© 2021 Oyuna Angatkina

DESIGN AND CONTROL OF AN ORIGAMI-ENABLED SOFT CRAWLING AUTONOMOUS ROBOT (OSCAR)

BY

OYUNA ANGATKINA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Andrew Alleyne, Chair
Assistant Professor Aimy Wissa
Associate Professor Sameh Tawfick
Professor Elizabeth Hsiao-Wecksler
Associate Professor Girish Chowdhary

Abstract

Soft mobile robots offer unique benefits as they are highly adaptable to the terrain of travel and safe for interaction with humans. However, the lack of autonomy currently limits their practical applications. Autonomous navigation has been well studied for conventional rigid-bodied robots; however, it is underrepresented in the soft mobile robot research community. Its implementation in soft robots comes with multiple challenges. However, the major challenge is the significant motion uncertainties due to the robot compliance, ground interactions, and limited available sensing. These uncertainties prevent high-level control implementation, such as autonomous navigation, to be performed successfully. Therefore, soft robots require robust design methods, as well as path following and path planning algorithms, to mitigate these uncertainties and enable autonomy.

This dissertation develops and implements autonomous navigation for a novel origami-enabled soft crawling autonomous robot called OSCAR. In order to implement autonomous navigation, it first mitigates the OSCAR's motion uncertainties by a multi-step iterative design process. Analysis has shown that OSCAR's motion uncertainties are the result of: (i) the ground-feet interaction, (ii) effectiveness of low-level closed-loop control and, (iii) variability in the manufacturing assembly process. The iterative control-oriented design allows a robust and reliable OSCAR performance and enables high-level path following control implementation. To design and implement path following control, this research presents an idealized kinematic model and introduces an empirically based correction to make the model predictions match the experimental data. The dissertation investigates two separate path following controllers: a model-based pure pursuit and a feedback controller. The controllers are investigated in both simulation and experiment and the need for feedback is clearly demonstrated. Finally, this research presents the path planning in order to complete OSCAR’s autonomous navigation. The simulation and experimental results show that OSCAR can accurately navigate in 2D environment, while avoiding static obstacles. Lastly, the coupled locomotion of multiple OSCARs demonstrates an extension of functionality and expands the potential design and operation space for this promising type of soft robot.

To my family, friends, and teachers.

Acknowledgements

First, I would like to thank my advisors Dr. Andrew Alleyne and Dr. Aimy Wissa, for their invaluable and continuing commitment to help me achieve my greatest potential. I consider myself extremely lucky for working and learning from them. They both provided unique expertise that was fundamental to the success of this research. I will always be thankful to Dr. Alleyne for the priceless lessons he taught me that I will take with me throughout my career. I will always be thankful to Prof. Wissa for her confidence, encouragement to hard work, and mentorship. I cannot express how grateful I am, as both of my advisors went above and beyond to help me grow as a researcher. I would also like to acknowledge the members of my doctoral examination committee, Dr. Sameh Tawfick, Dr. Elizabeth Hsiao-Wecksler, and Dr. Girish Chowdhary, who have helped guide and strengthen my research.

I would like to thank my parents for their support, hard work and dedication. You are my main motivation to grow. Thank you for always encouraging me to believe in myself and follow my dreams. Without you, I would not be where I am today.

These acknowledgments would not be complete without thanking my colleagues at the Alleyne Research Group (ARG) and Bio-inspired Adaptive Morphology Laboratory (BAM Lab). It was a privilege for me to work alongside incredibly talented, motivating, and hardworking lab mates. Thank you to Justin, Matt, Herschel, Bryan, Nate, Malia, Spencer, Chris, Ashley, and Pamela for creating a collaborative and hardworking atmosphere. All of you have been a source of inspiration for me and kept me motivated through the graduate school. Thank you to Kimberly for assistance and contribution to this research. Thank you to Mihary, Ophelia, Chengfang, Valeria, Paul, and Diaa for making my graduate school experience so enjoyable. I would also like to especially thank Daniel Block for his help and contribution to building OSCAR.

Table of Contents

LIST OF FIGURES xi

LIST OF TABLES xvii

LIST OF SYMBOLS xviii

CHAPTER 1 INTRODUCTION 1

1.1 Motivation and Background 1

1.1.1 Soft Mobile Robots 2

1.1.2 Origami-Enabled Mobile Robots 3

1.2 Research Objectives 4

1.2.1 Problem Statement 4

1.2.2 Dissertation Scope 6

1.3 Organization of Dissertation 7

CHAPTER 2 OSCAR CONTROL-ORIENTED DESIGN 9

2.1 Design Challenges 9

2.2 OSCAR 11

2.3 Origami Towers 13

2.4 Robot Locomotion and Iterative Feet Design 15

2.4.1 Robot Crawling Locomotion 15

2.4.2 Iterative Feet Design 16

2.4.3 Stabilizers 19

2.5 Motivation for the Assembly Guide and Analysis of Low-Level Closed-Loop Control 20

2.6 OSCAR's Assembly Guide 22
2.6.1 Assembly Guide 22
2.6.2 OSCAR's Assembly Process 24
2.7 Chapter Summary 26
CHAPTER 3 EXPERIMENTAL TESTBED 27
3.1 Testbed Components 27
3.2 Image Processing Software 31
3.3 Testbed Operation 33
3.4 OSCAR Low-Level Servo Position Control 34
3.4.1 Proportional-Integral (PI) Servo Position Controller 35
3.4.2 Motion Symmetry by the Low-Level Controllers 37
CHAPTER 4 OSCAR'S KINEMATIC MODEL 40
4.1 Kinematic Model 40
4.1.1 Kinematic Model Overview 40
4.1.2 Kinematic Model Idealized Assumptions 42
4.1.3 Lumped Kinematic Submodel (LKS) 43
4.1.4 Segmented Kinematic Submodel (SKS) 43
4.2 Kinematic Model Validation 45
CHAPTER 5 PATH FOLLOWING CONTROL 50
5.1 Motivation 50
5.2 Model-Based Open-Loop Pure Pursuit Controller 52
5.3 Proportional Feedback Controller 53
5.4 Simulation and Experimental Results 55
5.4.1 Controllers' Performance with the Ideal Kinematic Model 55
5.4.2 Pure Pursuit Performance with the Corrected Kinematic Model 57
5.4.3 Experimental Results for the Feedback Controller 58
5.5 Chapter Summary 60
CHAPTER 6 PATH PLANNING AND AUTONOMOUS NAVIGATION FOR

OSCAR ... 61

6.1 Motivation ... 61
6.2 Background ... 61
6.3 Hybrid A* Planner ... 64
6.4 Path Planning Results ... 67
6.4.1 Scenario 1 ... 69
6.4.2 Scenario 2 ... 69
6.5 OSCAR Autonomous Navigation Experiments ... 70
6.5.1 Scenario 1 ... 71
6.5.2 Scenario 2 ... 75
6.6 Chapter Summary ... 78

CHAPTER 7 COUPLED LOCOMOTION STRATEGY ... 80

7.1 Motivation ... 80
7.2 Proposed Coupled Locomotion Strategy ... 81
7.2.1 Coupled Locomotion Strategy ... 81
7.2.2 Docking Mechanism ... 83
7.3 Coupled Locomotion Strategy Validation ... 84
7.4 Chapter Summary ... 88

CHAPTER 8 CONCLUSIONS AND FUTURE WORK ... 89

8.1 Summary of Research Contributions ... 89
8.2 Future Work ... 92

REFERENCES ... 95

APPENDIX A LABVIEW VIS FOR THE ROBOT OPERATION ... 100

A.1 Camera calibration ... 100
A.2 Image Processing Algorithm ... 103
A.3 Robot Control Main VI Operation ... 110

APPENDIX B OSCAR LOW-LEVEL CONTROL CODE ... 113

B.1 Arduino Code ... 113

APPENDIX C ROBOT KINEMATIC MODEL 120

C.1 Robot Kinematic Model 120

C.2 InverseAnalyticalKinematic1.m 121

APPENDIX D PATH FOLLOWING SIMULATION 123

D.1 PurePursuitController.m 123

D.2 AnalyticKinematic.m 126

D.3 FeedbackController.m 127

APPENDIX E PATH PLANNING IMPLEMENTATION 129

E.1 PotentialFieldFunction.mlx 129

E.2 Planner_HybridAstar.mlx 130

E.3 PathTransform.mlx 130

APPENDIX F SUPPLEMENTARY FILES 132

List of Figures

Figure 1.1 The number of publications in the field of soft robotics in range 2010-2020, obtained from the Web of Science (http://apps.webofknowledge.com/). The field is categorized into two main areas: static manipulators and grippers, and mobile robots.... 5

Figure 1.2 OSCAR: Origami-enabled Soft Crawling Autonomous Robot.............................. 7

Figure 2.1 OSCAR's exploded view drawing showing its main components; the actual robot is shown in Fig. 2.3 10

Figure 2.2 Initial robot design called PERI: (a) main components; (b) top view. Image is adapted from [41] 10

Figure 2.3 OSCAR: a) main components; b) side view.. 11

Figure 2.4 Drawing of a servo with a magnetic encoder. The encoder sensor is mounted rigidly at the back plate (grey), and its magnetic ring is mounted at the servo horn to measure angles. To ensure sensor alignment with respect to the ring, it is installed on the acrylic mounting plate; then on the back plate, which has alignment elements 12

Figure 2.5 Origami tower: (a) at the fully contracted state; (b) at fully expanded state......... 13

Figure 2.6 Origami tower with attached connectors.. 14

Figure 2.7 OSCAR's top view showing the direction of creases in the opposite chirality origami towers 14

Figure 2.8 Origami cell expansion depicted as function of applied angle α and cell height *l*. Maximum cell's height is 15 mm and it corresponds to the input angle 0.6283 rad. Image is taken from [41] 15

Figure 2.9 OSCAR' crawling locomotion schematic 16

Figure 2.10 Timeline of the foot design evolution with description of performance improvement 17

Figure 2.11 Foot designs *D1-D3* implemented on the robot. Shown here is an intermediate robot design with bellows 18
Figure 2.12 Sliding ratchet foot: (a) low friction situational deployment during plate forward motion; (b) high friction situational deployment during the plate backward motion 19
Figure 2.13 Stabilizers: (a) functionality; (b) implementation 20
Figure 2.14 Straight line (red) path following with the robot starting at an initial offset in y-direction: (a) in open-loop; (b) with feedback path following controller. Dotted black and blue lines are robot trajectories in the correct and faulty cases; solid black and blue lines are robot orientations 21
Figure 2.15 Angular control inputs for the path following in open-loop (a) and with proportional feedback (b). Angular inputs in (a) are used in both faulty and correct performances in Fig. 2.8, a. Angular inputs for Fig. 2.8, b are shown in (b); inputs are similar for both faulty and correct performances 21
Figure 2.16 Assembly guide. The front and back plate's alignment in the X-Z plane is achieved by setting the plates' left faces against the red shaded areas, thus aligning them along the red line. Alignment in the X-Y plane is provided by contact of plates' bottom surfaces with the green shaded areas. The front and back plates are aligned parallel to each other in the Y-Z plane along the green lines. The front plate is aligned in the Y-Z plane by contact with the blue shaded area; the same holds for the back plate. Once aligned, the front plate is fixed by the fixtures in Fig. 2.17, and set screws fix the back plate. 24
Figure 2.17 Side view of fixture used for holding the front plate in the assembly guide 24
Figure 2.18 OSCAR in the assembly guide. The servo horns are centered for easy monitoring of servo positions during robot initialization 25
Figure 3.1 Schematic drawing of the experimental testbed (the PC and robot power supply are not shown) 28
Figure 3.2 Flow chart of the testbed operation 28
Figure 3.3 Experimental testbed 29
Figure 3.4 OSCAR markers (in 1:1 scale): (top) for front plate; (bottom) for back plate 30
Figure 3.5 Marker templates: (left) for front plate; (right) for back plate 32

Figure 3.6 Processed image with identified markers 32
Figure 3.7 Sequence of video frames depicting OSCAR locomotion 34
Figure 3.8 Step response of identified transfer function 35
Figure 3.9 Reference angle tracking with low-level controller: (a) with step reference input; (b) with ramp reference input 38
Figure 3.10 OSCAR displacement data collected for static angular inputs: (a) with step reference input in the low-level controller; (b) with ramp angular input in the low-level controller. The angular input values are shown in the legend, where each pair denotes the left and right servo angles, respectively 39
Figure 4.1 Kinematic model consists of two submodels: a lumped kinematic submodel (a) and a segmented kinematic submodel (b). The model on the left represents OSCAR's motion as a rigid body motion from point p_k to point p_{k+1}. The model on the right solves a vector loop corresponding to the robot at the fully expanded state. A vector $p_b p_{k+1} = \left[dx + dl \quad dy\right]^T$ in (b) is derived from (a). Based on the assumptions (A1)-(A2), OSCAR's body expands along the arc from the point p_b at the back plate to point p_{k+1}, which defines the orientation increment $d\theta$ 41
Figure 4.2 Kinematic model hierarchy. Image is taken from [41] 42
Figure 4.3 Comparison of experimental data and ideal kinematic model predictions for the same angular inputs. The angular inputs (in degrees) are listed on the right 47
Figure 4.4 Comparison of experimental data and corrected kinematic model predictions for the same angular inputs. The angular inputs (in degrees) are listed in the column on the bottom left 48
Figure 4.5 Comparison of achievable workspace for ideal kinematic model (a) and corrected kinematic model (b) 49
Figure 5.1 Pure pursuit controller schematic 52
Figure 5.2 Lateral error in the feedback controller 54
Figure 5.3 Simulation results for the path following with the ideal kinematic model 56
Figure 5.4 Robot trajectory with the pure pursuit controller with the ideal kinematic model. Due to modeling errors, OSCAR's trajectory (green) diverges from the path (red) and

simulation (black). Purple and pink solid lines show the robot orientation at the initial and final states, respectively 56
Figure 5.5 (a) Robot trajectory with the pure pursuit controller with the corrected kinematic model. Purple and pink solid lines show the robot orientation at the initial and final states, respectively. (b) Trajectory divergence for the three consecutive experiments... 57
Figure 5.6 Robot trajectory with the feedback controller: (a) combined experimental results showing the robot acquiring and following the path. The robot final orientation in experiment 1 matches the initial orientation in the experiment 2. (b) and (c) show the robot trajectory repeatability with the feedback controller for three trials in settings of experiment 1 and 2, respectively 59
Figure 5.7 Angular inputs for path following with feedback controller in Fig. 5.6(a) 60
Figure 6.1 Graphical explanation of hybrid A* path planning algorithm.............................. 64
Figure 6.2 Obstacles: (a) real size in a workspace and (b) in an occupancy map. In (b), obstacles have added clearance around them for collision avoidance purposes during the planning... 68
Figure 6.3 Autonomous navigation scenario schematics.. 68
Figure 6.4 Path planned for the first scenario from the start (green) to goal (red) configurations. Black shaded circles are the obstacles with added clearance space, and light blue lines are nodes expanded by the planner 69
Figure 6.5 Path planned for the second scenario from the start (green) to goal (red) configurations. Black shaded circles are the obstacles with added clearance space, and light blue lines are nodes expanded by the planner 70
Figure 6.6 OSCAR trajectory in simulation in part 1 of scenario 1. The red dashed lines show the robot's initial and final configurations in the reference path; a fully contracted robot state is shown. Obstacles in actual size (without clearance space) are shown in pink. 72
Figure 6.7 OSCAR trajectory in experiments in part 1 of scenario 1. The red and black dashed lines show the robot's configurations in the reference path and experiment 1. The robot body motion (grey shaded area) in experiment 1 demonstrates obstacle avoidance. Four trials show the repeatability of the results... 72

Figure 6.8 Combined simulation and experimental results for part 1 of scenario 1 73
Figure 6.9 Simulation and experimental results for OSCAR trajectory in part 2 of the scenario 1. Here, red and black dashed lines show the robot configuration in reference path and experiment 5, respectively. The robot body motion (grey shaded area) in experiment 5 demonstrates obstacle avoidance. Four trials show the repeatability of the results 73
Figure 6.10 Coupled results of OSCAR trajectory in scenario 1. The final state in experiment 1 matches the initial state of experiment 5 74
Figure 6.11 Angular inputs for part 1 in scenario 1 75
Figure 6.12 Angular inputs for part 2 in scenario 1 75
Figure 6.13 OSCAR trajectory in part 1 of the scenario 2. Here, red and black dashed lines show the robot configuration in reference path and experiment 9. The robot body motion (grey shaded area) in experiment 9 demonstrates the obstacle avoidance. Four trials show the results repeatability 76
Figure 6.14 OSCAR trajectory in part 2 of the scenario 2. Here, red and black dashed lines show the robot configuration in reference path and experiment 13. The robot body motion (grey shaded area) in experiment 13 demonstrates the obstacle avoidance. Four trials show the results repeatability 77
Figure 6.15 Coupled results of OSCAR trajectory in scenario 2. The final state in experiment 9 of part 1 matches the initial state of experiment 13 of part 2 77
Figure 6.16 Angular inputs for part 1 in scenario 2 78
Figure 6.17 Angular inputs for part 2 in scenario 2 78
Figure 7.1 (a) Schematics of the earthworm-like locomotion, adapted from [23]; (b) coupled two-segment OSCAR locomotion strategy 82
Figure 7.2 Docking mechanism: (a) front plate of segment 2 and (b) back plate of segment 1 84
Figure 7.3 Two-segment robot at the fully contracted state (top view) 85
Figure 7.4 Sequence of video frames depicting the coupled locomotion 86

Figure 7.5 Coupled robot trajectory. Here, the highlighted areas show examples of backward slippage of front plate during contraction (red boxes) and backward slippage of back plate during expansion (green boxes) 86
Figure 7.6 Displacement time history. Highlighted areas shown for two cycles only correspond to the actuation times during two alternating states: (a) segment 1 - expansion and segment 2 - contraction; (b) segment 1 - contraction and segment 2 – expansion 87
Figure A.1 Block diagram of 'Camera control.vi' 101
Figure A.2 Vision Assistant Express VI window 102
Figure A.3 Settings for calibration grid dots (left) and extracted grid dots (right)............... 103
Figure A.4 Calibration model settings (left) and workspace image (right) 103
Figure A.5 Block diagram of the image processing algorithm operations 104
Figure A.6 RGB image of workspace (left); binary image after correction and threshold steps (right) 105
Figure A.7 Image after binary inversion (left); image after applied particle filter (right).... 105
Figure A.8 Specification tab in geometric matching 106
Figure A.9 Marker templates: for front plate (left) and for back plate (right)...................... 106
Figure A.10 Processed image showing the identified markers on a black background 107
Figure A.11 Fragment of the final processed image with identified markers 108
Figure A.12 Marker settings inputs on the front panel in 'Main.vi' 109
Figure A.13 Initialization subVIs 110
Figure A.14 Dialog message for completing the robot assembly.. 111
Figure A.15 Image processing and scaling subVIs.. 111
Figure A.16 Front plate orientation calculation... 111
Figure A.17 Angular inputs the robot calculation 112

List of Tables

Table 2.1 OSCAR Electronic Components 30
Table 6.1 Obstacles in scenario 1 69
Table 6.2 Obstacles in scenario 2 70
Table 6.3 Reference paths for two parts of scenario 1.. 71
Table 6.4 Reference paths for two parts of scenario 2.. 76

List of Symbols

α	Angular input to the origami tower as a function of its height, rad
β	Angle of an origami cell's vector $\mathbf{R}$, rad
dl	Ideal OSCAR length at a fully contracted state, mm
$d\tilde{l}$	Corrected OSCAR length at a fully contracted state, mm
ds	Total displacement in one locomotion cycle, mm
$dx,\ dy,\ d\theta$	State increments denoting inrement of position (mm) and orientation (rad)
$d\tilde{x},\ d\tilde{y},\ d\tilde{\theta}$	State increments after applied model correction
k	Subscript denoting index of the current locomotion cycle
K_p	Proportional gain
l	Length of an origami cell's vector $\mathbf{R}$, mm
L	Lookahead distance, mm
N	Number of origami cells in a single tower
p_b	Centroid position of the back plate, mm
p_k	Robot state that includes front plate position and orientation, i.e., $p_k = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$
p_0	Initial OSCAR state
p_{Goal}	Final OSCAR state
R	OSCAR turn radius, mm
$R_{\min}$	Minimum OSCAR turn radius, mm
$\mathbf{R}$	Vector corresponding to two origami cells in a tower in robot vector loop

Ω	Angle along the arc of radius R and chord length ds, rad
u_x	Longitudinal control input
u_y	Lateral control input
φ_1	Angular input to the left OSCAR motor, rad
φ_2	Angular input to the riht OSCAR motor, rad
$\hat{x}$	Discrete OSCAR position
x, y, θ	OSCAR front plate position (mm) and orientation (rad)

Chapter 1
Introduction

1.1 Motivation and Background

Soft robotics is a relatively young, rapidly emerging, field of robotics that offers unique solutions that are hard to achieve with traditional rigid-bodied robots [1], [2]. In contrast to conventional robots, composed of rigid links connected at discrete joints, soft robots have compliant bodies. Compliance makes soft robots safe for close interaction with humans or the environment, thereby potentially closing the gap between robots and humans during operation. Unlike rigid-bodied robots, soft ones can deform and absorb the impact energy in case of collision, minimizing the risk of injury. This makes them prominent candidates for applications in the human-assistive and wearable technologies [3]. Moreover, body compliance allows soft robots higher adaptability to complex unstructured environments [4], better ability to grasp unknown objects [5], and improved navigation in confined spaces [6].

This dissertation is specifically interested in soft mobile robots, as their intrinsic safety and adaptability open an invaluable potential for their applications in real-world tasks, such as search-and-rescue, surveillance, in-pipe inspection, and medicine [7]. More specifically, our focus is an emerging class of origami-enabled mobile robots [8]. While preserving compliance, origami-enabled robots have improved locomotion and performance compared to more traditional hydraulically or pneumatically actuated soft mobile robots. Additionally, due to the origami fabrication process specifics, they are faster to prototype and manufacture, and their design is easily scalable and customizable.

