Goal,Actions);
%error
%S = simulation(2,scenario,Start,Goal,Trajectory,true,'
    sim__seed_101_nodes_100.mp4');
```

Listing 50: Script that executes the mobile base RRT visualization

```
function mobile_plot_graph(fig, data,S)
figure(fig);

goal = S.nodes(S.goalId,:);
goalIsConnected = false;
goalId = -1;
id = data(:,1);
state.x = data(:,2);
state.y = data(:,3);
state.th = data(:,4);

epsilon = 0.0001;

% determine edges
[r,c] = size(data);
idx_p = r;
r = r-1;
edges = zeros(r,c-4);
for i = 1:r
    for j = 1:c-4
        edges(i,j) = -1;
```

```
22      end
23      delta = [abs(state.x(i) - goal(1));abs(state.y(i) - goal(2));abs(
        state.th(i) - goal(3))];
24      if (delta(1) <= epsilon &&...
25          delta(2) <= epsilon &&...
26          delta(3) <= epsilon )
27         goalIsConnected = true;
28         goalId = id(i) + 1;
29      end
30  end
31
32  opt_path = data(idx_p,:);
33
34  for i=1:r
35      j = 5;
36      while (j < c && data(i,j) ~= 0)
37          edges(i,j-4) = data(i,j);
38          edges(i,j+1-4) = data(i, j+1);
39          j = j + 2;
40      end
41  end
42  hold on
43  plot(state.x(1) , state.y(1), 'go', ...
44      'MarkerSize',8, ...
45      'MarkerEdgeColor','k', ...
46      'MarkerFaceColor',[0,1.0,0]);
47  plot(state.x(2:r), state.y(2:r), 'bo', ...
48      'MarkerSize',8, ...
49      'MarkerEdgeColor','k', ...
50      'MarkerFaceColor',[0.5,0.5,0.5]);
51  plot(goal(1), goal(2), 'ro', ...
```

```
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,0,0]);
plot(goal(1), goal(2), 'ro', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,1.0,0]);


% plot Cobs
%patch('Faces', F, 'Vertices', V1, 'FaceColor', [0 0 1], 'EdgeColor',
    [0,0,0], 'LineStyle', '-.', 'FaceAlpha', 0.4);
V = [S.obs(1,1),S.obs(1,2);S.obs(1,3),S.obs(1,4);S.obs(1,5),S.obs(1,6);S
    .obs(1,7),S.obs(1,8)];
patch('Faces',[1,2,3,4], 'Vertices', V, ...
    'FaceColor', [0,0,1], 'EdgeColor',[0,0,0], 'LineStyle', '-', '
    FaceAlpha', 0.6);
V = [S.cobs(1,1),S.cobs(1,2);S.cobs(1,3),S.cobs(1,4);S.cobs(1,5),S.cobs
    (1,6);S.cobs(1,7),S.cobs(1,8)];
patch('Faces',[1,2,3,4], 'Vertices', V, ...
    'FaceColor', [0,0,1], 'EdgeColor',[0,0,0], 'LineStyle', '--', '
    FaceAlpha', 0.2);

% plot boundary
x1 = S.x_lim(1); x2 = S.x_lim(2);
y1 = S.y_lim(1); y2 = S.y_lim(2);
patch('Faces',[1,2,3,4], 'Vertices', [x1,y1;x2,y1;x2,y2;x1,y2], ...
    'FaceColor', [1,1,1], 'EdgeColor',[0,0,0], 'LineStyle', '-', '
    FaceAlpha', 0.0);
x1 = S.x_lim(1); x2 = S.x_lim(2);
y1 = S.y_lim(1); y2 = S.y_lim(2);
```

```
rad = S.agent_rad;
patch('Faces',[1,2,3,4], 'Vertices', [x1+rad,y1+rad;x2-rad,y1+rad;x2-rad
    ,y2-rad;x1+rad,y2-rad], ...
    'FaceColor', [1,1,1], 'EdgeColor',[0,0,0], 'LineStyle', '--', '
    FaceAlpha', 0.0);
% plot edges
[er, ec] = size(edges);
for i = 1:er
    pt1 = [state.x(i),state.y(i)];
    for j = 2:2:ec
        val = edges(i,j);
        if val ~= -1 && val < r
            pt2 = [state.x(val+1),state.y(val+1)];
            plot([pt1(1), pt2(1)], [pt1(2), pt2(2)],'k-');
        end
    end
end

plot(state.x(2:r), state.y(2:r), 'bo', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[0.5,0.5,0.5]);

% plot path
for i=1:length(opt_path)
    plot(state.x(opt_path(i)+1) , state.y(opt_path(i)+1), 'go', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,1.0,0]);
end

```