However, the current state-of-art soft mobile robots are currently limited in practical applications. The main reason for that is the absence of task-level control, such as autonomous navigation that would enable the robots' operation in real tasks. Despite a large body of literature on soft robot designs [2], [8], due to the field novelty, there is still minimal research conducted on soft robot autonomous navigation. This fundamental challenge outlines a need for developing a soft robot navigation control framework and tools to advance the field.

1.1.1 Soft Mobile Robots

The current state-of-art in soft robotics has a vast diversity of nature-inspired designs which we can categorize into (i) stationary manipulators/grippers and (ii) mobile robots. While soft manipulators, such as the octopus-arm-inspired manipulator in [9], and the grippers in [5], exhibit great grasping adaptability and have been actively studied, we are particularly interested in soft mobile robots. Some examples of existing robots include multi-gait quadrupeds [4], [10], worm- and caterpillar-inspired robots [6],[11]–[13], and snake-inspired robots [14], [15]. Besides animal-inspired gates, some robots locomote by jumping [16], rolling [17], and growing [18]. In addition to terrestrial robots, there are underwater robots, such as a manta-ray robot [19], an octopus robot [20], and a fish robot [21], which closely resemble biological species. A more detailed review of soft robot designs at the time of writing of this thesis can be found in [2], [3], [7], [22], with the main locomotion schemes summarized in [23].

Soft mobile robot designs have proven themselves uniquely capable of adapting to tasks [4],[21], traveling across multiple uneven terrains [17], accommodating a variety of environmental conditions, e.g., snow and fire [10], and resistance to high-force mechanical damage [10],[13]. For example, the quadruped robot in [4] adapted to crawl into a narrow gap (2 cm) underneath a given obstacle in order to surpass it. The caterpillar robot (GoQBot) in [12] and fish robot in [21] demonstrated agile escape response maneuvers that are as efficient as their biological inspirations.

Soft robots are actuated by multiple sources [3] that include, but are not limited to: pneumatic, hydraulic, thermal, electric, and bio-hybrid actuation. Due to their large actuation force, high work/power density at the actuator, and fast response times, the pneumatic and hydraulic actuation schemes prevail in the field [7]. Such actuation is done in quadruped robots

[4], [10], snake robots [14], [24], and a fish robot [21] as examples. Despite pneumatic and hydraulic actuation advantages, untethered locomotion in such robots is a challenge due to the need to embed power sources on board [2]. While the actuation can be power dense, in terms of Watts/liter, these hydraulic and pneumatic systems require significant power sources to generate the high pressure fluid and these sources are typically much larger than the actuators. Therefore, although pumps and additional hardware for pressurizing working fluid can be placed on board, e.g., in [4], [21], [24], the size and weight of the overall system may limit the robots' performance [7]. Alternative solutions can be provided by other actuation sources listed above. However, they result in slower time response and a great deal of physical design complexity to embed them in the robot. In contrast, untethered actuation is not an issue for a new class of origami-enabled soft mobile robots.

1.1.2 Origami-Enabled Mobile Robots

Origami-enabled robots are an emerging class of soft mobile robots [1], [8]. Origami are complex shape 3D structures fabricated by folding from a planar, often composite, sheet. These structural systems are light-weight, easy to design, scale, and fabricate; in addition, they can achieve complex functionalities such as self-assembly [25],[26], locomotion [27], and manipulation [28]–[30]. Furthermore, origami robots' locomotion agility can potentially surpass traditional soft mobile robots [8] due to the overall lower power density requirement.

Examples of mobile origami-enabled robots include digestible robots [25], [31], milli-robots [32], worm-inspired robots [6], [33], a snake-inspired robot [27] and kirigami crawling robots [34], [35]. Kirigami robots are a subclass of origami robots, where the structure is achieved by internal cutting instead of folding. Origami mobile robots' actuation is done by a variety of means: electric servomotors, tendon-driven systems, shape-memory alloys (SMA), or external stimuli, e.g., the magnetic fields in [25], [31]. These robots can utilize multi-locomotion gaits to traverse across different terrains, as demonstrated by the milli-robot in [32]. Due to origami's scalability property, meaning the geometric shapes are independent of size, their scales vary significantly from millimeter to meter size. These robots have promising applications in minimally-invasive medical surgeries [25], [31], in-pipe inspections, and search-and-rescue missions [33]. A more comprehensive review of the origami robots can be found in [8].

1.2 Research Objectives

1.2.1 Problem Statement

Despite the recent advances in design and locomotion, control implementation in soft mobile robots comes with three significant challenges. The main challenge is ***the soft robot motion uncertainties***, which are caused by two reasons:

1. ***Robot compliance, including compliant interaction with the environment.*** Although compliance allows soft robot functionality, it also makes soft robots underactuated. This means that in addition to having *n* degrees of freedom, there are extra '*passive*' degrees of freedom which result in uncertainty, possibly significant, in the resulting robot motion [2]. Therefore, control implementation becomes more challenging for soft robots than compared to their rigid counterparts. Although motion uncertainty cannot be completely eliminated, it can be mitigated with careful design.
2. ***Lack of proprioceptive sensing and closed-loop control in soft mobile robots***. Due to their compliance, soft robots require different sensors than conventional rigid robots [3]. For example, these sensors need to be deformable to match the robot compliance [36]. As a result, soft mobile robots predominantly operate in open loop; i.e., their actuators are first characterized statically. Then, the robots locomote by executing a sequence of predetermined inputs or motion primitives. One example is the pneumatic RUBIC robot rolling locomotion in [17]. Sensor design and feedback control efforts are currently minimal in soft mobile robots. Preliminary efforts include magnetic curvature sensors for feedback control of pneumatic bending actuators in [15]. However, overall efforts in the field are not as developed as for other classes of rigid robots and without proprioceptive sensing and closed-loop actuators control it is hard to achieve precise robot motion.

Additionally, soft mobile robot models are not readily available. In well-studied traditional rigid robots, models are derived by standard methods, such as forward kinematics in rigid link robots or rigid-body analysis in mobile robots [37]. In the current state of the field, soft robot models are unique and highly dependent on the robot design and actuation type.

Finally, there is a limited number of studies on autonomous navigation implementation. From currently published research in the soft robotics field, only 25% focus on mobile robots, and less than 5% of the overall research addresses untethered gait control (Fig. 1.1). Due to a lack of readily available sensors, models, and motion uncertainties, work on autonomous navigation of soft mobile robots is highly limited.

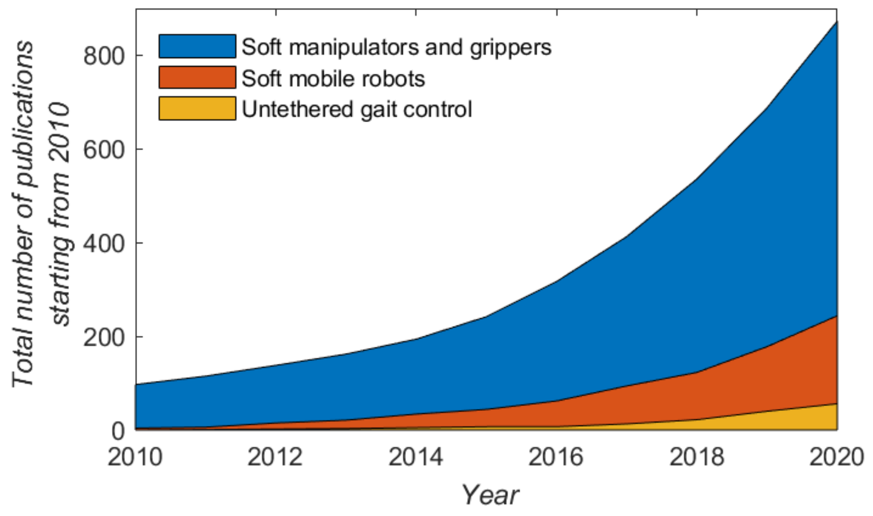


Figure 1.1 The number of publications in the field of soft robotics in range 2010-2020, obtained from the Web of Science (http://apps.webofknowledge.com/). The field is categorized into two main areas: static manipulators and grippers, and mobile robots

Although the work on autonomous navigation is limited there has been some prior autonomous navigation. One example has been demonstrated by an untethered pneumatic snake robot in [24]. This work uses a bi-directional A* planner for path planning, and iterative learning control (ILC) for path following. Although ILC demonstrates great results for repetitive tasks [38], application of ILC for path following is impractical, as the autonomous navigation naturally involves changing environments and terrains for exploration. Therefore, different control approaches for path following should be investigated. In another example, path calculation as a part of path planning has been done for obstacle-aided navigation in a soft tip-growing vine robot [39]. Despite the vine robot being a continuum robot, its application area is similar to mobile robots, which is navigation through environment with obstacles. This work focuses on precalculating joint points (buckles) for the vine robot to reach a goal position based on information about the environment. Later these joints will be mechanically made on the robot backbone before its deployment.

To summarize, autonomous navigation is currently underrepresented in the mobile soft robotics space; in several articles, it is often included only when outlining future research direction for soft mobile robotics [2], [7]. The main steps required to achieve autonomous navigation in soft robots include robust design methods, sensing, path following, and path planning.

1.2.2 Dissertation Scope

This dissertation develops an autonomous navigation framework for the novel Origami-enabled Soft Crawling Autonomous Robot (OSCAR), shown in Fig. 1.2, under the problem statements listed above. The final goal is achieved in the following five objectives, where the first four develop and experimentally validate the autonomous navigation for a single OSCAR:

1. **Control-oriented OSCAR design**. This objective considers an iterative robot design, which is a key enabler of OSCAR's autonomous navigation. As it will be shown in Chapter 2, OSCAR suffers from significant motion uncertainties caused by the feet and ground interaction, the low-level control, and the assembly process. An iterative design allows to mitigate these uncertainties and achieve a reliable robot performance. Thus, it enables the high-level path following control implementation.
2. **OSCAR kinematic model**, which is based on OSCAR's geometry. It is used in the path following control design and implementation.
3. **Path following control**, which is done by two different controllers: an adopted model-based pure pursuit and a proportional feedback controller. Both controllers are experimentally validated in the designed experimental setup, which performs robot localization.
4. **Path planning.** This objective considers the application of hybrid A* approach to planning OSCAR path. This part experimentally validates the autonomous

navigation framework for two different scenarios with the feedback path following controller.

5. **OSCAR coupled multi-segment locomotion**. This last objective extends OSCAR functionality by introducing a modular approach to the OSCAR robot concept. It studies its coupled locomotion of two coupled OSCARs. The modularity allows OSCAR to move separately, as in parts 1-4, or as a single coupled robot. Moreover, the coupled robot can reconfigure its body in the presence of faulty segments or certain types of obstacles.

The aforementioned research objectives outline the overall goal of autonomous navigation implementation for OSCAR and creates a design and control framework for other origami-enabled mobile robots.

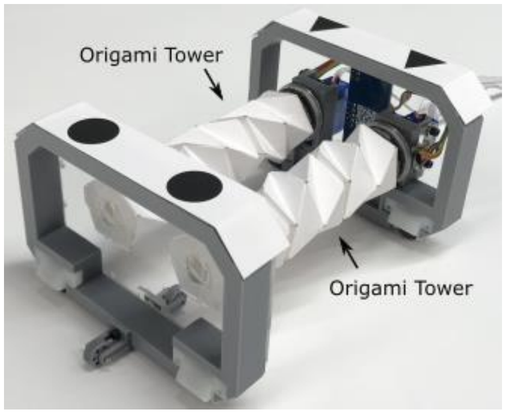


Figure 1.2 OSCAR: Origami-enabled Soft Crawling Autonomous Robot

1.3 Organization of Dissertation

The remainder of the dissertation is organized as follows. Chapters 2 and 3 present the OSCAR iterative control-oriented design process. Included in Chapter 3 is the last part of the robot design, which is OSCAR low-level closed-loop control. Also, Chapter 3 presents the experimental setup that has been used for all the experiments in this thesis. The OSCAR two-part kinematic model is presented in Chapter 4. Then, Chapters 5 and 6 develop autonomous navigation for a single robot. In particular, Chapter 5 presents the path following control, and Chapter 6 presents the path planning algorithm. Then, the autonomous navigation is experimentally validated in Chapter 6, with the controller from Chapter 5. Chapter 7 presents the coordinated navigation of a two-segment robot composed of two OSCARs. Finally, Chapter 8 provides concluding remarks and suggestions for future work.

Chapter 2
OSCAR Control-Oriented Design

2.1 Design Challenges

A major challenge preventing the development of autonomous navigation in soft mobile robots is *motion uncertainties*. This chapter presents a control-oriented design of a novel Origami-enabled Soft Autonomous Crawling Robot (OSCAR) (Fig. 2.1), which solves this problem. OSCAR's motion uncertainties result from its compliant elements, the interaction between its feet and the ground, and lack of proprioceptive sensing, as demonstrated in the initial robot design shown in Fig. 2.2 [40]–[42]. Mitigating uncertainties through an iterative robot design is one of the major contributions of this work. This was accomplished via the following processes, which are detailed in this chapter and Chapter 3:

- Iterative design of the robot feet;
- Alignment of the robot plates using an improved assembly process;
- Measuring the angular input to the origami towers using encoders;
- Closed-loop low-level servo angular position control.

As a result of following these processes, we have achieved a soft mobile robot design capable of accurately following the reference path and performing autonomous navigation as described in Chapters 5 and 6.

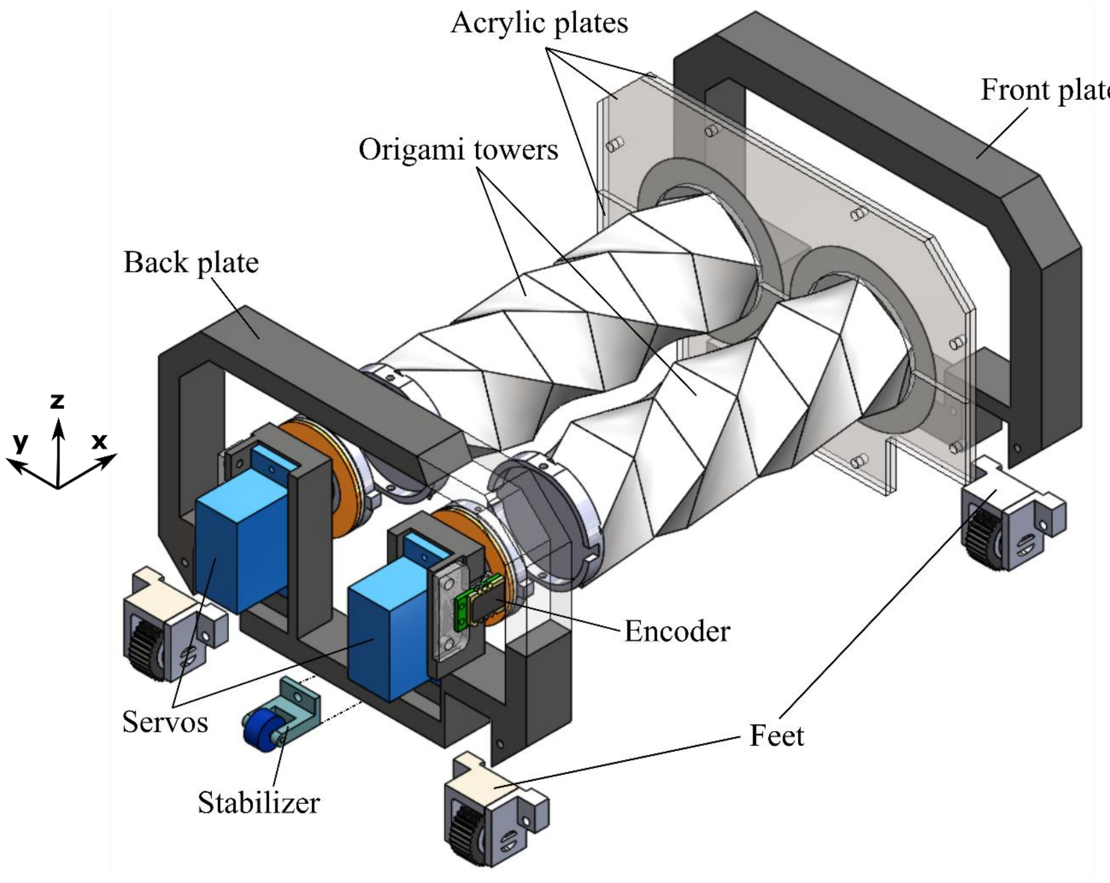


Figure 2.1 OSCAR's exploded view drawing showing its main components; the actual robot is shown in Fig. 2.3

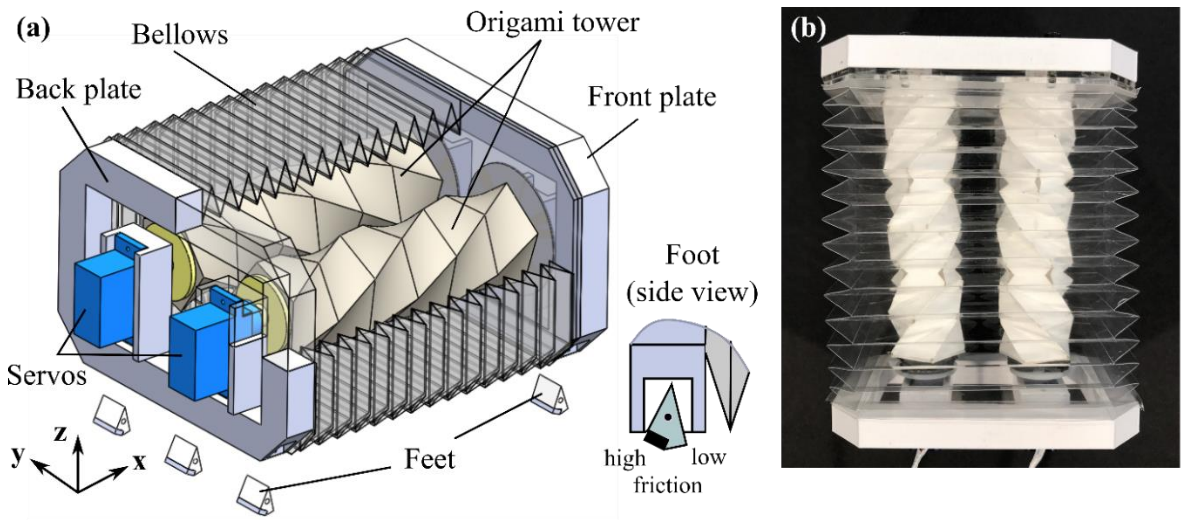


Figure 2.2 Initial robot design called PERI: (a) main components; (b) top view. Image is adapted from [41]

2.2 OSCAR

OSCAR is an origami-enabled soft crawling robot inspired by the caterpillar crawling locomotion. Its exploded view is shown in Fig. 2.1, and the actual robot is shown in Fig. 2.3. The OSCAR's main components are two opposite chirality origami towers. They act as linear actuators when given a rotational input [40]. Each origami tower consists of six identical origami cells made with a Kresling pattern [43] and each tower has relief cuts between every two consecutive cells to allow turning (Fig. 2.3, b). The direction of creases in the cell's Kresling pattern determines the tower's chirality, i.e., positive or negative.

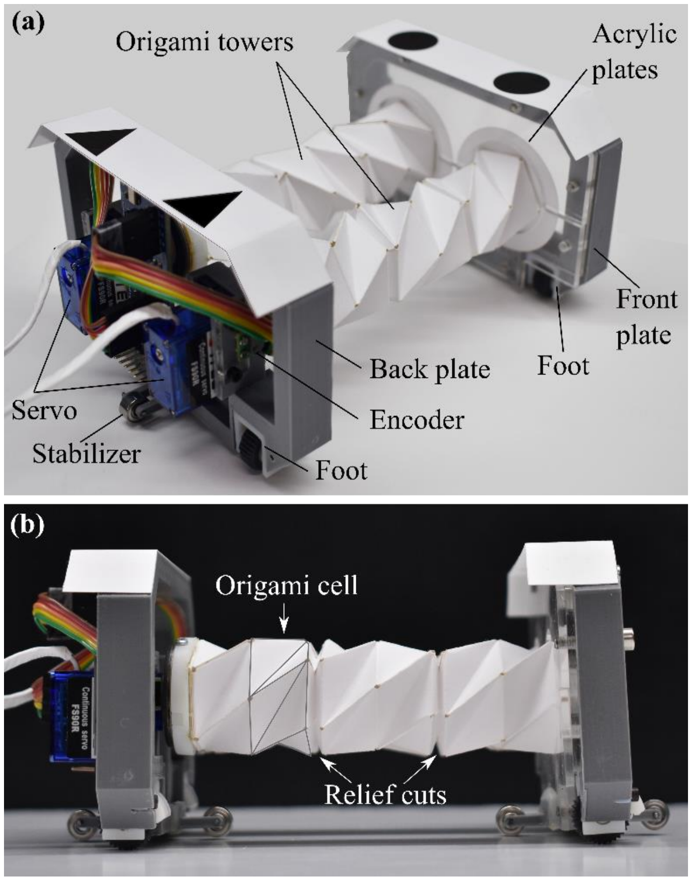


Figure 2.3 OSCAR: a) main components; b) side view

The origami towers are rigidly attached to the front plate by a set of acrylic plates from one end and are driven by servo motors from the opposite end. These are continuous rotation servos mounted at the robot back plate. The towers expand or contract with provided servo rotation, resulting in the robot body expansion or contraction. OSCAR's height and width are 72 mm and 106 mm, respectively. Its length in the fully contracted and fully expanded states is 95 mm and 155 mm, respectively. The rotational input to the origami towers is measured by magnetic encoders; see details in Fig. 2.4. Their angular resolution is 0.02 degrees.