```
106 plot(state.x(1) , state.y(1), 'go', ...
107     'MarkerSize',8, ...
108     'MarkerEdgeColor','k', ...
109     'MarkerFaceColor',[0,1.0,0]);
110
111 plot(goal(1), goal(2), 'ro', ...
112     'MarkerSize',8, ...
113     'MarkerEdgeColor','k', ...
114     'MarkerFaceColor',[1.0,0,0]);
115
116
117 % plot Cobs
118 for i = 1:4
119 V = [S.obs(i,1),S.obs(i,2);S.obs(i,3),S.obs(i,4);S.obs(i,5),S.obs(i,6);S
    .obs(i,7),S.obs(i,8)];
120 patch('Faces',[1,2,3,4], 'Vertices', V, ...
121     'FaceColor', [0,0,1], 'EdgeColor',[0,0,0], 'LineStyle', '-', '
    FaceAlpha', 0.6);
122 V = [S.cobs(i,1),S.cobs(i,2);S.cobs(i,3),S.cobs(i,4);S.cobs(i,5),S.cobs(
    i,6);S.cobs(i,7),S.cobs(i,8)];
123 patch('Faces',[1,2,3,4], 'Vertices', V, ...
124     'FaceColor', [0,0,1], 'EdgeColor',[0,0,0], 'LineStyle', '--', '
    FaceAlpha', 0.2);
125 end
126
127 if (goalIsConnected)
128     title(sprintf('RRT: %d nodes - Goal Id %d',r,goalId));
129 else
130     title(sprintf('RRT: %d nodes',r))
131 end
132
```

```
133 xlabel('Position X [cm]')
134 ylabel('Position Y [cm]')
135
136 legend('Start', 'Nodes', 'Goal','Path',...
137        'Obs_{task}','Obs_{c-space}', 'Location', 'eastoutside');
138
139 xlim(S.x_lim)
140 ylim(S.y_lim)
141 hold off
142 end
```

Listing 51: Function that plots the mobile base RRT

```
1  clear;clc;
2  cnv = 12*2.54;
3  goal = [1.5, 1.5]*cnv;
4  start = [11.0, 8.0]*cnv;
5  xmax = 15*cnv; xmin = -1*cnv;
6  ymax = 11*cnv; ymin = -1*cnv;
7
8  build_config_obstacles2;
9
10 figure(1);
11 hold on
12 plot(start(1) , start(2), 'go', ...
13     'MarkerSize',8, ...
14     'MarkerEdgeColor','k', ...
15     'MarkerFaceColor',[0,1.0,0]);
16 plot(goal(1), goal(2), 'ro', ...
17     'MarkerSize',8, ...
18     'MarkerEdgeColor','k', ...
19     'MarkerFaceColor',[1.0,0,1.0]);
```

```
20 if (numObs > 0)
21     fill(Cobs(1).vert(:,1),Cobs(1).vert(:,2),[1.0,0.5,0.5]);
22 end
23 for j=1:4
24     x(:,:) = Cobs(1+numObs).line(j,1,:);
25     y(:,:) = Cobs(1+numObs).line(j,2,:);
26     plot(x,y, 'k—');
27 end
28 % plot Cobs
29 for i=1:numObs
30 fill(Cobs(i).vert(:,1),Cobs(i).vert(:,2),[1.0,0.5,0.5]);
31 end
32
33 for i=(1+numObs):(1+numObs*2)
34     for j=1:4
35         x(:,:) = Cobs(i).line(j,1,:);
36         y(:,:) = Cobs(i).line(j,2,:);
37         plot(x,y, 'k—');
38     end
39 end
40 % plot boundary
41 for j=1:4
42         x(:,:) = Cobs(i+1).line(j,1,:);
43         y(:,:) = Cobs(i+1).line(j,2,:);
44         plot(x,y, 'k-', 'LineWidth',2);
45 end
46
47 title(sprintf('Configuration Space'))
48 xlabel('X [cm]')
49 ylabel('Y [cm]')
50 legend('start', 'goal', 'Obs_t_a_s_k','Obs_c_-_s_p_a_c_e', 'Location', '
```

```
    eastoutside');
axis([xmin xmax ymin ymax])
hold off
```

Listing 52: Script that plots the mobile base configuration space obstacles