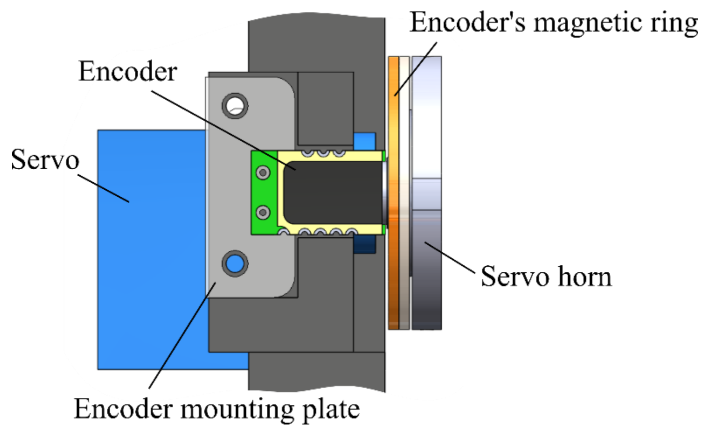


Figure 2.4 Drawing of a servo with a magnetic encoder. The encoder sensor is mounted rigidly at the back plate (grey), and its magnetic ring is mounted at the servo horn to measure angles. To ensure sensor alignment with respect to the ring, it is installed on the acrylic mounting plate; then on the back plate, which has alignment elements

In order to locomote, OSCAR utilizes anisotropic friction feet. Anisotropic friction-based locomotion is common in earthworms and caterpillars; some examples of such feet implementation in soft robots are listed in [23]. These feet provide low friction in one direction and high friction in the other in order to prevent backward slippage during locomotion [44]. When the OSCAR’s front or back plate moves forward, its feet operate under low friction. When the plate moves backward, its feet operate under high friction. Together, anisotropic friction feet and the consecutive origami towers’ expansion and contraction enable OSCAR’s crawling motion. Both OSCAR’s front and back plates are 3D printed on an Ultimaker printer with PLA material.

OSCAR can traverse a 2D plane by executing forward, left, and right turning motions. Consider servo angular inputs to the origami towers for robot expansion to be $[\varphi_1 \quad \varphi_2]^T$, where φ_1 and φ_2 are inputs to the left and right tower, respectively. When the angular inputs are equal, OSCAR moves forward. When the inputs are different, OSCAR turns. In all cases, zero angular inputs result in the robot's contraction.

In its initial design, shown in Fig. 2.2, the robot had protective plastic bellows (PET) that encased the origami towers. Its main function was to provide torsional rigidity to the robot, i.e., to prevent front and back plates from pivoting in the X-Z plane during locomotion. The robot cannot locomote without some form of torsional rigidity with respect to the ground plane. However, the bellows had a major drawback: they also produced torsional rigidity in the X-Y plane, which led to resistance to turning. This drawback has been addressed by substituting bellows with stabilizers in the final design (Fig. 2.1 and Fig. 2.3).

The following sections consider individual OSCAR components in sufficient detail to allow future readers to replicate the design. Its electronics and low-level servo position control are covered in Chapter 3.

2.3 Origami Towers

The origami towers are the OSCAR's main components, and they are made of 163 g/m^2 Neenah paper. They are fabricated by cutting crease patterns with a laser cutter (Epilog), and then folding and gluing them into their final assembled state. Figure 2.5 shows the towers in the fully contracted and fully expanded states. Each tower has a 3D printed servo horn connector at one end and a paper disk connector at the other end which attaches the tower to the front plate (Fig. 2.6). The servo connector rigidly connects the origami tower to a custom-made 3D printed servo horn by two M1.6 screws. The two opposite chirality towers are mounted in OSCAR such that their creases form a triangle directed toward the front plate from a top view (see Fig. 2.7).

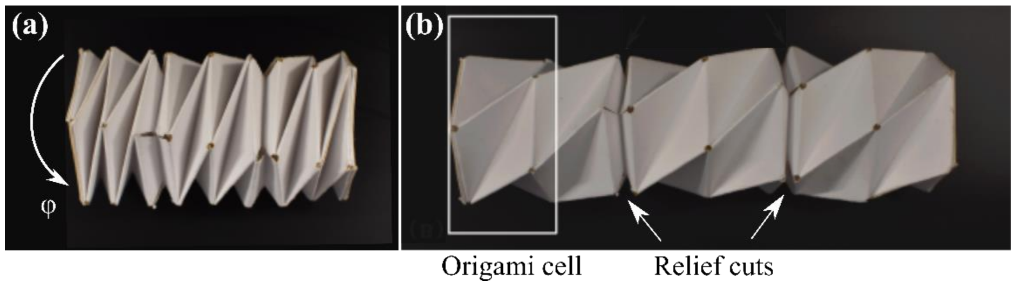


Figure 2.5 Origami tower: (a) at the fully contracted state; (b) at fully expanded state

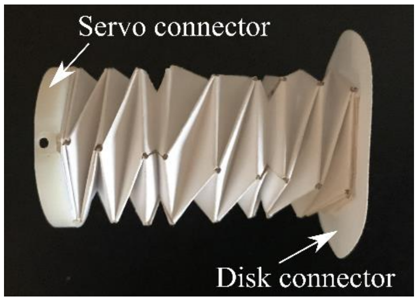


Figure 2.6 Origami tower with attached connectors

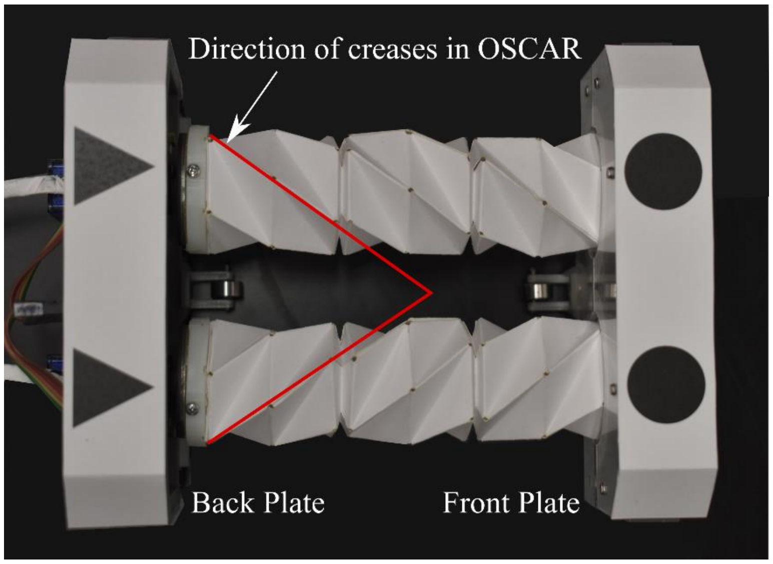


Figure 2.7 OSCAR's top view showing the direction of creases in the opposite chirality origami towers

Each origami Kresling cell has a 5-sided polygon at its base. Applying a rotational input causes the cell to expand according to the function presented in Fig. 2.8. This function represents the relationship between the angular input to the cell and its height. The function has been numerically calculated based on the origami cell's geometry in [45].

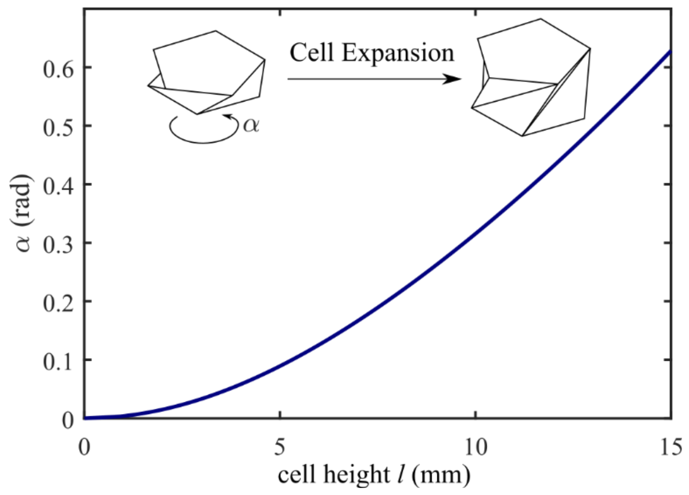


Figure 2.8 Origami cell expansion depicted as function of applied angle α and cell height *l*. Maximum cell's height is 15 mm and it corresponds to the input angle 0.6283 rad. Image is taken from [41]

2.4 Robot Locomotion and Iterative Feet Design

2.4.1 Robot Crawling Locomotion

Anisotropic friction feet are potentially the most critical design aspect for the OSCAR's crawling locomotion (Fig. 2.1). Ideally, these feet provide low friction in the forward direction and high friction in the backward direction to prevent backward slippage.

Considering ideal feet behavior, OSCAR's locomotion can be described as follows (Fig. 2.9). Starting from the fully contracted state, the towers expand due to the applied servo rotation. As a result, the robot front plate moves forward while the back plate stays fixed due to the high friction deployed at the robot feet, allowing the robot to expand. Then, during contraction, the front plate remains fixed due to the high friction side of the robot feet being deployed, while the back plate moves forward. Upon contraction, the robot returns to a fully contracted state, and the cycle repeats.

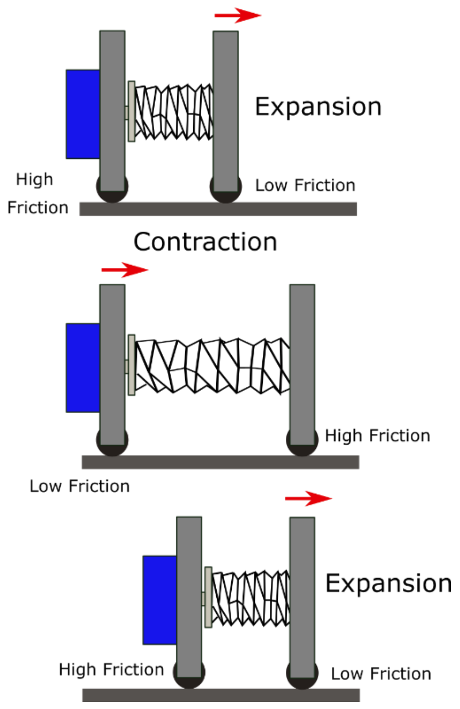


Figure 2.9 OSCAR' crawling locomotion schematic

2.4.2 Iterative Feet Design

In practice, the feet's anisotropic friction and their interaction with the environment are far from ideal. In fact, it is one of the main contributors to the OSCAR's *motion uncertainties*. As a result of non-ideal friction, the back plate slides backward during expansion, and the front plate slides backward during contraction, although they are assumed to be stationary in theory. Thus, the robot motion becomes *highly uncertain* and hard to control. To minimize uncertainties due to foot slippage, several design iterations have been investigated since introducing the initial wedge feet design in [41], [42].

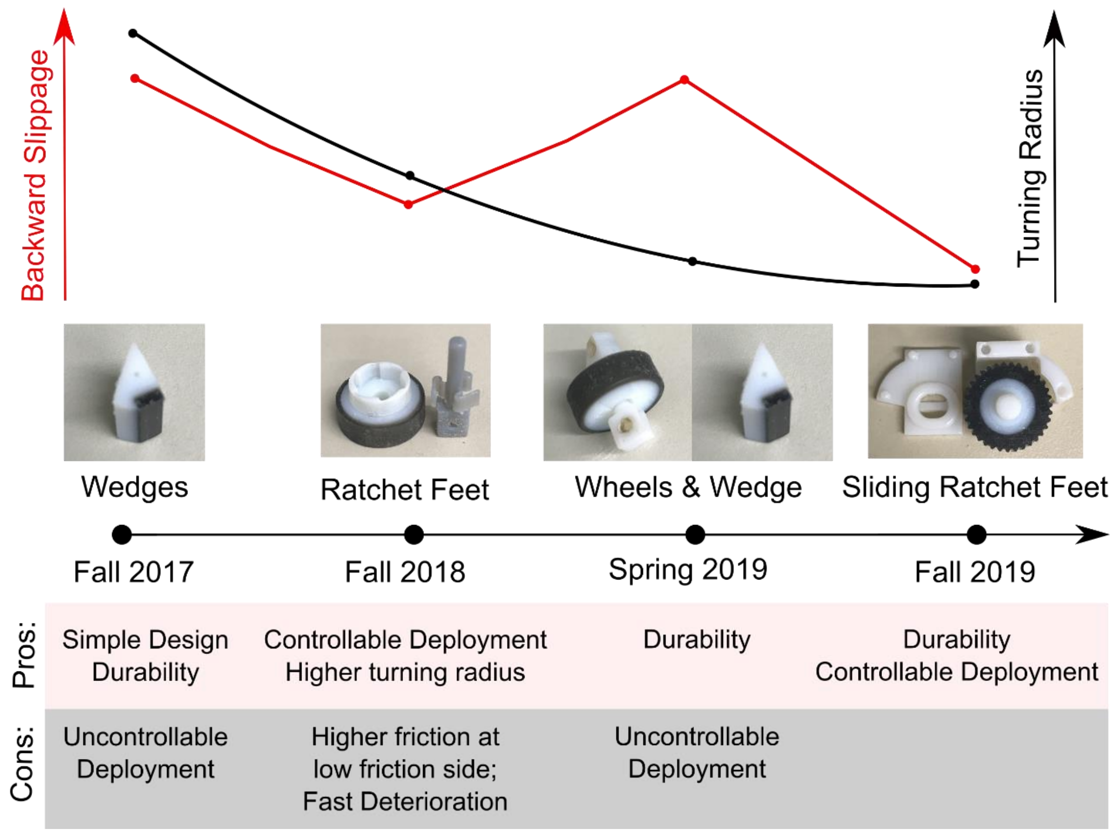
A timeline of feet design iterations (Fig. 2.10) includes the following designs:

D1. Anisotropic friction wedges (three per plate);

D2. Wheels with ratchet mechanism (two per plate);

D3. Combination of (*D1*) and wheels of (*D2*), but without ratchet mechanism (two wheels and one wedge per plate);

D4. Sliding ratchet feet (two per plate).



	Fall 2017	Fall 2018	Spring 2019	Fall 2019
Pros:	Simple Design Durability	Controllable Deployment Higher turning radius	Durability	Durability Controllable Deployment
Cons:	Uncontrollable Deployment	Higher friction at low friction side; Fast Deterioration	Uncontrollable Deployment	

Figure 2.10 Timeline of the foot design evolution with description of performance improvement

Besides significantly mitigating the backward slippage, the feet design iterations also focused on improving the robot's turning capabilities, as shown in Fig. 2.10. A lower turn radius corresponds to a higher curvature turn. In all designs, switching between low and high friction states is passive due to inertia (*D1*, *D2*, *D4*) or mechanical design (*D2*). All of the feet shown in Fig. 2.10 have been 3D printed on an Objet260 printer with VeroWhite and TangoBlack materials.

Figure 2.11 shows images of the different foot designs implemented on the OSCAR. As can be seen, the early designs still have the bellows attached; however, it is removed in a later design to enhance robot maneuverability. With the simple wedge foot design (*D1*), the robot had three feet per plate: one wedge at the center and two wedges at the sides (Fig. 2.11, left). When the plate moved forward, the wedge foot deploys a low friction side (VeroWhite). Conversely, when the plate moved backward, the wedge foot switches to the high friction side (TangoBlack).

Although these feet have high durability, their deployment is unreliable; i.e., in some cases, the high friction side remained undeployed during backward motion resulting in high backward slippage and robot low turning capability (i.e., large turning radius).

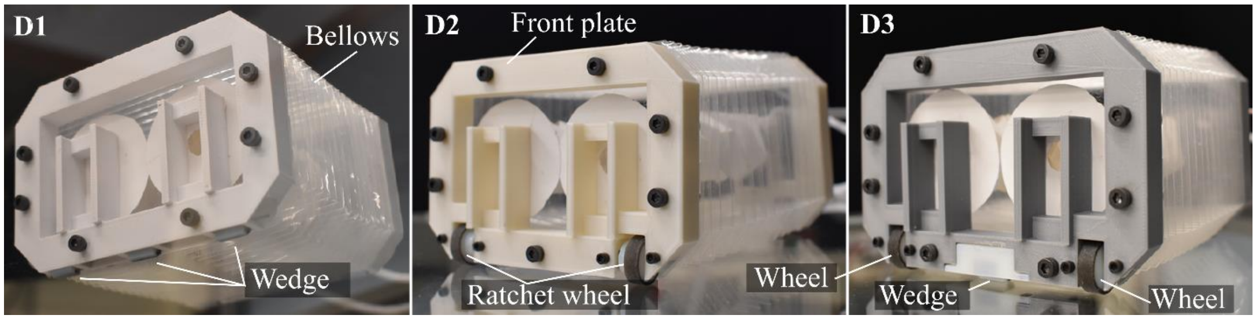


Figure 2.11 Foot designs *D1-D3* implemented on the robot. Shown here is an intermediate robot design with bellows

Therefore, to address the wedge feet's passive deployment problem, wheels with a ratchet mechanism (*D2*) were implemented (Fig. 2.11, center). Each foot has a wheel with six ratchets on its inner surface and a static axle with three pawls (Fig. 2.10). The wheel can freely rotate in the forward direction, while pawls stop it from moving backward. Although (*D2*) decreased backward slippage and improved turning, these feet have high friction during the desired low friction operation, as the pawls are in constant contact with the wheel. A further disadvantage of (*D2*) is fast pawl deterioration and wear since these are 3D printed out of relatively soft material. Therefore, a combination of two wheels and and a wedge (*D3*) was proposed. These feet combine the advantages of the two previous designs, as two wheels at the plate's sides allow better turning, while a single wedge at the center provides passive friction switching (Fig. 2.11, right). Nevertheless, this design resulted in unreliable friction switching similar to what was observed in *D1*. Thus, the final design of sliding ratchet feet (*D4*) has been implemented.

The sliding ratchet foot mechanism (*D4*) resembles a ratchet mechanism with a ratchet wheel freely sliding between two states: a locked and a free-rotation position (Fig. 2.12). Switching between these states happens passively due to inertia and ground friction during the plate's motion. Therefore, when the robot plate moves forward, the ratchet wheel slides back to a position where it can freely rotate, providing low friction for the forward motion (Fig. 2.12, a). Conversely, when the plate is forced to move backward by the origami towers' contraction, the ratchet wheel encounters two pawls and stops. In the locked position, the feet provide high friction preventing undesirable plate backward slippage (Fig. 2.12, b).

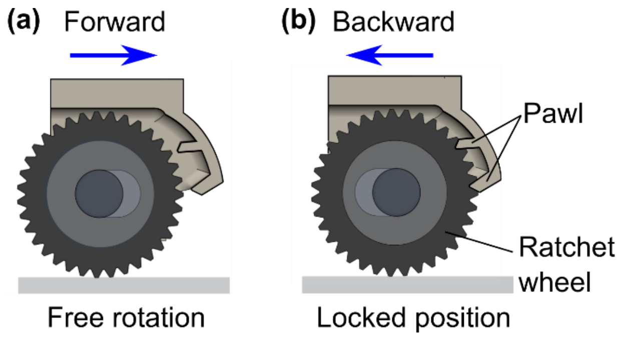


Figure 2.12 Sliding ratchet foot: (a) low friction situational deployment during plate forward motion; (b) high friction situational deployment during the plate backward motion

Due to its robust friction switching mechanism, the sliding ratchet feet have significantly decreased the backward slippage and improved OSCAR's turning compared to all the previous feet designs (see Fig. 2.10).

2.4.3 Stabilizers

Stabilizers prevent the plates from pivoting in the X-Z plane around the instantaneous ground connection point when in motion (Fig. 2.13). The stabilizers are freely rotating wheels offset by 11 mm from the plate horizontally and by 1.5 mm from the ground vertically to guarantee the grip between feet and ground. Two stabilizers per plate are used. As shown in Fig. 2.13, when the towers are actuated, they generate a pulling force that causes the plate to pivot about the foot ground connection point. Stabilizers limit plate pivoting in the X-Z plane and ensure that there is the robot-surface interaction. Previous OSCAR designs utilized the aforementioned bellows (Fig. 2.2) to stiffen the robot chassis and minimize the pivoting of the plates during contraction and extension of the towers. However, the stiffness penalty in turning was too high for the bellows to be functionally useful. Therefore, the stabilizers were the better design choice.

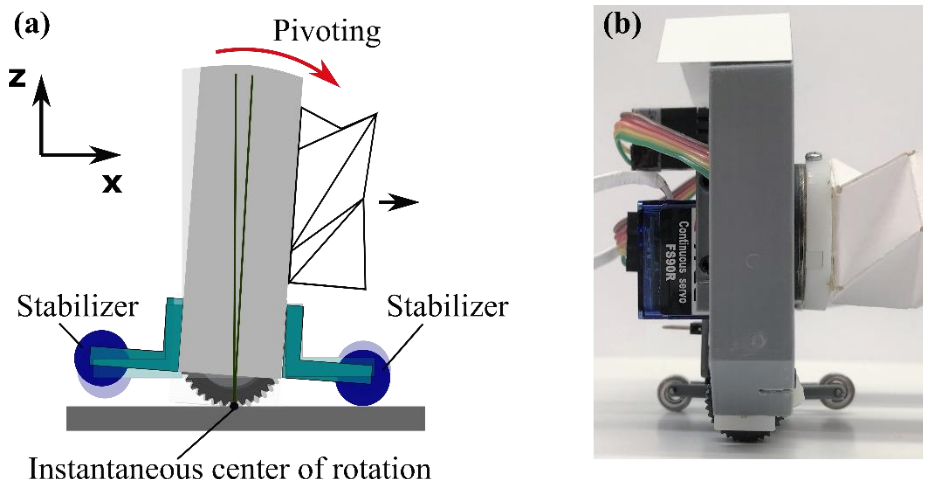


Figure 2.13 Stabilizers: (a) functionality; (b) implementation

2.5 Motivation for the Assembly Guide and Analysis of Low-Level Closed-Loop Control

The finalized OSCAR design used for autonomous control studies in this thesis and shown in Fig. 2.3, was built in Fall 2019. It had the low-level closed-loop servo position control to control tower rotation and, hence, robot expansion and contraction. The proportional-integral (PI) controllers with saturation based integral anti-windup have been used for this servo position control (see Chapter 3, (3.2)-(3.5)). In initial approaches, the controllers track step angular reference inputs provided to the robot.

With the finalized design, we conducted several initial path following experiments. The results showing OSCAR's trajectories and orientations are presented in Fig. 2.14. In these experiments, OSCAR attempted to follow a straight reference path, starting with an initial offset in the y-direction relative to the path. The robot was expected to converge to the path, i.e., its position error and orientation error should decrease with time. Here, we are focused on robot performance, so details on the applied path following controllers are omitted. The robot position is tracked in the experimental setup, explained in Chapter 3.

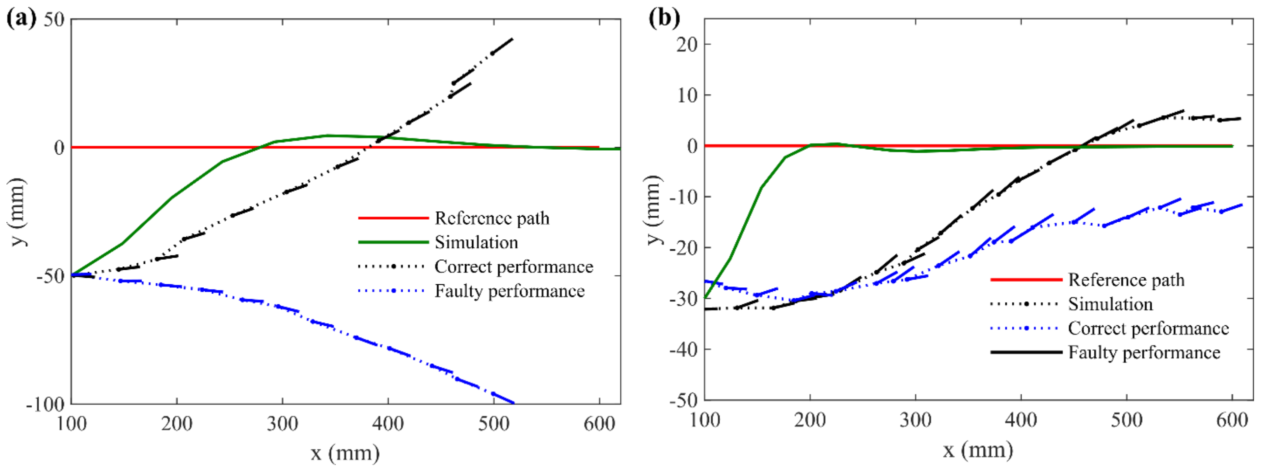


Figure 2.14 Straight line (red) path following with the robot starting at an initial offset in y-direction: (a) in open-loop; (b) with feedback path following controller. Dotted black and blue lines are robot trajectories in the correct and faulty cases; solid black and blue lines are robot orientations

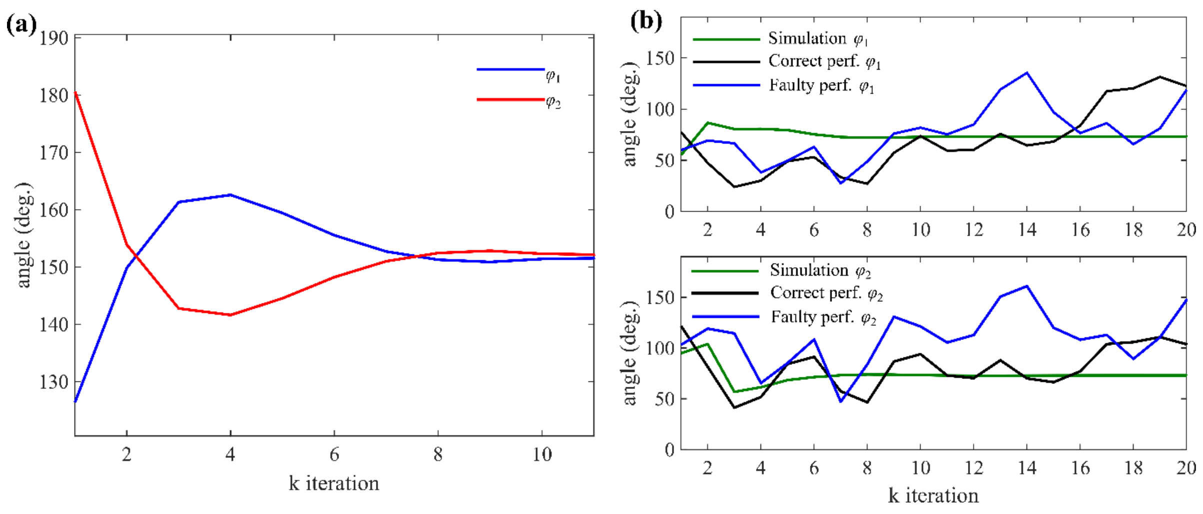


Figure 2.15 Angular control inputs for the path following in open-loop (a) and with proportional feedback (b). Angular inputs in (a) are used in both faulty and correct performances in Fig. 2.8, a. Angular inputs for Fig. 2.8, b are shown in (b); inputs are similar for both faulty and correct performances

Although design changes performed in previous sections significantly improved OSCAR performance, significant *motion uncertainties* were still present in experimental results. Figure 2.14(a) shows the path following with an open loop set of servo commands. These servo commands were computed based on the system model and resulted in desired angular inputs provided in Fig. 2.15(a). Figure 2.14(b) shows the path following with a closed loop proportional feedback controller that is based on path displacement error and orientation error. A similar one will be detailed and designed in Chapter 5. The same or similar angular inputs are provided in all experiments, as shown in Fig. 2.15, (a) and (b), respectively. In both cases, the robot does not perform like the simulation predicts. The ability to follow the path is poor for even the best performing robot response. In the case of a poorly performing, or faulty, robot response, the orientation or position can even diverge from the predicted response and have an opposite sign.

The reason for this is that there are significant uncertainties in the robot behavior. Due to these uncertainties, the robot cannot follow the reference path reliably, even with the feedback controller. After much investigation, it was found that there was variation in the OSCAR dependent on how it was assembled. In short, robot design is important; equally important is robot manufacturing. Manufacturing and assembly matter a great deal. This motivated the development of the OSCAR's assembly guide, presented in the following section, and the low-level servo control analysis covered in Chapter 3, which together mitigated these remaining uncertainties.

2.6 OSCAR's Assembly Guide

2.6.1 Assembly Guide

As illustrated above, a major contributor to OSCAR's motion uncertainties is the assembly process. It directly affects OSCAR's feet-ground interaction. Due to the origami towers' compliance, the front and back plate alignment to the ground is very difficult to guarantee during the assembly process. As a result, plates can tilt about the X-axis, which was observed and measured experimentally on OSCAR in early assembly efforts. This tilt causes uneven friction between the robot feet and the ground with some feet being slightly lifted off the ground. Clearly, this misalignment between the robots' warped plane and the X-Y plane of the ground environment leads to non-ideal robot-ground interaction.

The existing assembly challenges can be summarized in the following list:

- *Front and back plate misalignment in the Y and Z directions.* Due to the individual towers' compliance, the front and back plates' alignments to each other (in the Y direction) and the ground (in the Z-direction) are impossible to guarantee without external means during the assembly process.
- *Front and back plate tilting caused by servos initialization.* Once the origami towers are fixed in the front plate, any pre-stress in towers results in the front and back plates' tilting to the ground. This pre-stress can be caused by sudden servo motion during the robot initialization. The sudden servo motion can be caused when the servos are initially powered on. This phenomenon is due to the low-level servo controllers and is common among small-scale continuous rotation servos.
- *The robot's fully contracted length should be consistent* for each robot assembly to guarantee the performance repeatability.

To address these problems, a custom-designed and custom-built assembly guide has been developed and constructed. The 3D printed assembly guide (Fig. 2.16) aligns both the front and back plates in the X-Y and X-Z planes with a distance between them equal to the origami towers' fully contracted length in the X direction. This distance is set to $dl = 41mm$. The guide prevents the front and back plates from tilting around the X axis, while the assembly process outlined below prevents plates' tilting due to servos during robot initialization. As a result, the assembled robot has much improved evenness in the surface friction among all the feet and the ground.

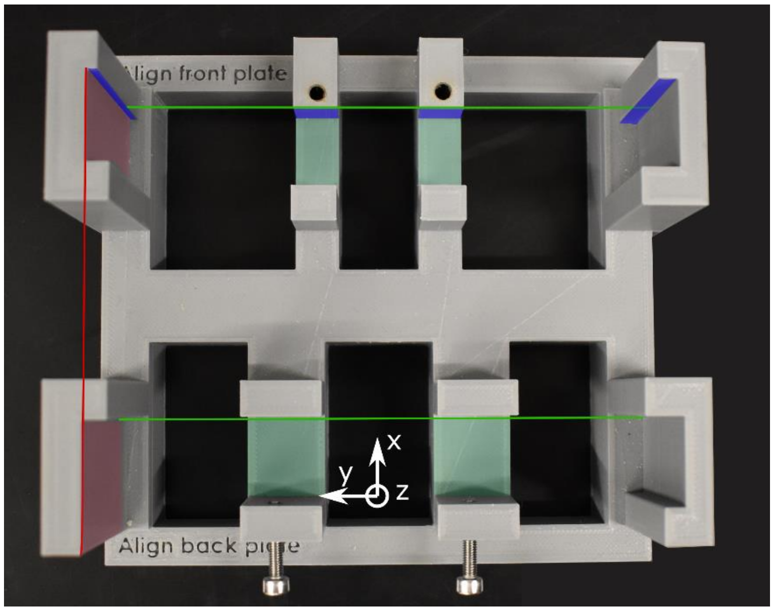


Figure 2.16 Assembly guide. The front and back plate's alignment in the X-Z plane is achieved by setting the plates' left faces against the red shaded areas, thus aligning them along the red line. Alignment in the X-Y plane is provided by contact of plates' bottom surfaces with the green shaded areas. The front and back plates are aligned parallel to each other in the Y-Z plane along the green lines. The front plate is aligned in the Y-Z plane by contact with the blue shaded area; the same holds for the back plate. Once aligned, the front plate is fixed by the fixtures in Fig. 2.17, and set screws fix the back plate.

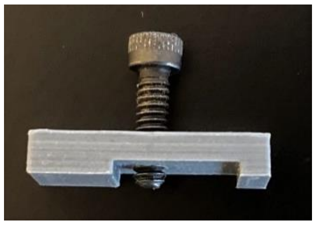


Figure 2.17 Side view of fixture used for holding the front plate in the assembly guide

2.6.2 OSCAR's Assembly Process

The following is presented to capture for the reader details that, while conceptually quite simple, have a large effect on the overall robot performance. The assembly process with the designed guide includes the following steps:

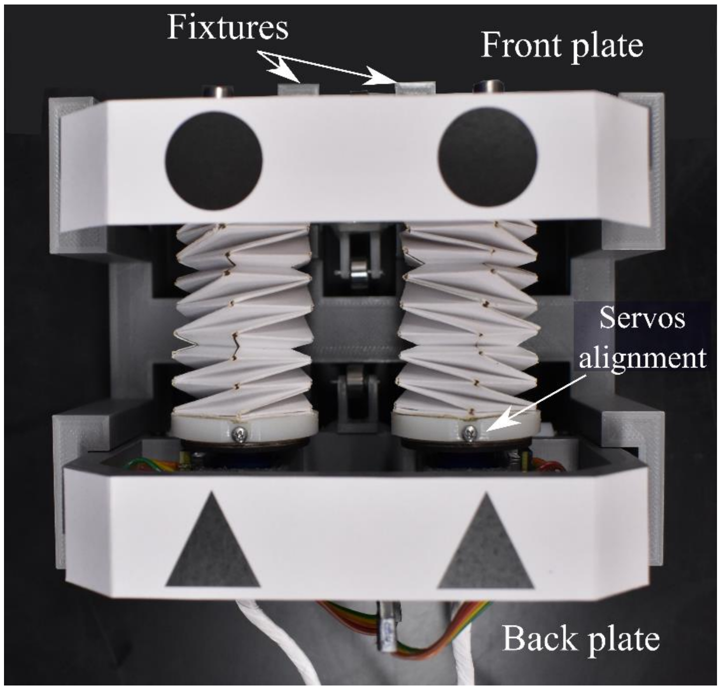


Figure 2.18 OSCAR in the assembly guide. The servo horns are centered for easy monitoring of servo positions during robot initialization

1. Front and back plates need to be pre-assembled. All the robot components need to be installed on the front and back plates. This includes the origami towers being installed on the back plate but left disconnected from the front plate.
2. The front and back plates need to be lined up to the left side of the assembly guide and fixed in the guide, as shown in Fig. 2.18. After being fixed, plates are aligned in the X-Y and X-Z planes. The detailed description can be found in Fig. 2.16. The origami towers' free ends need to be set in the untightened acrylic plates at the front plate. With applied servo rotation, towers can freely rotate without expanding.
3. The robot should be initialized by turning on the power source and microcontroller. During initialization, servo horns should be centered, as shown in Fig. 2.18. A single cycle of expansion and contraction inputs must be applied to the origami towers. The last step is not required but is believed to remove any accumulated stress in the origami towers and help self-center them.

4. The LabVIEW virtual instrument (VI) that localizes and controls OSCAR needs to be started, see ‘Main VI.vi’ in Appendix A. The robot re-initializes when the VI starts and this leads to the previously discussed sudden servo motions. As detailed in the experimental setup in Chapter 3, there is a microcontroller interface between the LabVIEW VI and the low-level servos. Manually press and hold the microcontroller ‘reset’ button and move servos to their centered positions.
5. Finally, the origami towers’ ends need to be fixed into the acrylic plates in the robot front plate, finishing the assembly. We can now release the fixtures holding the front and back plates in place.

To prevent tilting or deviations that can occur with use, and to maximize experimental repeatability, OSCAR is reassembled before each experiment. This means that steps from 2 to 5 are repeated before each experimental session. The reassembly takes on average 2 minutes and is not time-consuming. Different types of micro-size motors could be used in the future to prevent the reassembly process need.

2.7 Chapter Summary

The control-oriented design has been a crucial part of this research, as it enabled the OSCAR’s autonomous navigation. The iterative design process helped to eliminate the motion uncertainties present in OSCAR. Hence, OSCAR could be controlled by high-level control. The OSCAR’s motion uncertainties are caused by (i) its feet and the ground interaction and (ii) OSCAR’s low-level control. This chapter presents the feet design and assembly process that addresses the motion uncertainties due to the feet and ground interaction. The low-level control involves the experimental setup, and hence, it is presented in Chapter 3. The steps presented in this chapter are vital for understanding the robot details and practical challenges. Additionally, they should serve as a guide for future designs for this class of origami-enabled mobile robots.

Chapter 3
Experimental Testbed

3.1 Testbed Components

Experiments in this work are conducted on a testbed designed specifically for OSCAR. The testbed is used to perform robot localization and to implement the path following control algorithms. It has an extruded aluminum frame that holds a camera mounted on its top, as shown in the schematics in Fig. 3.1. The camera has a ‘god’s eye view’ of the robot’s operational workspace. The testbed workspace surface is prepared to be spatially uniform with sufficient traction. The picture of the actual testbed is shown in Fig. 3.3.

The camera is used for robot localization, which is performed via image processing algorithm developed specifically for OSCAR and covered in detail in Section 3.2. The camera captures the image of the robot in the operational workspace and sends it to the PC during the experiments, see flowchart in Fig. 3.2. The PC operating the testbed runs the image processing software that localizes the robot. The designed software identifies the robot position and orientation through markers placed on the robot (Fig. 3.4). Afterwards, the control algorithm calculates the robot inputs. The testbed operation is described in Section 3.3. All software is implemented in the LabVIEW NI Virtual Instrument (VI). The VI details are provided in Appendix A.

The camera used is an LI-OV5640-USB-72 camera from Leopard Imaging, which has a USB 2.0 interface. It has a resolution of 1280×960 pixels at a selected speed of 30 frames per second. The camera is raised from the robot workspace by $h_z = 623.6\,mm$, measured between the front of the camera lens and workspace surface (Fig. 3.1). At this distance, it provides a resolution of 1.13 mm and 1 mm in the x and y-direction, respectively, for measurements done in the plane of the robot markers. The markers, shown in Fig. 3.4, have a size of $16.7 \times 16.7\, mm$; thus, the camera resolution provides 6% error for marker identification. It is sufficient for the current application.

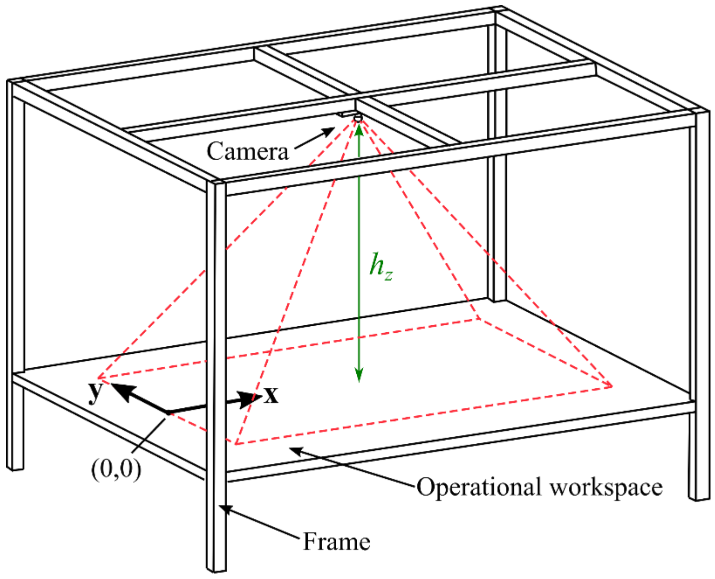


Figure 3.1 Schematic drawing of the experimental testbed (the PC and robot power supply are not shown)

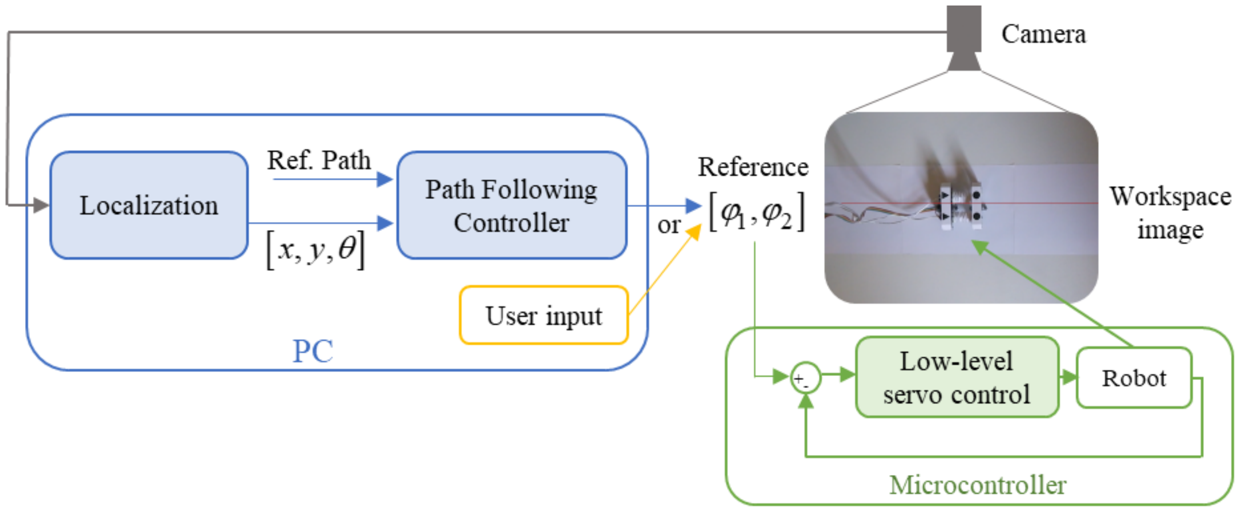


Figure 3.2 Flow chart of the testbed operation

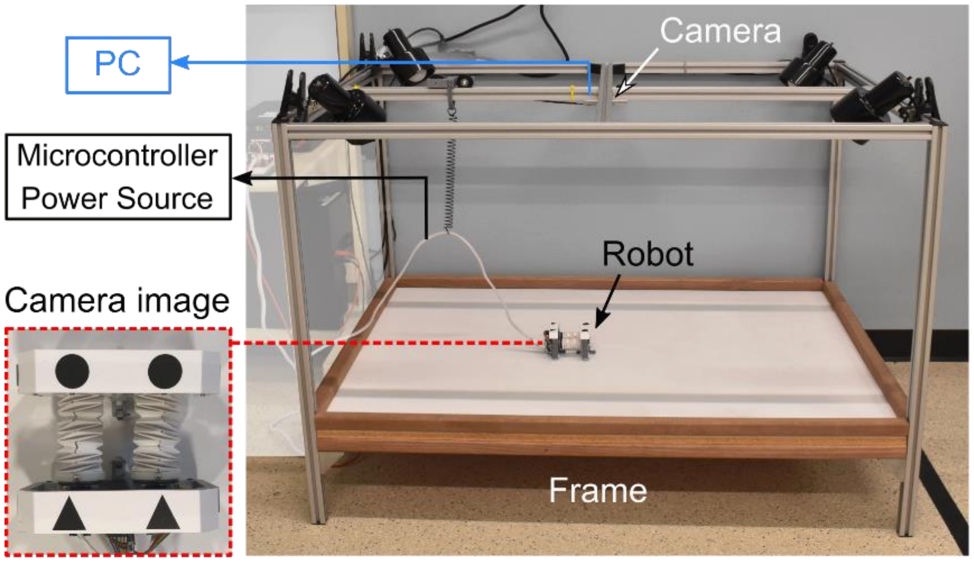


Figure 3.3 Experimental testbed

The camera has been calibrated with respect to the operational workspace using a standard calibration grid provided by LabVIEW NI. During calibration, the calibration grid covered all the surface of the operational workspace. The camera lens has some distortion, which has also been compensated by the calibration. The resulting measurement error is 0.4 mm in both x and y-directions, and the camera coordinate frame is as shown in Fig. 3.1.