```
clear;clc;
cnv = 12*2.54;
goal = [1.5, 1.5]*cnv;
start = [11.0, 8.0]*cnv;
xmax = 15*cnv; xmin = -1*cnv;
ymax = 11*cnv; ymin = -1*cnv;

build_config_obstacles2;

figure(1);
hold on
plot(start(1), start(2), 'go', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[0,1.0,0]);
plot(goal(1), goal(2), 'ro', ...
    'MarkerSize',8, ...
    'MarkerEdgeColor','k', ...
    'MarkerFaceColor',[1.0,0,1.0]);
% plot Cobs
for i=1:numObs
fill(Cobs(i).vert(:,1),Cobs(i).vert(:,2),[1.0,0.5,0.5]);
end

for i=(1+numObs):(1+numObs*2)
%     for j=1:4
```

```
%          x(:,:) = Cobs(i).line(j,1,:);
%          y(:,:) = Cobs(i).line(j,2,:);
%          plot(x,y, 'k--');
%      end
end
% plot boundary
for j=1:4
         x(:,:) = Cobs(i+1).line(j,1,:);
         y(:,:) = Cobs(i+1).line(j,2,:);
         plot(x,y, 'k-', 'LineWidth',2);
end

title(sprintf('Task Space'))
xlabel('X [cm]')
ylabel('Y [cm]')
legend('start', 'goal', 'obstacles', 'Location', 'eastoutside');
axis([xmin xmax ymin ymax])
hold off
```

Listing 53: Script that plots the mobile base task space obstacles

```
function M = simulation(fig, scenario, Start, Goal, Trajectory, dosave,
    fname)
if nargin < 7
    fname = 'simulation.mp4';
elseif nargin < 6
    dosave = false;
    fname = 'blank_sim.mp4';
end

figHandle = figure(fig);
frameCount = 1;
```

```
set(figHandle, 'units', 'inches', 'pos', [0 0 15 11.5]);

if scenario == 1
build_config_obstacles;
elseif scenario == 2
build_config_obstacles2;
elseif scenario == 0
build_config_obstaclesNone;
end
cnv = 12*2.54;
xmax = 15*cnv; xmin = -1*cnv;
ymax = 11*cnv; ymin = -1*cnv;
%%
figure(figHandle);
xlim([xmin xmax]);
ylim([ymin ymax]);
hold on
plotRobot(Start,[0,0,1]);
hold off
[r,c] = size(Trajectory);
%% loop
for i=1:r
    figure(figHandle);
    xlim([xmin xmax]);
    ylim([ymin ymax]);

    hold on
    % plot obstacles
    for u=1:numObs
        fill(Cobs(u).vert(:,1),Cobs(u).vert(:,2),[1.0,0.5,0.5]);
    end
```

```
42      for v=(1+numObs):(1+numObs*2)
43          for j=1:4
44              x(:,:) = Cobs(v).line(j,1,:);
45              y(:,:) = Cobs(v).line(j,2,:);
46              plot(x,y, 'k--');
47          end
48      end
49      % plot boundary
50      for j=1:4
51              x(:,:) = Cobs(v+1).line(j,1,:);
52              y(:,:) = Cobs(v+1).line(j,2,:);
53              plot(x,y, 'k-', 'LineWidth',2);
54      end
55      state = Trajectory(i,2:4);
56      plotRobot(state,[0,0,1]);
57      plotRobot(Goal,[1,0,1]);
58
59      hold off
60      M(frameCount) = getframe;
61      clf(figHandle);
62      frameCount = frameCount + 1;
63  end
64
65  figure(figHandle)
66  xlim([xmin xmax]);
67  ylim([ymin ymax]);
68  xlabel('X [cm]')
69  ylabel('Y [cm]')
70  title('Robot Path')
71  movie(M,1);
72
```

```
if dosave
    vid = VideoWriter(fname, 'MPEG-4');
    vid.FrameRate = 15;
    open(vid)
    writeVideo(vid,M)
    close(vid)
end

end
```

Listing 54: Function that simulates the mobile base executing the RRT output

```
clear;clc;
data = csvread('experiment_data.csv',1,0);
labels = {'seed','type','success','vlimit','goal_look','min_nodes','
    num_nodes','num_misses','t_RRT','t_A*','pathNodes','pathDist','
    thetaDist','thetaOpt'};
type{1} = data(1:20,:);
type{2} = data(21:40,:);
type{3} = data(41:60,:);