The operational workspace has dimensions $1219.2 \times 914.4\,mm$ (or $4' \times 3'$). Considering the OSCAR dimensions $(95 \times 106\ mm)$ at the fully contracted state, the testbed allows for 12 robot body lengths. However, the actual operational workspace is limited to the calibrated image size captured by the camera. Furthermore, the measurements are done in the plane of the robot markers. Therefore, the resulting actual workspace has a size of $700 \times 250\,mm$ in the markers' plane, as shown in Fig. 3.6. Therefore, the actual workspace covers only 7 robot body lengths. This is a limitation for the current OSCAR experiments, as will be seen in Chapter 5. We are able to compensate for this limitation by concatenating multiple experiments to create a larger workspace. Future efforts requiring a larger workspace would necessitate new hardware, including a wider camera field of view with retained or improved resolution.

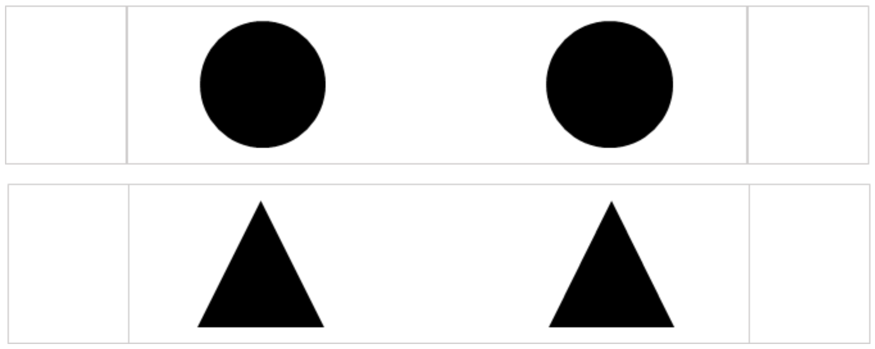


Figure 3.4 OSCAR markers (in 1:1 scale): (top) for front plate; (bottom) for back plate

Additionally, the testbed includes an offboard power supply for OSCAR, which provides a 5.5 V constant voltage to the servos and current ranging from 0 to 1.2 A. OSCAR has as an offboard Arduino microcontroller (Fig. 3.3). It implements a low-level closed-loop servo position control, as shown in Fig. 3.2 and covered in detail in Section 3.4. The tethered configuration is chosen over the untethered one for ease when conducting experiments, as our primary goal is to demonstrate the feasibility of autonomous navigation. These offboard elements could be miniaturized and placed on board. Untethered locomotion was previously demonstrated in [40] and [46].

Table 2.1 OSCAR Electronic Components

Component	Details	Quantity
Motor	• Feetech FS90R (Polulu) • Continuous rotation servos	2
Encoder	• Sensor RLC2HD (RLS) • Magnetic ring MR026C016B036B02 (RLS)	2 2
Encoder counter	• Dual LS7366R Encoder Counter (SuperDroid Robots) • SPI communication	1
Microcontroller *(off board)*	• RedBoard for Arduino (SparkFun) • USB connection to PC	1
Power source *(off board)*	• B&K Precision 1900 • 5.5V constant voltage input	1

As stated in Chapter 2, the angular inputs to the origami towers are measured by the incremental magnetic encoders. In order to read their data, the encoders are interfaced with the robot microcontroller through a counter. The counter is placed onboard of OSCAR. All OSCAR electronic components are listed in Table 2.1.

3.2 Image Processing Software

In order to perform localization, OSCAR has four planar black-and-white markers (Fig. 3.4) placed on top of the robot plates, as shown Fig. 2.3 and Fig. 3.3. Two markers per plate are used to determine both position and orientation of the plate. The markers have unique geometric shapes: circles for the front plate and triangles for the back plate. This allows the front and back plates' positions to be distinguished from each other in the developed algorithm.

Alternative solutions for the robot localization could be the commercially available motion capture systems, such as the ones provided by OptiTrack and VICON. These systems use infrared (IR) cameras and track the subjects by following reflective markers. As markers reflect the emitted IR light, these systems can localize the robot in 3D space. However, the listed motion capture systems are expensive as they require multiple cameras and need a larger operational space than what is currently available for OSCAR at the writing of this thesis. Another alternative low-cost localization method is done by using fiducial markers, e.g., AprilTag [47]. AprilTag uses square black-and-white QR-code like markers tracked by the camera with the provided software. However, these markers require a larger area than the area available at the top of the robot plate and may result in erroneous measurements, as stated in [48].

The developed LabVIEW-based image processing algorithm utilizes a *geometric matching* algorithm from the NI Vision toolbox to identify the robot markers. This algorithm allows marker identification regardless of its rotation, displacement, and changing light conditions. Details on the developed algorithm are provided in Appendix A. The marker templates, shown in Fig. 3.5, are created beforehand to apply geometric matching. The template is a sample image of the marker that contains information about its geometric shape, size, and centroid position.

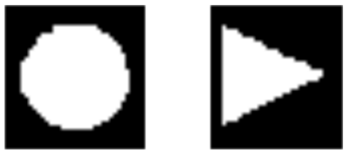


Figure 3.5 Marker templates: (left) for front plate; (right) for back plate

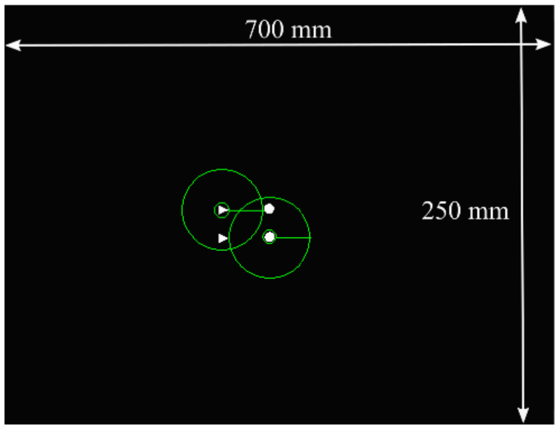


Figure 3.6 Processed image with identified markers

The localization is performed in a three-step process. First, the RGB image taken by a camera is processed to leave only the marker-sized areas. In this process, the RGB image is converted to greyscale and then to binary format. After that, the image is filtered to leave only marker-sized areas, called particles. Then, the geometric matching matches these particles with marker templates by comparing their shapes. To speed up the localization process, we identify only one of the markers per plate and locate the second marker in a circular area (green line) around it in the processed image (Fig. 3.6). Upon matching, marker positions are found in the global camera coordinate frame, shown in Fig. 3.1. Finally, with two known marker positions per plate, the corresponding plate centroid position and orientation are calculated.

The resulting position error is 0.7 mm in both x and y-directions, and the orientation error is 1 degree. The errors have been calculated based on camera calibration error being scaled by the factor of 0.865. This scaling factor was empirically found to convert the measurement from the operational workspace plane to the markers' plane.

3.3 Testbed Operation

As described in Chapter 2, OSCAR locomotes by consecutively expanding and contracting its body with provided control inputs. These inputs are the servos' reference angles $[\varphi_1 \quad \varphi_2]^T$. We can define a locomotion cycle as one consecutive expansion and contraction of the robot body. After each locomotion cycle, OSCAR returns to its fully contracted state. Therefore, as OSCAR moves, two distinctive states could be identified: a fully expanded state and a fully contracted state (Fig. 3.7). For the control purposes, the OSCAR's measured state is defined as a front plate's centroid position and orientation, i.e., $p = [x \quad y \quad \theta]^T$, measured in the global coordinate frame shown in Fig. 3.1.

As shown in Fig. 3.2, when OSCAR returns to its fully contracted state, the camera takes an image of the robot's operational workspace. Then, the image processing software localizes the robot. The defined state $p_k = [x \quad y \quad \theta]^T$ is the robot state for the the current locomotion cycle, where *k* indicates the index of the current locomotion cycle. Based on the identified robot position and orientation, the control inputs for the current locomotion cycle $[\varphi_1 \quad \varphi_2]_k^T$ are defined by either the path-following controller or by the user input. These control inputs are then sent to the robot microcontroller via a tethered serial connection. The robot microcontroller implements a low-level servo position control, described below, and the robot expands. To measure forward displacement and characterize the backward slippage, the localization is repeated at the robot's fully expanded state, see Fig. 3.7 (middle). In this case, the robot receives the zero angular reference servo inputs for contraction, i.e., $[\varphi_1 \quad \varphi_2]^T = [0 \quad 0]$. The robot returns to the fully contracted state, see Fig. 3.7 (bottom). The above process repeats for the next expansion and contraction cycle.

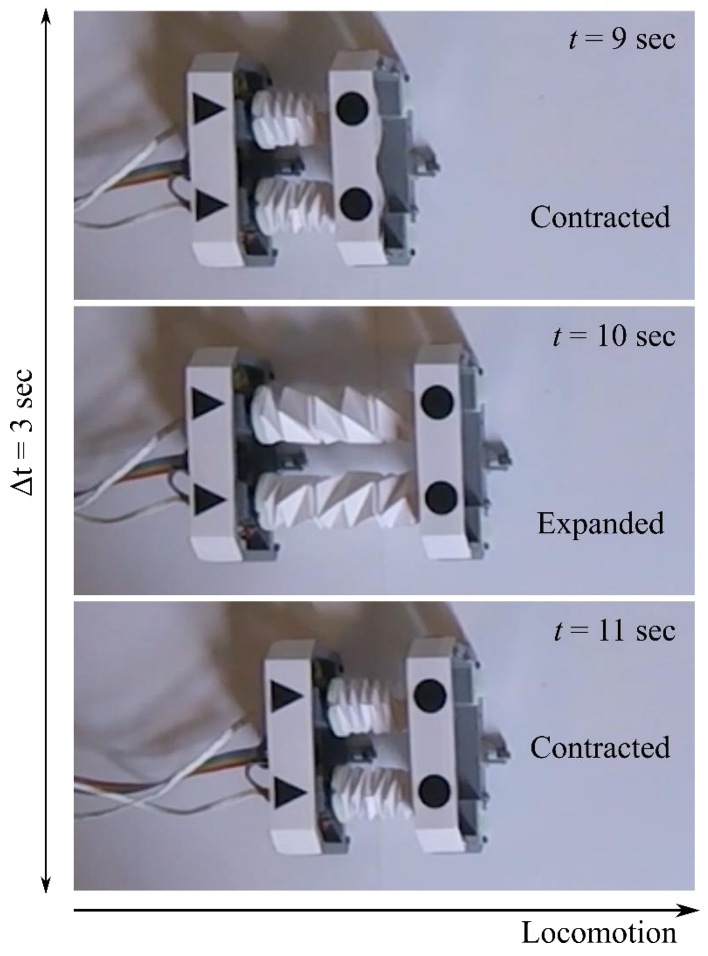


Figure 3.7 Sequence of video frames depicting OSCAR locomotion

3.4 OSCAR Low-Level Servo Position Control

Low-level closed-loop servo position control is critical to perform OSCAR's path following and path planning. The low-level closed loop control reduces servo position uncertainty and enables repeatable expansion and contraction cycles. The low-level control provides a controlled expansion and contraction of the origami towers by providing closed-loop control of the servos' rotational position. Since it is closed loop, any disturbances or motor nonlinearities can be compensated. The resulting predictability of the servo motion is a key enabler of the OSCAR's path following ability.

This section first describes the PI control architecture and then highlights the importance of the constant velocity during expansion and contraction.

3.4.1 Proportional-Integral (PI) Servo Position Controller

The continuous rotation servo motor operates with a PWM input signal, which regulates the servo angular velocity. The servo motor can be approximated as a first-order transfer function [49] with the PWM signal duty cycle as an input and the angular velocity as an output

$$G(s) = \frac{\Omega(s)}{\Delta U(s)} = \frac{K}{\tau s + 1} \tag{3.1}$$

In this transfer function, the coefficient $K = 17.879$ and time constant $\tau = 0.0278$ sec are identified from a set of servo step responses based on the encoder data. Figure 3.8 compares step responses of identified model and data for an input $\Delta u = (u - 90)/90,$ where $u = 120$ is in the Arduino command. It should be noted, there is a small-time delay that is ignored in the identified model. The data samples are collected every 5 ms.

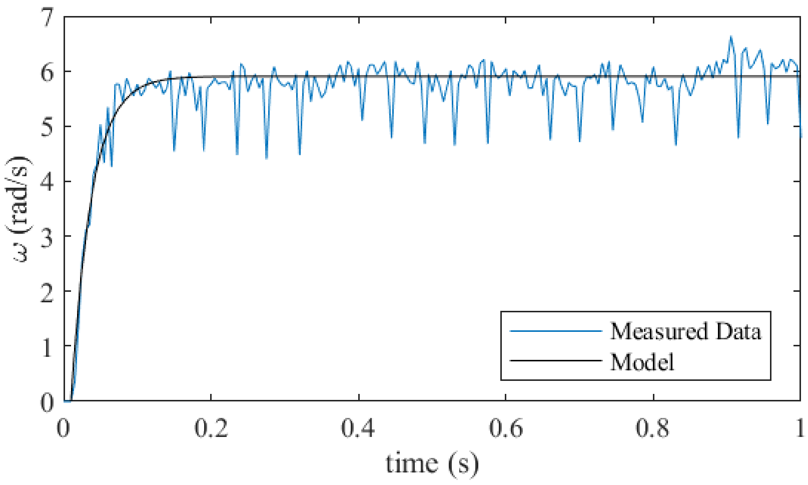


Figure 3.8 Step response of identified transfer function

The servos' angular positions are controlled by proportional-integral (PI) controllers. For the origami towers' safety, the servo position is regulated to within ± 2 degrees of the angular reference input. The PI controller has the following standard form in continuous time,

$$\Delta u(t) = k_p e + k_I \int_0^t e(\tau) d\tau \tag{3.2}$$

where $e = \varphi_{ref} - \varphi$ is the error between reference and measured angles, k_P and k_I are proportional and integral gains, respectively. The control input $\Delta u(t)$ is bounded by (3.3) and subjected to limitation, or saturation, on the integrator term to prevent windup (3.4). Future work could develop a formal anti-windup algorithm but for the purpose of this investigation the saturation approach was sufficient.

$$\Delta \underline{u} \le \Delta u \le \Delta \overline{u} \tag{3.3}$$

$$\left| k_I \int_0^t e(\tau)\, d\tau \right| \le \alpha \tag{3.4}$$

The controller gains were initially estimated by MATLAB’s built-in ‘pidtool’ designer using the plant transfer function (3.2). After implementation, the gains were manually re-tuned to improve performance. The tuning goal was to remove the servos’ steady-state chattering and prevent accidental towers’ over-expansion and over-contraction due to closed-loop overshoot. This overshoot was observed in early OSCAR tests and led to premature fatigue, wear, and failure of origami towers. The final controller gains in (3.2) in discrete time are $k_p = 0.6$ and $k_I = 0.4$. The values of limits in (3.3) and (3.4) are $\Delta \underline{u} = -0.2$, $\Delta \overline{u} = 0.2$ and $\alpha = 0.1$.

The servos have a deadband near zero angular speed inputs that affect the low-level controller performance, as described in subsection 3.4.2. The deadbands have been manually aligned to be in a range $[-\delta, \delta]$, and a deadband compensation scheme *d* (3.6) has been added to enhance the controller performance. The final form of control input is

$$u = 90(1 + s\Delta u + d) \tag{3.5}$$

where u is a PWM signal, $s = \pm 1$ indicates tower chirality and

$$d = \begin{cases} \delta \ if \ \Delta u > 0 \\ -\delta \ if \ \Delta u < 0 \end{cases}. \tag{3.6}$$

The multiplier 90 is added in (3.5), as the PWM input signal is given in the Arduino command, where $u = 90$ corresponds to zero angular velocity.

3.4.2 Motion Symmetry by the Low-Level Controllers

This subsection analyzes the low-level controller performance by studying the OSCAR displacement data for the range of available static angular inputs chosen below. The data has been collected in the experimental setup and is shown in Fig. 3.10.

The range of available OSCAR angular inputs is defined by their ratio $\varphi_2/\varphi_1 = [1/\bar{r}, \bar{r}]$, where $\bar{r}$ is the maximum angular ratio. Here, the ratios $\varphi_2/\varphi_1 = 1/\bar{r}$ and $\varphi_2/\varphi_1 = \bar{r}$ correspond to the maximum left and right turns, respectively. The ratio $\varphi_2/\varphi_1 = 1$ is a straight motion. The theoretically possible maximum angular input to the tower is $\varphi_{\max} = 216^\circ$, and the maximum angular input ratio is $\bar{r} = 1.8$ based on robot construction. However, the implemented upper limits are set to $\bar{r} = 1.6$ and $\varphi_{\max} = 180^\circ$ in the tradeoff between performance and the hardware safety. Tower expansion to its physical limit may result in accumulated cyclic wear so a safety margin is built in.

The data presented in Fig. 3.10 corresponds to angular ratios approximately $r = (1,\ 1/2,\ 1/4)\bar{r}$. OSCAR starts at the origin $p_0 = [x \quad y \quad \theta]^T = [0 \quad 0 \quad 0]^T$, and it performs $k = 12$ locomotion cycles with given reference angles in all experiments. Markers show OSCAR positions after each expansion and contraction, and lines denote OSCAR trajectories. When the two origami towers expand or contract non-uniformly, they result in significant motion uncertainties, as demonstrated in Fig. 3.10(a). To achieve uniformity, two main changes have been introduced to the low-level controller: (i) deadband alignment and compensation; (ii) ramp reference angular inputs.

As OSCAR has the opposite chirality origami towers, the servos must rotate in opposite directions to expand or contract the robot body. If the existing servos deadbands are misaligned, the two towers result in expanding and contracting non-uniformly. The solid lines in Fig. 3.10(a) show the resulting robot motion. Without deadband corrections, the robot trajectories are significantly biased to the right of the horizontal axis for the same left and right turns inputs. The displacement becomes more symmetric with deadbands being manually aligned and compensated by (3.5)-(3.6) (dotted lines in Fig. 3.10(a)).

Further, a more constant and uniform speed in both towers during expansion and contraction is achieved by providing a ramp reference input instead of a step in the low-level PI controllers. The ramp reference input is a ramp signal until it reaches the desired reference value, as can be seen in Fig. 3.9(b). The ramp slope is 5.9 rad/sec. This experimentally determined value for the tower expansion/contraction speed provides the best OSCAR performance. However, it a subject to change for different robot designs.

Figure 3.9 compares responses of the low-level PI controllers (3.5) with a step and ramp reference angular inputs. A single locomotion cycle is shown. For comparison purposes, the desired reference angles are equal during expansion $\varphi_1 = \varphi_2 = 150\text{ deg}$. A first-order filter is applied to reference angles.

With a step reference angle in Fig. 3.9(a), the robot PWM becomes saturated at its maximum value. Therefore, as the towers expand in opposite directions, even small misalignments of the servo deadbands cause differences in servo speeds at saturation. As can be seen in Fig. 3.9(a), the two servo speeds are different during contraction. Thus, it results in a robot displacement bias, as demonstrated in experimental data (dotted lines) in Fig. 3.10(a). In contrast, with the ramp reference input, the controller tracks both desired position and slope, as shown in Fig. 3.9(b). It results in uniform tower expansion and contraction, which leads to the symmetry in the robot displacement, as demonstrated in experimental data in Fig. 3.10(b).

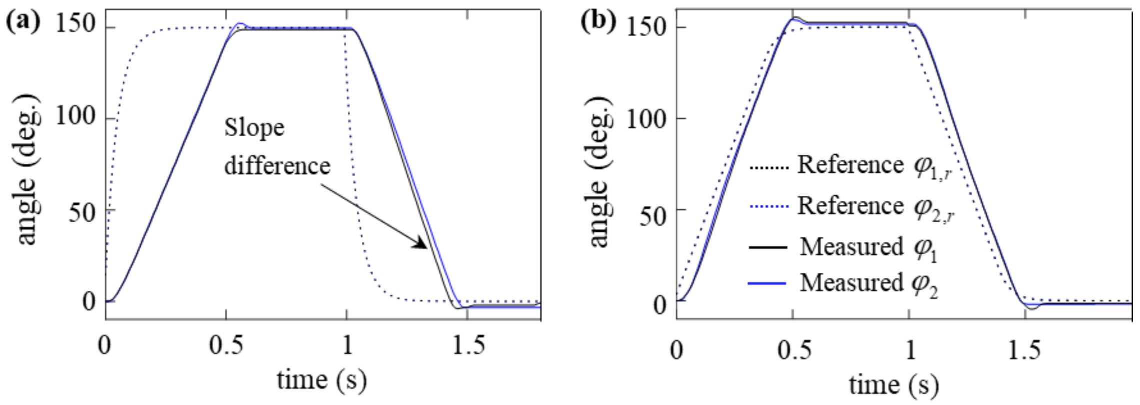


Figure 3.9 Reference angle tracking with low-level controller: (a) with step reference input; (b) with ramp reference input

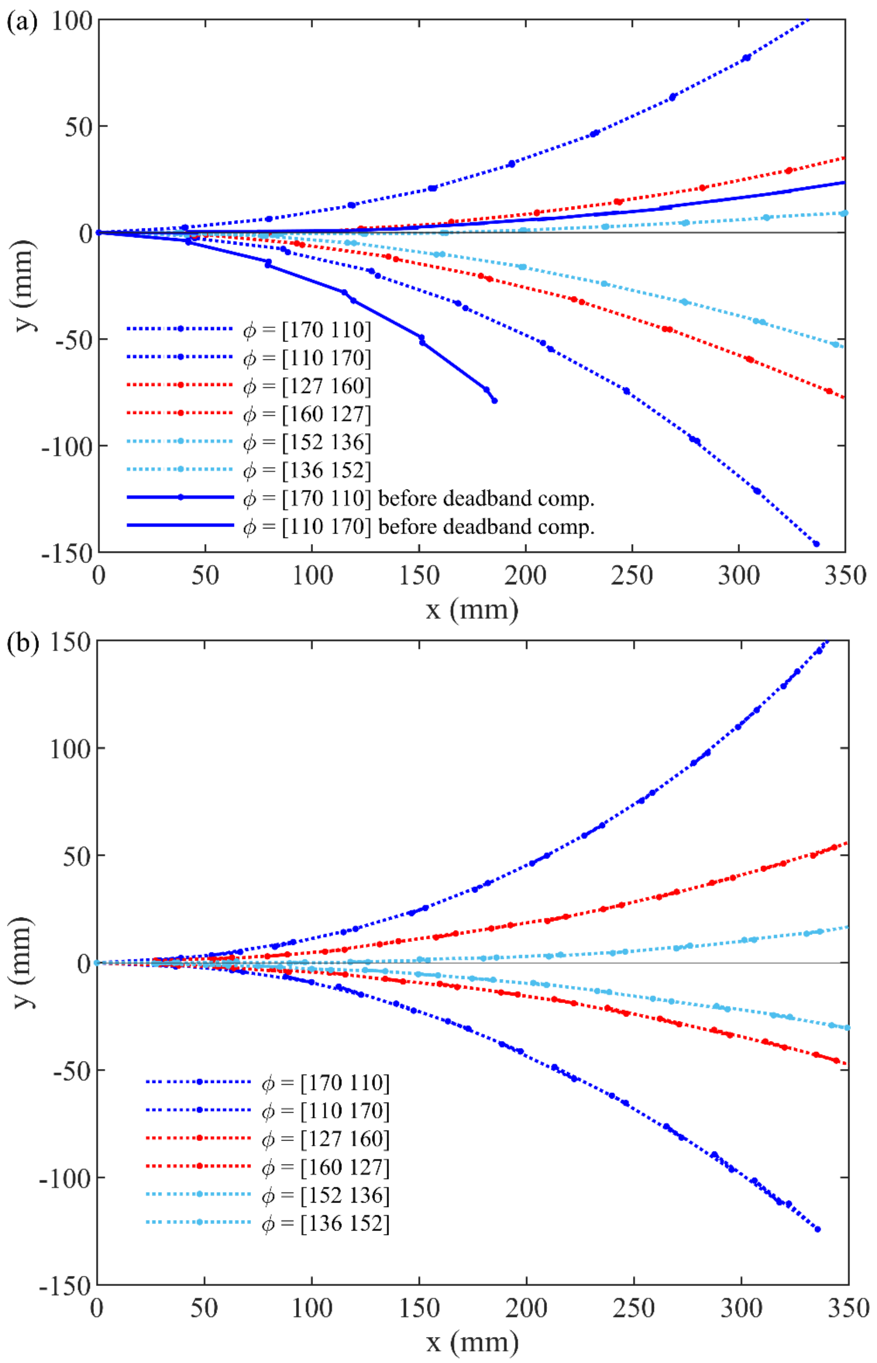


Figure 3.10 OSCAR displacement data collected for static angular inputs: (a) with step reference input in the low-level controller; (b) with ramp angular input in the low-level controller. The angular input values are shown in the legend, where each pair denotes the left and right servo angles, respectively

Chapter 4
OSCAR’s Kinematic Model

Unlike rigid-bodied robots, soft robots lack the readily available standardized models that can be used for high-level control development [2]. As mentioned in Chapter 1, the soft robotics include a wide variety of individual designs. Therefore, it is a challenge to create a standard model, and, as a result, each robot type generates its own model structure, as will be done here. This work develops and utilizes the OSCAR’s kinematic model that was developed and presented in [41], [42], [50]. Since OSCAR’s motion is sufficiently slow, any motion dynamics are not considered. The kinematic model is used in the path following control design and path planning covered in Chapters 5 and 6.

This Chapter is organized into two sections. The first section derives the kinematic model under idealized assumptions. Then, the second section validates the kinematic model against experimental data. Based on this validation, a correction is introduced to the kinematic model to represent the experimental data. As will be shown, this correction has a significant effect on model validity and also highlights one of the key challenges of working with soft origami robots: model uncertainty.

4.1 Kinematic Model

4.1.1 Kinematic Model Overview

Despite OSCAR’s soft nature, its motion can be approximated using rigid body motion. OSCAR locomotes by consecutively expanding and contracting its body. As defined in Chapter 3, the locomotion cycle is a single expansion and contraction of the OSCAR’s body. After each locomotion cycle, OSCAR returns to the fully contracted state. Therefore, its motion can be described as a rigid body translating discretely between locomotion cycles, where the rigid body is OSCAR at the fully contracted state.

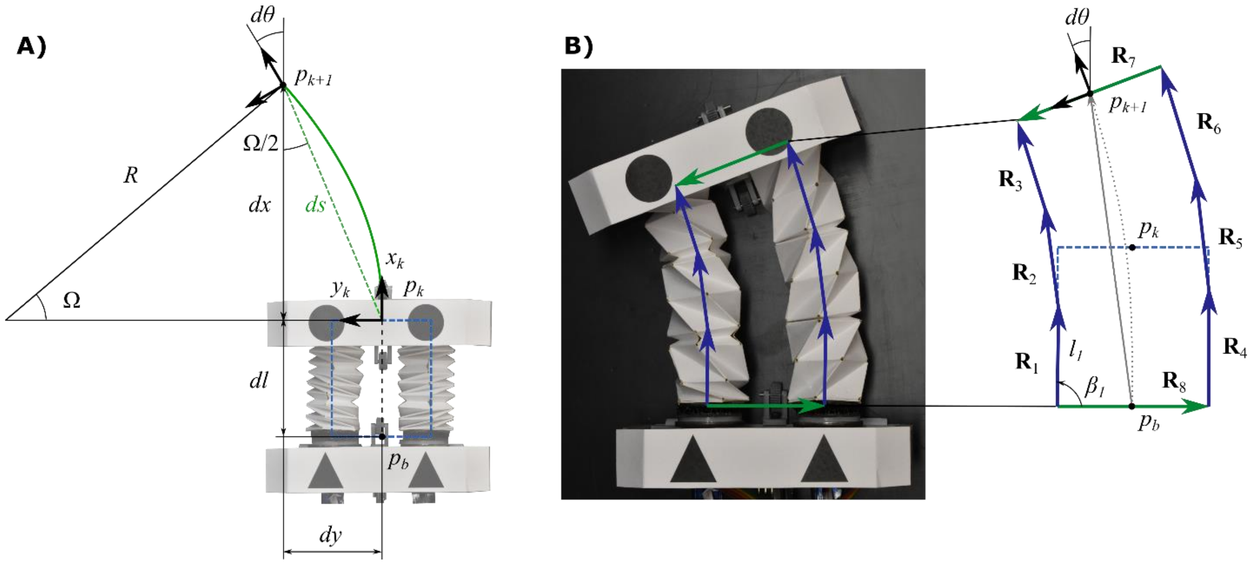


Figure 4.1 Kinematic model consists of two submodels: a lumped kinematic submodel (a) and a segmented kinematic submodel (b). The model on the left represents OSCAR's motion as a rigid body motion from point p_k to point p_{k+1}. The model on the right solves a vector loop corresponding to the robot at the fully expanded state. A vector $p_b p_{k+1} = [dx + dl \quad dy]^T$ in (b) is derived from (a). Based on the assumptions (A1)-(A2), OSCAR's body expands along the arc from the point p_b at the back plate to point p_{k+1}, which defines the orientation increment $d\theta$.

To describe motion, a local coordinate frame is assigned at the centroid of the front plate, as shown in Fig. 4.1. The corresponding robot state is a position and orientation of the local coordinate frame, i.e., $p_k = [x \quad y \quad \theta]_k^T$, where k denotes an index of the current locomotion cycle. Hence, the OSCAR's motion can be described as a rigid body motion of point p_k moving between locomotion cycles.

The kinematic model consists of two submodels: a lumped kinematic submodel (LKS) and a segmented kinematic submodel (SKS) [41]. They are shown graphically in Fig. 4.1, with their hierarchy shown in Fig. 4.2. The LKS is a simplified kinematic model that computes the robot state after the current locomotion cycle p_{k+1}, based on the known current state p_k. Instead, the SKS is a detailed kinematic model that analyzes state p_{k+1} by considering individual origami cell expansions. The SKS computes the required servo angular inputs $u = [\varphi_1 \quad \varphi_2]_k^T$ to achieve the state p_{k+1}, given from the LKS. Together, the LKS and SKS convert the arc radius R and distance ds (see Fig. 4.1) into the robot state p_{k+1} and angular inputs to origami towers φ_i, $i = 1, 2$.

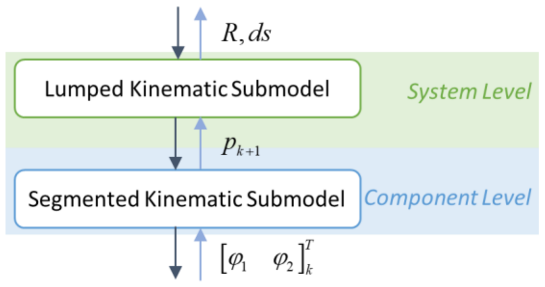

R, ds
Lumped Kinematic Submodel
System Level
p_{k+1}
Segmented Kinematic Submodel
Component Level
$[\varphi_1 \quad \varphi_2]_k^T$

Figure 4.2 Kinematic model hierarchy. Image is taken from [41]

Conversely, the inverse kinematic model (blue arrows in Fig. 4.2) calculates the robot position p_{k+1}, its turn radius R, and total forward displacement ds for given angular inputs φ_i, $i = 1, 2$.

4.1.2 Kinematic Model Idealized Assumptions

The kinematic model is derived under the following idealized assumptions:

A1. Ideal feet friction: The feet have no-backward slippage.

A2. Angular input is evenly distributed: All origami cells within a corresponding origami tower expand equally during robot expansion.

Due to the ideal friction assumption (*A1*), the kinematic model assumes the back plate does not slip backward and remains fixed during the robot expansion, and only the front plate moves forward. Then, during contraction, the front plate remains fixed, and only the back plate moves forward. Thus, the robot position p_{k+1} corresponds to the robot's fully expanded position during the k^{th} locomotion cycle. Due to the equal input distribution assumption (*A2*), the robot body expands along a prescribed arc, starting from the back plate's center p_b to the point p_{k+1} (Fig. 4.1).

4.1.3 Lumped Kinematic Submodel (LKS)

The lumped kinematic submodel (LKS) describes a rigid body OSCAR's motion from known state p_k to p_{k+1} along the arc of radius R with total displacement ds (Fig. 4.1). Therefore, the robot state p_{k+1} is equal to

$$p_{k+1} = p_k + \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} \tag{4.1}$$

where dx, dy and $d\theta$ are the state increments for the current locomotion cycle k. According to Fig. 4.1(a), the state increments can be expressed as

$$dx = ds\cos\left(\frac{\Omega}{2}\right) \tag{4.2}$$

$$dy = ds\sin\left(\frac{\Omega}{2}\right) = \frac{ds^2}{2R} \tag{4.3}$$

$$d\theta = 2\arctan\left(\frac{dy}{dx + dl}\right) \tag{4.4}$$

where dl is the robot length at the fully contracted state, and Ω is the total turn angle

$$\Omega = 2\arcsin\frac{ds}{2R} \tag{4.5}$$

According to the local coordinate frame notation, the left turn corresponds to an orientation increment, while the right turn corresponds to an orientation decrement.

4.1.4 Segmented Kinematic Submodel (SKS)

In the segmented kinematic submodel (SKS), the expansion of individual origami cells defines the angular inputs for the servos φ_i, $i = 1, 2$ for the k^{th} locomotion cycle. The SKS solves a vector loop system, which represents OSCAR in its expanded state; this is illustrated in the right-hand side of Fig. 4.1. In this vector loop, every two consecutive origami cells are represented by rigid links of variable length given by vectors $\mathbf{R}_i,\ i \in [1,\ 6]$. The origami towers' relief cuts are represented as pin joints that allow two connected vectors rotation with respect to each other. The remaining vectors $\mathbf{R}_i,\ i \in [7,\ 8]$ are rigid links of known lengths constrained by the front and back plates.

Therefore, the vector loop of OSCAR's expanded state can be described by a set of equations:

$$\begin{aligned} &\mathbf{R}_1 + \mathbf{R}_2 + \mathbf{R}_3 - 1/2\,\mathbf{R}_7 - 1/2\mathbf{R}_8 - \overrightarrow{p_b p_{k+1}} = 0 \\ &\mathbf{R}_4 + \mathbf{R}_5 + \mathbf{R}_6 - 1/2\,\mathbf{R}_7 + 1/2\mathbf{R}_8 - \overrightarrow{p_b p_{k+1}} = 0 \end{aligned} \tag{4.6}$$

where the vector $\overrightarrow{p_b p_{k+1}} = [dx + dl \quad dy]^T$ with given position increments (4.2)-(4.3). Each vector $\mathbf{R}_i$ has a length l_i and orientation β_i defined similarly as for vector $\mathbf{R}_1$ in Fig.4.1(b). Equation (4.6) can be expressed as $x,\ y$-axis projections in a form

$$\begin{aligned} &l_1 \cos\beta_1 + l_2 \cos\beta_2 + l_3 \cos\beta_3 - \frac{l_7}{2}\cos\beta_7 - \frac{l_8}{2}\cos\beta_8 + dy = 0 \\ &l_1 \sin\beta_1 + l_2 \sin\beta_2 + l_3 \sin\beta_3 - \frac{l_7}{2}\sin\beta_7 - \frac{l_8}{2}\sin\beta_8 - (dx + dl) = 0 \\ &l_4 \cos\beta_4 + l_5 \cos\beta_5 + l_6 \cos\beta_6 - \frac{l_7}{2}\cos\beta_7 + \frac{l_8}{2}\cos\beta_8 + dy = 0 \\ &l_4 \sin\beta_4 + l_5 \sin\beta_5 + l_6 \sin\beta_6 - \frac{l_7}{2}\sin\beta_7 + \frac{l_8}{2}\sin\beta_8 - (dx + dl) = 0 \end{aligned} \tag{4.7}$$

Equations (4.7) should satisfy the following set of geometric constraints

$$\begin{aligned} l_1 = l_2 = l_3 \\ l_4 = l_5 = l_6 \end{aligned} \tag{4.8}$$

$$\begin{aligned} \beta_2 - \beta_1 = \beta_3 - \beta_2 \\ \beta_5 - \beta_4 = \beta_6 - \beta_5 \end{aligned} \tag{4.9}$$

$$\begin{aligned} \mathbf{R}_1 \perp \mathbf{R}_8, \mathbf{R}_4 \perp \mathbf{R}_8 \\ \mathbf{R}_3 \perp \mathbf{R}_7, \mathbf{R}_6 \perp \mathbf{R}_7 \end{aligned} \tag{4.10}$$

and following physical constraints based on OSCAR's design parameters

$$l_7 = l_8 = 44\ mm \tag{4.11}$$

$$l_i \le 30\ mm,\ i \in [1, 6] \tag{4.12}$$

$$|\beta_{i+1} - \beta_i| \le 19^\circ,\ \ i \in \{1, 2, 4, 5\} \tag{4.13}$$

The constraint in (4.8) implies that the angular input φ_i is evenly distributed among the total number N of cells in a tower, $N = 6$. Similarly, the orientation increment $d\theta$ from (4.4) is evenly distributed between two pin joints in the corresponding tower in (4.9). Vectors adjacent to the front and back plates are perpendicular to the plates, to represent the physical assembly. Equation (4.11) indicates the distance between two origami towers. Finally, (4.12) and (4.13) imply that the link lengths l_i and angles $|\beta_{i+1} - \beta_i|$ at relief cuts are limited by their maximum allowable physical values.

The system of nonlinear algebraic equations with corresponding constraints (4.7)-(4.13) is solved numerically for the unknown link lengths and orientations, l_i and β_i, using the *fmincon* function in MATLAB. Based on assumption (*A2*), the servo inputs are given by

$$\begin{aligned} \varphi_1 &= N\alpha(l_1/2) \\ \varphi_2 &= N\alpha(l_4/2) \end{aligned} \tag{4.14}$$

Here, $\alpha(\cdot)$ is the input angle to an individual origami cell as the function of the origami cell's length [40]. This function is represented graphically in Fig. 2.8 in Chapter 2.

4.2 Kinematic Model Validation

This section presents the kinematic model validation for the experimental data shown in Fig. 3.10 in Chapter 3. While the kinematic model assumes ideal friction at the feet on the robot, some backward slippage is unavoidable in the actual feet even with the optimized designs, as highlighted in Chapter 2. This nonideal friction causes small backward slippage of the back plate during expansion. Hence, the front plate's total forward displacement is decreased from the idealized kinematic representation. For the same reason, the front plate slips backward by some amount during contraction. Therefore, the actual robot state, defined at the front plate, is unavoidably overestimated by the kinematic model prediction (Fig. 4.3). Moreover, the slippage, and hence the model accuracy, is highly dependent on the interaction between the environment and the robot feet. Therefore, it is entirely feasible that this could change during locomotion as the ground environment changes which presents an inherent challenge for a controller.

To validate the kinematic model, the inverse SKS outputs have been compared with experimental data. Here, the experimental data contains OSCAR's displacement and orientation changes for the range of static angular inputs $\begin{bmatrix}\varphi_1 & \varphi_2\end{bmatrix}^T$ that cover the OSCAR's achievable workspace. Due to the nonideal friction, the robot has losses in the incrementing between cycles of both predicted displacement and orientation. To adjust for that and match the data, the following empirical correction has been introduced to the inverse SKS outputs

$$d\tilde{\theta} = \gamma d\theta \tag{4.15}$$

$$\begin{aligned} d\tilde{x} &= \sqrt{\left(dx + d\tilde{l}\right)^2 + dy^2}\cos\frac{d\tilde{\theta}}{2} - d\tilde{l} \\ d\tilde{y} &= \sqrt{\left(dx + d\tilde{l}\right)^2 + dy^2}\sin\frac{d\tilde{\theta}}{2} \end{aligned} \tag{4.16}$$

where $\begin{bmatrix}dx & dy & d\theta\end{bmatrix}^T$ are position and orientation increments output by the SKS inverse for the angular inputs $\begin{bmatrix}\varphi_1 & \varphi_2\end{bmatrix}^T$. These angular inputs are the same as those from experimental data because the servo motor closed loop ensures the motors achieve their desired rotational positions. Here, γ is the empirically determined efficiency factor for the orientation increment, $\gamma = 0.15$, and $d\tilde{l}$ is the corrected length of the fully-contracted origami towers, $d\tilde{l} = 41mm$. The $d\tilde{l}$ is applied instead of dl in (4.4)-(4.7) in the corrected kinematic model.

In the above equations, (4.15) accounts for the robot orientation loss by calculating an adjusted orientation increment $d\tilde{\theta}$, while (4.16) adjusts the state increments to reflect that in adjusted displacements $\left(d\tilde{x}, d\tilde{y}\right)$. As stated above, OSCAR has only a 15 % turning efficiency compared of the ideal model being 100%. The displacement loss is accounted by $d\tilde{l}$, whose adjusted value is stated above. In the ideal kinematic model, $dl = 23mm$ [41]. The experimental value of $d\tilde{l} = 41mm$ is found by measuring the contracted origami towers' length in the assembly guide (see Chapter 2).

Comparison of the experimental data with the ideal (before correction) and corrected kinematic model is presented in Fig. 4.3 and Fig. 4.4, respectively. The inverse SKS has been used for comparison. As can be seen, the ideal kinematic model (green dashed lines) significantly overpredicts the experimental data (dotted lines) (Fig. 4.3). Instead, the corrected kinematic model (grey dashed lines), i.e., with applied (4.15)-(4.16), accurately matches the experimental data (dotted lines), as shown in Fig. 4.4. As can be seen, the corrected model predictions remain close to the actual OSCAR motion across multiple locomotion cycles.