for i=1:3
    d = type{i};
    mu(i,:) = mean(d);
    sig(i,:) = std(d);
end
```

Listing 55: Script that analyzes the statistics from the RRT experiments

Bibliography

[1] S. Verma, P. Gonthina, Z. Hawks, D. Nahar, J. O. Brooks, I. D. Walker, Y. Wang, C. de Aguiar, and K. E. Green, "Design and evaluation of two robotic furnishings partnering with each other and their users to enable independent living," in *Proceedings of the 12th EAI International Conference on Pervasive Computing Technologies for Healthcare*. ACM, 2018, pp. 35–44.

[2] P. S. Gonthina, A. D. Kapadia, I. S. Godage, and I. D. Walker, "Modeling variable curvature parallel continuum robots using euler curves," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, p. to appear.

[3] D. Trivedi, C. D. Rahn, W. M. Kier, and I. D. Walker, "Soft robotics: Biological inspiration, state of the art, and future research," *Applied bionics and biomechanics*, vol. 5, no. 3, pp. 99–117, 2008.

[4] R. J. Webster III and B. A. Jones, "Design and kinematic modeling of constant curvature continuum robots: A review," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1661–1683, 2010.

[5] J. Burgner-Kahrs, D. C. Rucker, and H. Choset, "Continuum robots for medical applications: A survey," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1261–1280, 2015.

[6] I. D. Walker, H. Choset, and G. S. Chirikjian, "Snake-like and continuum robots," in *Springer Handbook of Robotics*. Springer, 2016, ch. 20, pp. 481–498.

[7] M. M. Tonapi, I. S. Godage, A. Vijaykumar, and I. D. Walker, "A novel continuum robotic cable aimed at applications in space," *Advanced Robotics*, vol. 29, no. 13, pp. 861–875, 2015.

[8] M. Csencsits, B. A. Jones, W. McMahan, V. Iyengar, and I. D. Walker, "User interfaces for continuum robot arms," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3123–3130.

[9] C. G. Frazelle, A. Kapadia, and I. Walker, "Developing a kinematically similar master device for extensible continuum robot manipulators," *Journal of Mechanisms and Robotics*, vol. 10, no. 2, p. 025005, 2018.

[10] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[11] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[12] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, no. 2, 2010.

[13] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[14] L. A. Lyons, R. J. Webster, and R. Alterovitz, "Planning active cannula configurations through tubular anatomy," in *2010 IEEE international conference on robotics and automation*. IEEE, 2010, pp. 2082–2087.

[15] A. D. Marchese, R. K. Katzschmann, and D. Rus, "Whole arm planning for a soft and highly compliant 2d robotic manipulator," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 554–560.

[16] J. Xiao and R. Vatcha, "Real-time adaptive motion planning for a continuum manipulator," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 5919–5926.

[17] J. Li, Z. Teng, J. Xiao, A. Kapadia, A. Bartow, and I. Walker, "Autonomous continuum grasping," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 4569–4576.

[18] M. Neumann and J. Burgner-Kahrs, "Considerations for follow-the-leader motion of extensible tendon-driven continuum robots," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 917–923.

[19] L. G. Torres and R. Alterovitz, "Motion planning for concentric tube robots using mechanics-based models," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 5153–5159.

[20] L. G. Torres, C. Baykal, and R. Alterovitz, "Interactive-rate motion planning for concentric tube robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1915–1921.

[21] L. G. Torres, A. Kuntz, H. B. Gilbert, P. J. Swaney, R. J. Hendrick, R. J. Webster, and R. Alterovitz, "A motion planning approach to automatic obstacle avoidance during concentric tube robot teleoperation," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2361–2367.

[22] A. Kuntz, A. W. Mahoney, N. E. Peckman, P. L. Anderson, F. Maldonado, R. J. Webster, and R. Alterovitz, "Motion planning for continuum reconfigurable incisionless surgical parallel robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6463–6469.

[23] C. Bergeles and P. E. Dupont, "Planning stable paths for concentric tube robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3077–3082.

[24] J. Xu, V. Duindam, R. Alterovitz, and K. Goldberg, "Motion planning for steerable needles in 3d environments with obstacles using rapidly-exploring random trees and backchaining," in *2008 IEEE international conference on automation science and engineering*. IEEE, 2008, pp. 41–46.

[25] Z. Hawks, C. Frazelle, K. E. Green, and I. D. Walker, "Motion planning for a continuum robotic mobile lamp: Defining and navigating the configuration space," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, p. submitted.

[26] W. Felt, M. J. Telleria, T. F. Allen, G. Hein, J. B. Pompa, K. Albert, and C. D. Remy, "An inductance-based sensing system for bellows-driven continuum joints in soft robots," *Autonomous Robots*, vol. 43, no. 2, pp. 435–448, 2019.

[27] M. W. Hannan and I. D. Walker, "Kinematics and the implementation of an elephant's trunk manipulator and other continuum style robots," *Journal of robotic systems*, vol. 20, no. 2, pp. 45–63, 2003.

[28] J. Siek, A. Lumsdaine, and L.-Q. Lee, *The boost graph library: user guide and reference manual*. Addison-Wesley, 2002.