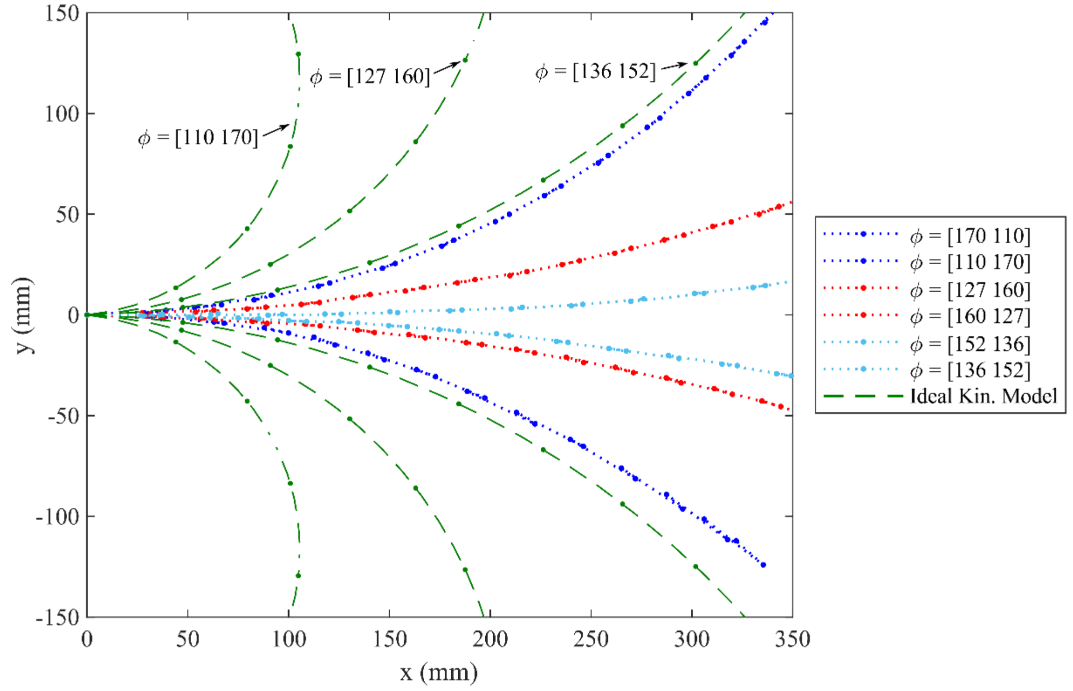


Figure 4.3 Comparison of experimental data and ideal kinematic model predictions for the same angular inputs. The angular inputs (in degrees) are listed on the right

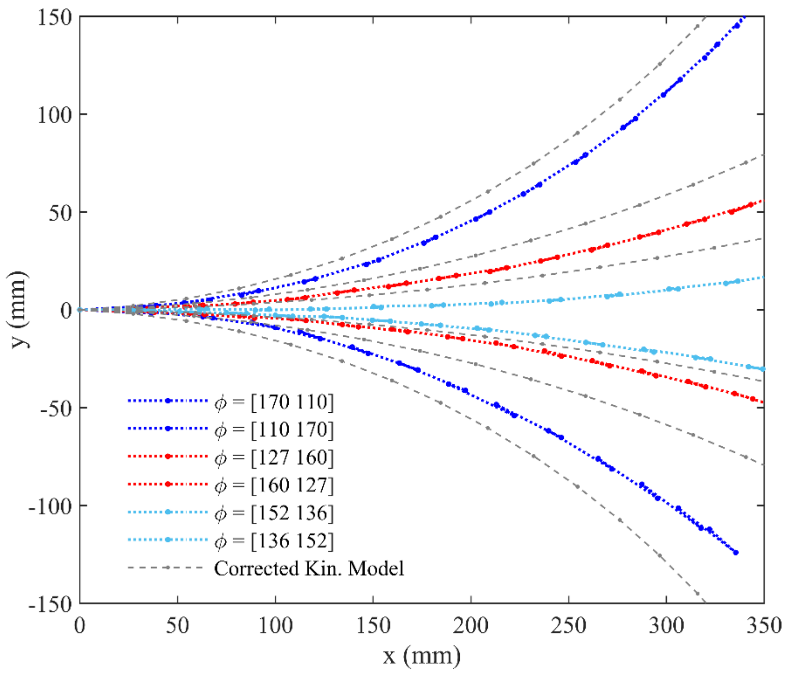


Figure 4.4 Comparison of experimental data and corrected kinematic model predictions for the same angular inputs. The angular inputs (in degrees) are listed in the column on the bottom left

The OSCAR's achievable workspaces for the ideal kinematic model (left) and the corrected kinematic model (right) are shown for comparison in Fig. 4.5. The OSCAR's achievable workspace is defined as a set of all positions that it can achieve within a single locomotion cycle. The (0,0) origin points in Fig. 4.5 indicate the front plate's center at its initial state, and green arcs show their trajectory during the locomotion cycle. The results in Fig. 4.5 demonstrate, quite dramatically, the reduction in the workspace due to the reduced locomotion efficiency which is due in large part to the robot environment interaction. As a result of the displacement and orientation losses, the corrected workspace (right) is significantly narrower than the ideal one (left) and the maximum distance per locomotion cycle is 30-50 % smaller. The net knowledge result is that the actual motion capability of the physical robot is significantly less than what would be predicted by an idealized model. This information is very valuable because it feeds the motion planning algorithms, so they only ask for robot motions that stay within the robot's achievable constraints.

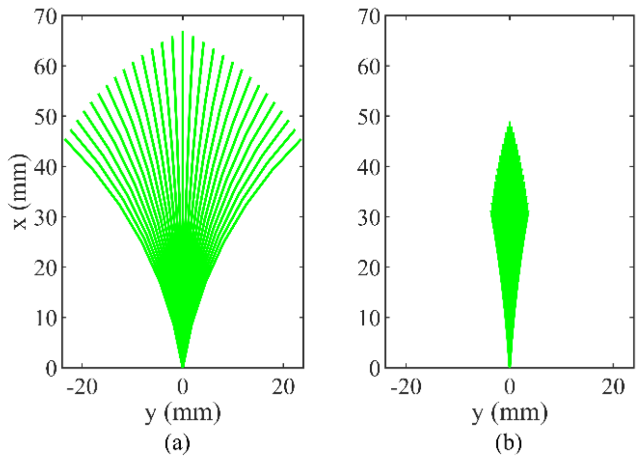


Figure 4.5 Comparison of achievable workspace for ideal kinematic model (a) and corrected kinematic model (b)

To apply the above correction in the forward kinematic model, (4.15) is inverted and (4.16) is reevaluated for the state increments $\left[d\tilde{x} \;\; d\tilde{y} \;\; d\tilde{\theta} \right]^T$ from the LKS

$$d\theta = \left(1/\gamma\right) d\tilde{\theta} \tag{4.17}$$

$$\begin{aligned} dx &= \sqrt{\left(d\tilde{x} + dl\right)^2 + d\tilde{y}^2}\cos\frac{d\theta}{2} - dl \\ dy &= \sqrt{\left(d\tilde{x} + dl\right)^2 + d\tilde{y}^2}\sin\frac{d\theta}{2} \end{aligned} \tag{4.18}$$

Then, the updated state increments from (4.17) and (4.18) are used in the SKS to calculate the required angular inputs.

Chapter 5
Path Following Control

5.1 Motivation

The path tracking or motion control is the central aspect of autonomous navigation [51]. This problem can be formulated as path following or trajectory tracking. In path following, the robot has to reach and follow a given geometric path without necessarily concerning itself with timing along the path. Instead, in trajectory tracking, the robot needs to follow a trajectory, a geometric path with associated timing law. Both problems have been well understood and studied for rigid-body robots, like autonomous vehicles or rigid mobile robots, as reviewed below.

In rigid-bodied mobile robots, the path tracking controllers continuously determine steering angle and velocity inputs to allow the reference path following. The existing control solutions have been reviewed in [52], as well as in [53] and [54]. The most widely implemented class of controllers is the geometric controllers, which utilize the system's geometric model. As a result, they are both simple to implement and efficient [52], [54]. The most popular geometric controller is a pure pursuit [55], [56]. In this method, the robot is constantly pursuing a reference point on the path ahead of the robot. Another example of the geometric control is the Stanley controller [57]. It utilizes the lateral and heading errors to the reference path in the steering angle control law. Besides the geometric controllers, there is the class of kinematic controllers. These are the feedback controllers based on the kinematic model, e.g., as presented in [53]. In most controllers, the kinematic bicycle model is used. The geometric and kinematic controllers are designed for path tracking in moderate conditions and may not be suited for path tracking in high-speed conditions, as they ignore rigid-body robot dynamics. Instead, the dynamic controllers account for the system dynamics. However, they are computationally expensive and should be selected based on a tradeoff between the model fidelity and computational complexity. The variety of control methods also include classical controllers, such as PID and sliding model control (SMC), as well as optimal and adaptive controllers, such as linear-quadratic regulators (LQR) and model reference adaptive controllers (MRAC), as listed in [52]. These controllers show good path following but have challenges in parameter tuning (PID) and may be sensitive to path curvature variation (SMC, adaptive controllers). Additionally, path following has been realized with the model predictive controllers (MPC). MPC predicts system behavior for a short time prediction horizon by solving an optimization problem. It then, selects the control input for a single time step and repeats the calculation. A linear MPC allows path tracking for limited conditions, while nonlinear MPC may allow accurate path following over a wide range of dynamics and operating conditions. However, it is computationally expensive as it requires solving a nonlinear optimization problem at each step, which limits its application. To address this problem, nonlinear system dynamics has been linearized at each step and linear MPC was applied at each step in [58]. The advantages and drawbacks of all categories of controllers have been summarized in a table in [52].

Task-level motion control is still underrepresented in soft mobile robots, as reviewed in Chapter 1. Due to their compliance, these robots are underactuated. They suffer from motion and model uncertainties, which leads to additional challenges for their autonomous navigation compared to the rigid-bodied robots. Hence, designing the controller enabling accurate path following is crucial for soft robots' autonomous navigation. In OSCAR, the previously mentioned uncertainties have been addressed in the earlier chapters. Since OSCAR has a relatively slow motion, in this thesis we focus on the path following and consider the trajectory tracking to be out of scope.

The path following control problem is formulated as follows in this work. Given a reference path as a set of waypoints, $p_{path} \in \mathbb{R}^2$ in the global coordinate frame, the robot should converge to and follow the path [51]. By utilizing the rigid body approximation for OSCAR's motion, described in Chapter 4, control algorithms traditionally used in the rigid-bodied robots can be adapted to OSCAR. This work employs two path following controllers: an adapted model-based pure pursuit controller and a proportional feedback controller.

5.2 Model-Based Open-Loop Pure Pursuit Controller

As stated above, the pure pursuit controller is one of the most popular and effective controllers for rigid-bodied robots, due to its computational simplicity and robustness [52],[59]. The pure pursuit continuously regulates the robot steering angle by fitting an arc between its current position and the goal point on the path. The goal point is found at a specified lookahead distance L from the robot's current position. The resulting arc curvature defines the steering angle [4].

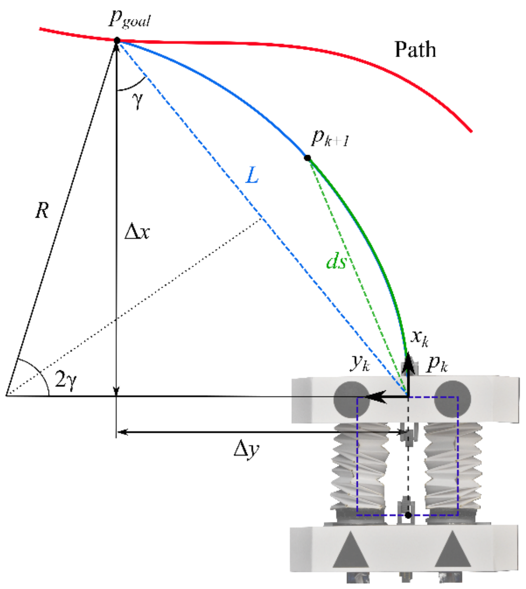


Figure 5.1 Pure pursuit controller schematic

Similarly, for OSCAR, an arc is fitted between its current position p_k and the goal point p_{goal} located in the lookahead distance L on the path, as shown in Fig. 5.1. Geometrically, $\sin\gamma = \frac{\Delta y}{L}$ and $\sin\gamma = \frac{L}{2R}$. Therefore, the resulting arc has a radius

$$R = \frac{L^2}{2\Delta y} \tag{5.1}$$

where Δy is the lateral error to p_{goal} in the robot's local coordinate frame. Equation (5.1) is subjected to the minimum radius constraint

$$R \geq R_{\min} \tag{5.2}$$

which results from the robot's achievable workspace, shown in Fig. 4.5. This constraint represents the limits of the robot maximum curvature turn, as $c_{\max} = 1/R_{\min}$.

To adapt the pure pursuit for OSCAR, its motion within a single locomotion cycle along the defined arc is considered, as shown in Fig. 5.1. The displacement ds defines the robot's motion. The ds is a user-defined constant parameter, also constrained by the achievable workspace $ds \in \left[0,\ ds_{\max}\right]$. The outputs R and ds of the pure pursuit algorithm are substituted to the kinematic model to determine the angular inputs $\left[\varphi_1 \quad \varphi_2\right]_k^T$. Since the kinematic model is a part of the resulting controller, the adopted pure pursuit is an open-loop controller. During path following, the controller iteratively computes the angular inputs for each locomotion cycle.

5.3 Proportional Feedback Controller

The feedback path following controller in this work has the separate longitudinal and lateral control inputs that are coupled to calculate the robot's angular inputs. This is analogous to the path following in the rigid-bodied robots, where the decoupled longitudinal control input corresponds to the forward speed and the lateral control input regulates the steering angle to compensate the lateral error to the path [60]. Here, the longitudinal and lateral directions correspond to the x and y-axes of the local coordinate frame, as shown in Fig. 5.2. Assuming the robot moves with a constant speed, its longitudinal control input is constant

$$u_{x,k} = \rho u_{x,\max} \tag{5.3}$$

where ρ is a constant chosen to be in $\rho \in \left[0,1\right]$, and $u_{x,\max}$ is the maximum longitudinal input, equal to the maximum angular input to the origami tower, $u_{x,\max} = \varphi_{\max}$. The subscript *k* indicates the index of the current locomotion cycle.

The lateral control input is defined by a proportional controller

$$u_{y,k} = K_p e_{y,k} \tag{5.4}$$

where K_p is a controller gain and $e_{y,k}$ is the lateral error defined between a point along the x-direction in the preview distance *D* and the path, as shown in Fig. 5.2. Here, the preview distance *D* is a user-controlled tunable parameter, tuned to tradeoff the controller aggressiveness with stability.

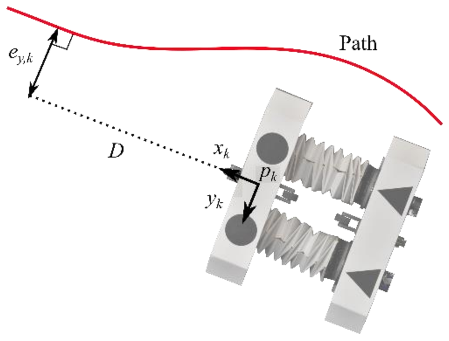


Figure 5.2 Lateral error in the feedback controller

The control inputs in (5.3) and (5.4) are coupled with the angular inputs to the origami towers according to

$$\begin{aligned} (\varphi_1 + \varphi_2)_k / 2 = u_{x,k} \\ (\varphi_2 - \varphi_1)_k / 2 = u_{y,k} \end{aligned}\ . \tag{5.5}$$

Equation (5.5) states that the angular inputs' average is proportional to the robot forward displacement, denoted by the longitudinal control input $u_{x,k}$, while their difference corresponds to the turning motion, denoted by the lateral control input $u_{y,k}$. Based on (5.5), the angular inputs to the *k*th locomotion cycle are

$$\begin{bmatrix} \varphi_1 \\ \varphi_2 \end{bmatrix}_k = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix}_k \tag{5.6}$$

The calculated angles are subjected to the following set of constraints

$$0 \le \varphi_i \le \varphi_{\max}, \quad i = 1, 2 \tag{5.7}$$

$$\frac{1}{\bar{r}} \le \frac{\varphi_2}{\varphi_1} \le \bar{r} \tag{5.8}$$

which states that the reference angular input to the origami tower must be positive and bounded (5.7). Moreover, the angular ratio should be limited (5.8). Here, $\bar{r}$ is the maximum angular ratio and it corresponds to the maximum right turn. The ratio $1/\bar{r}$ corresponds to the maximum left turn. In these constraints, $\varphi_{\max} = 180^\circ$ and $\bar{r} = 1.6$, as stated in Chapter 3. Based on (5.6), the constraint in (5.7) can be expressed as

$$\left|u_{y,k}\right| \le u_{x,k}\,\frac{\bar{r}-1}{\bar{r}+1}\ . \tag{5.9}$$

5.4 Simulation and Experimental Results

The designed path following controllers' performance has been investigated for the straight horizontal reference path. For this case study, OSCAR starts at some initial offset in the y-direction from the path. Its initial state is approximately $p_0 = \begin{bmatrix} 0 & -60 & \pi/6 \end{bmatrix}^T$ in all experiments and simulations; the units are mm and radians, respectively. OSCAR has to converge to and follow the reference path with bounded position and orientation errors.

5.4.1 Controllers' Performance with the Ideal Kinematic Model

In the simulation, the pure pursuit has a lookahead distance $L = 110$ mm and a forward displacement $ds = 48$ mm. The robot minimum turn radius is $R_{\min} = 110$ mm derived from the ideal achievable workspace, shown in Fig. 4.5(a). The feedback controller has an empirically determined gain $K_p = 0.7$ and preview distance $D = 100$ mm. In all simulations and experiments presented in this Chapter, the longitudinal input constant in the feedback controller is $\rho = 0.8$, and forward displacement ds in the pure pursuit is tuned to match the resulting forward displacement in the feedback controller. All the control parameters were tuned for the best performance in terms of overshoot and convergence to the path for the given operating conditions.

The path following simulation results with both controllers using the ideal kinematic model are shown in Fig. 5.3. The robot acts in an ideal way in simulation, so the robot plant and kinematic model corrections, (4.15)-(4.16) and (4.17)-(4.18), respectively, are not needed. As shown in Fig. 5.3, the robot acquires and follows the path with both controllers.

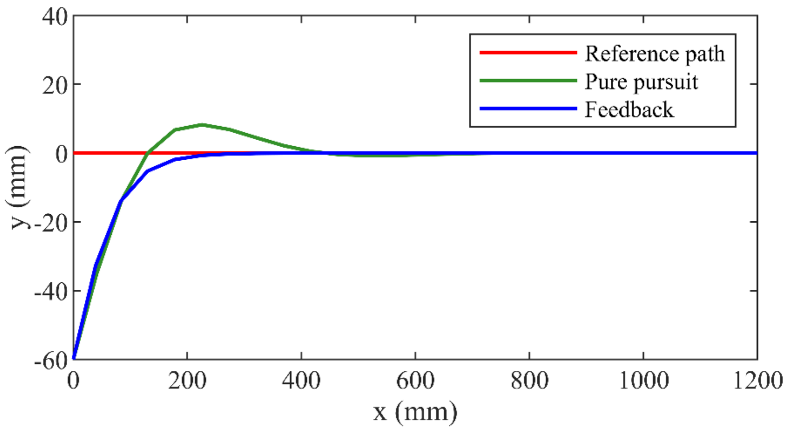


Figure 5.3 Simulation results for the path following with the ideal kinematic model

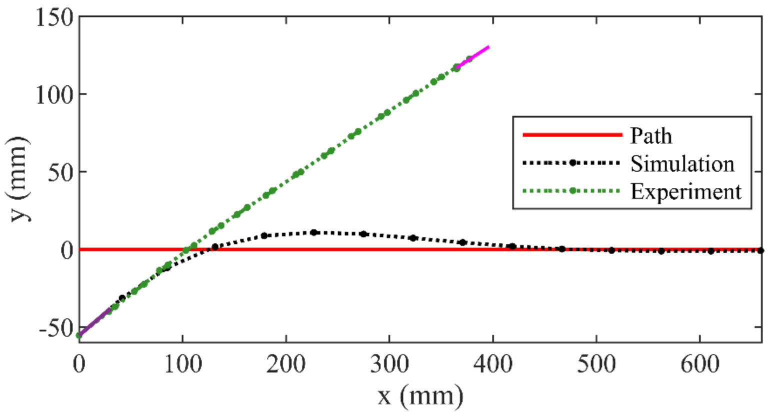


Figure 5.4 Robot trajectory with the pure pursuit controller with the ideal kinematic model. Due to modeling errors, OSCAR's trajectory (green) diverges from the path (red) and simulation (black). Purple and pink solid lines show the robot orientation at the initial and final states, respectively

However, as shown in Fig. 5.4, the experimental results are not analogous to those predicted by simulation. The experimental validation of the pure pursuit shows that the robot diverges from the path, due to significant modeling errors in the ideal kinematic model and motion uncertainties present in the robot. The adopted pure pursuit is an open-loop controller that relies solely on a kinematic model. Since the idealized kinematic model does not account for the system inefficiencies and losses, primarily due to the robot-environment interaction, it overpredicts the robot state at each cycle and greatly diminishes the pure pursuit path acquisition and tracking performance. This illustrates the need for the previously mentioned corrections in the kinematic model.

5.4.2 Pure Pursuit Performance with the Corrected Kinematic Model

In the pure pursuit with the applied correction in the kinematic model, the controller gains are $L = 100$ mm and $ds = 30$ mm. These values were determined empirically. The minimum turn radius is $R_{\min} = 200$ mm due to the corrected kinematic model.

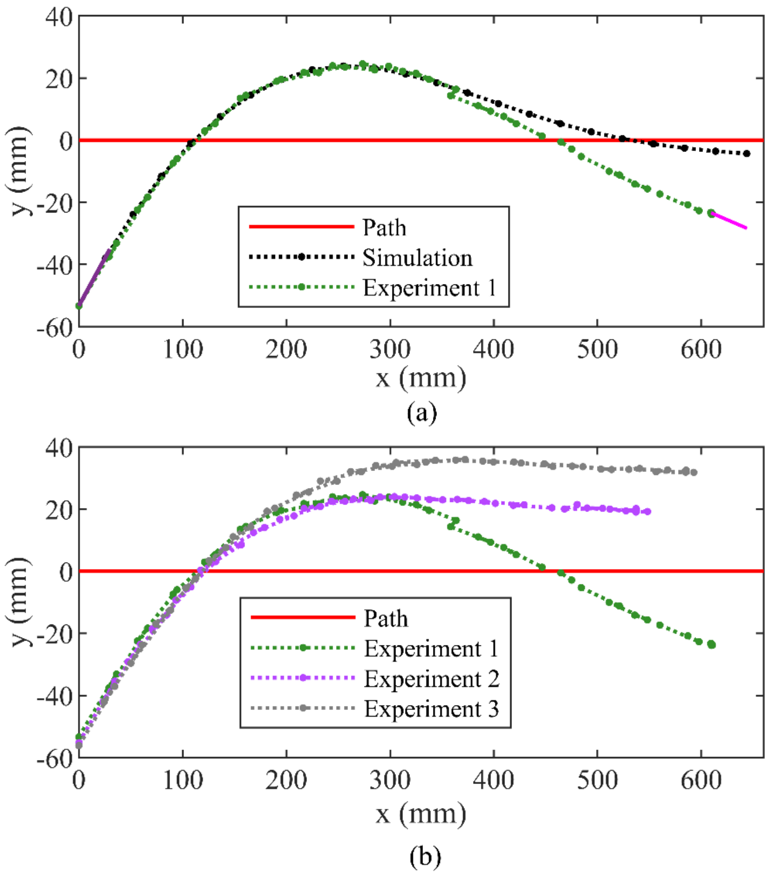


Figure 5.5 (a) Robot trajectory with the pure pursuit controller with the corrected kinematic model. Purple and pink solid lines show the robot orientation at the initial and final states, respectively. (b) Trajectory divergence for the three consecutive experiments

Fig. 5.5(a) presents the simulation and experimental results with the corrections. Although the performance is much better than the un-corrected approach, the robot still cannot acquire and follow a path. The experimental behavior deviates from the simulated behavior due to remaining uncertainty between the modeled and actual system. To demonstrate the variability of the results, three consecutive experiments are shown in Fig. 5.5(b). As can be seen, the variability is not repeatable from one experiment to the next. This indicates that the robot-terrain interaction is very difficult to capture and use for pure-pursuit.

The analysis shows that the OSCAR has significant motion uncertainties caused primarily by its interaction with the ground and secondarily by some uncertainty in the origami towers' compliance. As a result, there is a strong need for the feedback controller.

5.4.3 Experimental Results for the Feedback Controller

For the experimental studies with the feedback controller, the gain and preview distance are empirically tuned to $K_p = 0.7$ and $D = 250$ mm, respectively. Figure 5.6 shows the comparison of the experimental and simulation results, obtained with the corrected kinematic model that accounts for motion inefficiencies. The results of two combined experiments are presented in Fig. 5.6(a). Due to the limited workspace in the experimental setup, the overall response has to be done in two sequential phases. In Fig. 5.6(a), the first experiment is started and run to the 600+mm (x-axis) limit of our experimental workspace. Then, the final values of the robot orientation and y-axis offset from the first experiment are used for the second experiment starting at beginning of the x-axis. This can be done since the robot dynamics are not a significant factor. Stitching the two responses together gives the overall response in Fig. 5.6(a).

The experimental results illustrate that the feedback controller allows the robot to acquire and follow a path despite the present motion uncertainties. Their relative amount can be observed by comparison of simulation and experimental results. In the experiment, the robot has a larger overshoot than predicted in simulation but the steady-state error in the y-direction remains within measurement error of the simulated result, which is acceptable. The larger overshoot is caused by unknown uncertainties not captured in simulation. Additionally, the experiment 1 and 2 have been conducted three times to verify the repeatability of the controller performance results, as shown in Fig. 5.6 (b) and (c).

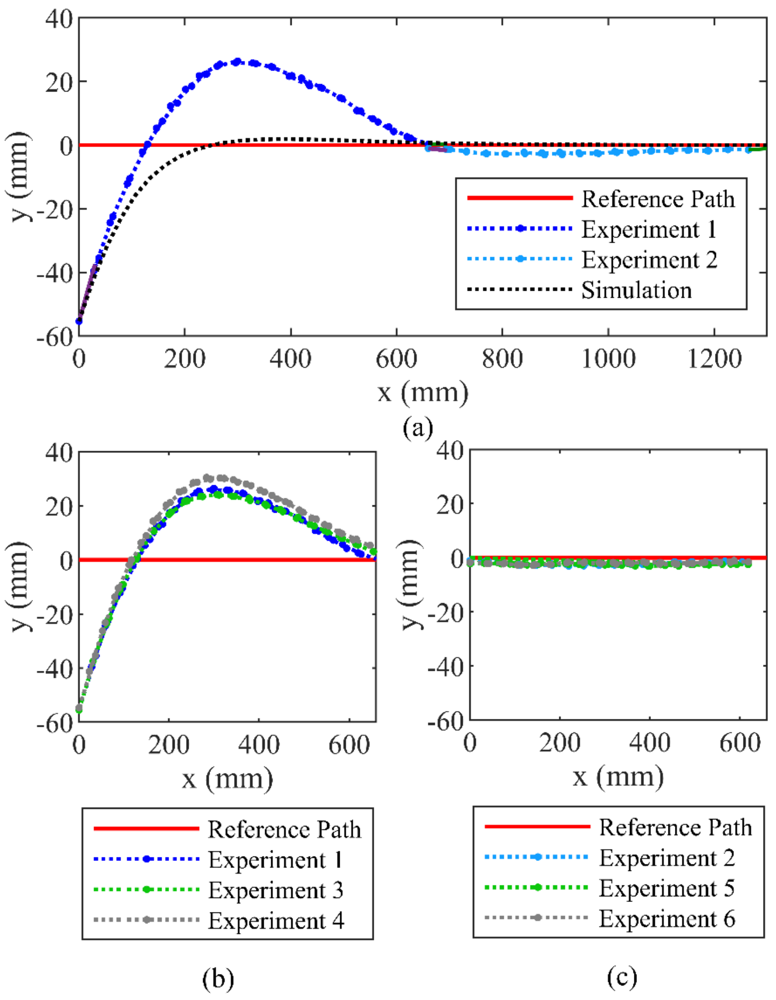


Figure 5.6 Robot trajectory with the feedback controller: (a) combined experimental results showing the robot acquiring and following the path. The robot final orientation in experiment 1 matches the initial orientation in the experiment 2. (b) and (c) show the robot trajectory repeatability with the feedback controller for three trials in settings of experiment 1 and 2, respectively

The angular inputs for simulation and combined experiments 1 and 2 are shown in Fig. 5.7. The rest of the experiments in Fig. 5.6 (b) and (c) have similar inputs and thus are not shown. The presented angles in Fig 5.7 are expansion inputs shown for each locomotion cycle. As can be seen, the angular inputs are initially saturated, as the robot makes maximum allowed left turns $(\varphi_1 > \varphi_2)$ to converge to the path in both simulation and experiment. As the robot approaches the path (see the second half of experiment 1 and beginning of experiment 2 in Fig.

5.6 and Fig. 5.7), it corrects for overshoot by making the right turns $(\varphi_2 > \varphi_1)$. In the simulation, the predicted overshoot is smaller than in the experiment, and therefore, almost no right turns are needed. As the robot converges to and moves along the path, both simulation and experimental angular inputs become the same.

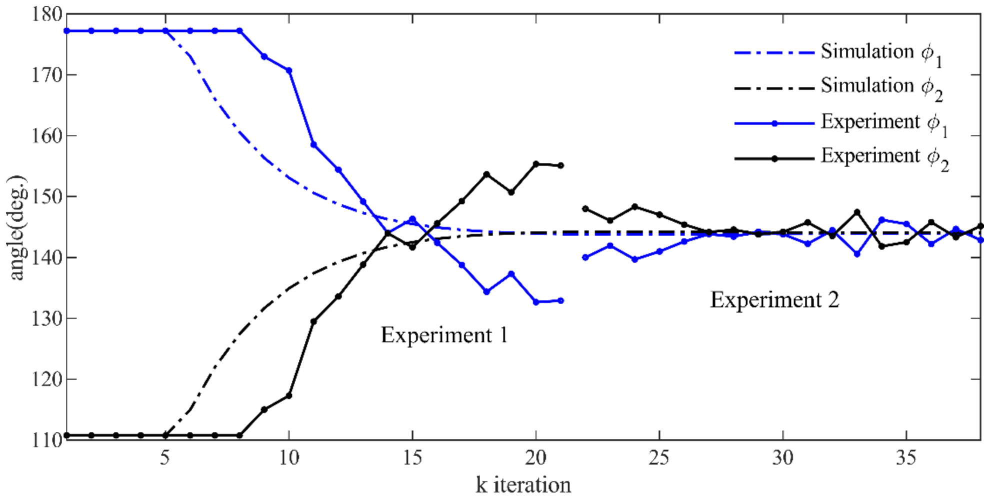


Figure 5.7 Angular inputs for path following with feedback controller in Fig. 5.6(a)

5.5 Chapter Summary

As demonstrated in this Chapter, path following is feasible for the given class of soft origami robots. Two path following efforts were initiated in this work. One used a well-known model-based pure pursuit controller, and the other was a simple feedback controller using a lookahead distance. In the simulation, both worked well enough to be a viable path following approach. However, the uncertainties prevalent in the overall system drastically reduced the efficacy of the open-loop model-based approach. These uncertainties included the robot-terrain interaction, the origami-tower flexibility, as well as lower-level uncertainties in items such as servo deadband that were dealt with in Chapter 3. The results given here indicate that feedback control for path acquisition and path following is a strict requirement for future autonomous navigation of soft origami robots. The feedback controller is used in the OSCAR autonomous navigation experiments in Chapter 6, where it demonstrated accurate path tracking and static obstacle avoidance.

Chapter 6
Path Planning and Autonomous Navigation for OSCAR

6.1 Motivation

The robot requires three main components to enable autonomous navigation commonly known as perception, planning, and control [61], [62]. The perception is the ability to receive information about the robot's current state and environment. With a known state and environment, planning allows the robot to plan a collision-free path to its goal configuration intelligently. Finally, the robot executes this path using a controller.

As explained in Chapter 3, OSCAR has offboarded localization in the experimental setup to determine its state. Also, OSCAR has a path following controller to follow a provided reference path, as presented and tested in Chapter 5. Therefore, the only missing part in the autonomous navigation framework is path planning, the subject of this chapter.

6.2 Background

Before describing the path planning algorithms investigated, let us first introduce the essential concepts. Commonly, the rigid-body robot configuration is defined by its planar position and orientation, i.e., $p = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$, $p \in C$ [63]. Then, the configuration space C is a space of all configurations [64]. In this case, $C = \mathbb{R}^2 \times [0,\ 2\pi)$, which defines the robot motion by planar translation and 2D rotation. The path planning is performed in the configuration space, where the robot is simplified to a point denoting its current configuration.

Consider static obstacles $O_i,\ i=1,...,n$ in the robot workspace $W \in \mathbb{R}^2$. These obstacles are rigid body objects in the workspace W. We define the robot shape to be $R(p)$ at configuration p. Then, the obstacles in configuration space can be defined as

$$CO_i = \left\{ p \in C \middle| \, R(p) \bigcap O_i \neq \varnothing \right\},\ i=1,...,n \tag{6.1}$$

The free configuration space is the configuration space without obstacles, i.e., $C_{free} = C \setminus \bigcup_{i=1}^{n} CO_i$. Thus, for collision avoidance the path planning should be done in C_{free} [64].

Generally, the path planning problem can be formulated as planning a feasible path from the initial configuration p_0 to the given final configuration p_{Goal} while avoiding obstacles. The obstacles could be static and/or dynamic. Feasibility means the robot can execute the path, i.e., the path should satisfy the robot motion constraints. Additionally, the path optimality could be imposed, which means that the planned path should be optimal (i.e., minimize a cost function) [53].

For planning the path, the configuration space is often discretized into a grid that forms a graph. The graph $G(v,e)$ is a structure that consists of vertices $v \in V$ and edges $e \in E$, where the edges connect two adjacent vertices. For example, for two vertices v_1 and v_2, the edge is a pair (v_1, v_2). A single vertex can have multiple edges. In the discretized configuration space, the resulting grid nodes form graph vertices, and the pairs of adjacent nodes form graph edges. The resulting graph is presented as an occupancy grid [65]. The occupancy grid contains information about obstacles, such that the grid cells that are occupied by obstacles are marked as unavailable.

Path planning has been an active research area for rigid-bodied mobile robots, like autonomous vehicles. The existing motion planning algorithms are reviewed in [53], [61], [66]. These methods can be categorized into the following three categories:

1. *Graph-search methods* discretize the continuous configuration space and represent it as a graph, as described above. The algorithm then searches the graph to find the minimum cost path by growing a search tree [53]. The graph-search methods include Dijkstra’s algorithm [67], the A* algorithm [68], its variation

called the hybrid A* algorithm [69], and the D* algorithm [70], among others. Additionally, these methods include state lattice-search methods, where the graph is obtained by uniformly discretizing the configuration space with a set of motion primitives. The resulting graph is called a state lattice, over which the above search methods are applied. The graph-search methods are guaranteed to find a path if one exists. However, the deterministic sampling of the whole configuration space in these algorithms makes the search problem computationally complex for high-dimension or large-sized configuration spaces.

2. *Incremental-search methods* plan a path by randomly sampling the configuration space and incrementally building the graph [61],[66]. When the graph is large enough to connect the start and goal region, the search method traces graph nodes from the start to goal configurations and outputs the resulting path. Examples of the incremental-search methods are probabilistic road maps (PRM) [71] and rapidly exploring random tree (RRT) methods, e.g., RRT [63] and RRT* [72]. Due to random sampling, these methods find solutions faster than graph-search methods for the high-dimension configuration spaces. These spaces include, for example, velocity or acceleration as additional states in the configuration space. However, their computational time can be unbounded if the solution does not exist or if the search is not guided correctly by the heuristic that drives the random sampling [53].
3. *Variational methods* solve the non-linear trajectory optimization problem in the space of parametrized curves to find a path. These methods divide into direct and indirect methods based on how the optimization problem is solved [73]. As highlighted in [53], variational methods converge to local minima solutions. The appropriate initial guess is needed to obtain the global minimum solution.

Given a well-defined workspace and static obstacles, the hybrid A* path planning algorithm has been chosen in the proposed framework for its simplicity and ease of implementation. Hybrid A* allows for planning a smooth path while satisfying OSCAR motion constraints. It is explained in detail in the following section.

6.3 Hybrid A* Planner

The hybrid A* method is a graph search algorithm designed for robots with non-holonomic motion constraints, e.g., autonomous cars [69], [74]. The non-holonomic constraint means that the robot cannot directly move laterally, as its lateral motion is coupled with the forward displacement [63]. The hybrid A* planner was first used in an autonomous car Junior during the DARPA Urban Challenge in 2007 [69]. Unlike other discrete graph search algorithms, e.g., Dijkstra's and A*, hybrid A* plans a path in the continuous space, making the planned path feasible for a non-holonomic robot [69].

To accommodate the non-holonomic constraint, the hybrid A* grows the search tree along a set of precomputed motion primitives $\mu(\theta,\delta)$, obtained by discretizing the available range of motion. Here, δ is a multiple of discretization steps and θ is the current orientation. It is represented by the set of arcs of fixed length, as shown in Fig. 6.1. The algorithm's pseudocode is presented in Algorithm 1, adopted from [74].

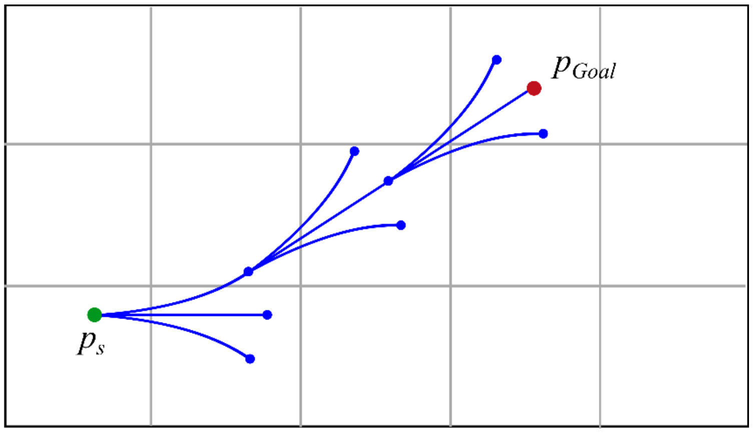


Figure 6.1 Graphical explanation of hybrid A* path planning algorithm

Consider the configuration space is discretized into a grid and given by an occupancy map m (Fig. 6.1). For a 2D position x of the current configuration p, its corresponding discrete position on the map (denoted by a hat) is

$$\hat{x} = (x - o_m) / \zeta \tag{6.2}$$

where o_m is the map origin, and ζ is the map resolution. The discrete position is stored alongside the actual position, where the discrete position is used for collision avoidance checking. It is also used to update the nodes already existing in the graph with the same discrete position if the new nodes have a lower cost. During the search, the node expands (i.e., search tree grows) from the actual position, ensuring path feasibility.

The algorithm uses two cost functions: *cost-to-come* $g\left(p_s, p\right)$, which is an accumulated cost from the start to the current node, and *cost-to-go* $h\left(p, p_{Goal}\right)$, called a heuristic. The heuristic is a distance estimate from the current to the goal node. The valid heuristic needs to be admissible, i.e., it should be a lower bound of the true cost [63]. An example of a valid heuristic is the 2D Euclidian distance between the two given nodes. The heuristic is critical for the search, as it guides the algorithm and avoids expanding nodes far from the optimal path. Thus, it reduces the computational time to find a path. The total cost at each node is defined as a sum

$$f\left(p\right) = g\left(p_s, p\right) + h\left(p, p_{Goal}\right) \tag{6.3}$$

Based on the above, each node of the search tree can be fully defined by

$$p = \left(\hat{x},\, \hat{\theta},\, x,\, g,\, f,\, n_p\right) \tag{6.4}$$

where $\hat{x}$ and $\hat{\theta}$ are discrete position and orientation; x is an actual position; g, f are *cost-to-come* and total cost, respectively; and n_p is a predecessor node. Here, the discrete and actual orientations are the same $\hat{\theta} = \theta$, assuming $\theta_s = \hat{\theta}_s$ [74]. The information about the predecessor node is stored in order to reconstruct the resulting path if it exists. Finally, the algorithm utilizes two sets: an open set Q containing adjacent nodes of already expanded nodes and a closed set R of all processed nodes.

Given the initial configuration p_s, the algorithm defines the start node of the search graph; see line 1 in Algorithm 1. This node has zero cost-to-come and no predecessor. The algorithm then pushes node p_s into the open set Q and defines an empty set *R*. Line 4 starts the *while* loop, which terminates if path from the p_s to p_{Goal} is found or no path exists. Inside the loop, the node *p* with the lowest *f* value is selected from *Q*. When node *p* is expanded, it is moved from the set *Q* to the set *R* (lines 6 and 7). If *p* is in the region of p_{Goal}, then the planned path is reconstructed through the predecessor list starting from the node *p*.

Otherwise, the successor p' of p will be generated from the motion primitives' set. If the successor is in collision with the obstacle, it will be discarded and added to the closed set *R*. The corresponding search tree branch will be pruned (deleted) [75]. Alternatively, it will be compared with elements of *Q*. If there is a node in Q with the same discrete position as p', the *cost-to-come* g' will be calculated and compared to that of the existing node in *Q* (line 16). Here, $l(p, p')$ is the length of the motion primitive [75]. If g' is lower than g, the existing node in *Q* will be replaced by p'. The node p' has predecessor *p* and calculated costs f' and h'. Otherwise, the node p' will be added to *Q* if it is not in *Q*. These steps will repeat until all motion primitives are expanded from *p* and processed.

Finally, the iterations will continue until the path is found (line 9), or there are no more elements in Q. In the latter case, the algorithm returns that there is no path found (line 26).

Algorithm 1: Hybrid A* search [74]

```
1.  $p_s \leftarrow \left(\hat{x}_s, \hat{\theta}_s, x_s, 0, h(x_s, G), -\right)$
2.  $Q \leftarrow \{p_s\}$
3.  $R = \varnothing$
4.  while $Q \neq \varnothing$ do
5.      $p \leftarrow$ node with minimum $f$ value in $Q$
6.      $Q \leftarrow Q \setminus \{p\}$
7.      $R \leftarrow R \cup \{p\}$
8.      if $p = p_{Goal}$ then
9.          return reconstructed path to $p_{Goal}$ via predecessor list of $p$
10.     else
11.         for all $\delta$ do
12.             $p' \leftarrow$ succeeding state of $p$ using $\mu(\theta, \delta)$
13.             if $m(p'_{\hat{x}}) =$ obstacle then
14.                 $R \leftarrow R \cup \{p'\}$
15.             else if $\exists p \in Q: p_{\hat{x}} = p'_{\hat{x}}$ then
16.                 $g' = g(p) + l(p, p')$
17.                 if $g' < g$ value of existing node in $Q$ then
18.                     replace existing node in $Q$ with $p'$
19.                 end if
20.             else
21.                 $Q \leftarrow Q \cup \{p'\}$
```

```
22.                 end if
23.             end for
24.         end if
25. end while
26. return no path found
```

The hybrid A* search algorithm is well-informed due to the heuristic use and thus has fast convergence to the solution. The resulting path is feasible, and it lies in the neighborhood of the global optimum solution [69]. A similar algorithm called a bi-directional hybrid A* is used in the soft snake robot path planning [24]. In the bi-directional method, the search tree grows from both start and goal positions. When the two search trees become close, their branches are connected to form a complete path.

6.4 Path Planning Results

The path planning with a hybrid A* planner is done in the MATLAB Navigation Toolbox. The OSCAR range of motion is limited by the minimum turn radius constraint

$$|R| \geq R_{\min} \tag{6.5}$$

where $R_{\min} = 467.7$ mm from the robot's achievable workspace in Fig. 4.4, b. The length of each motion primitive is equal to the total robot displacement in a single locomotion cycle, which is $ds = 30$ mm, as chosen in Chapter 5. Both $R_{\min}$ and ds are inputs to the planner tool. Finally, the set of motion primitives, defined by (6.5) and ds, is discretized into $n = 19$ motion primitives to satisfy a requirement of being an odd number in order to smooth the planned path.

In the current study, only circular obstacles are considered, as shown in Fig.6.2(a). They are specified by the coordinates of their center (*x*, *y*) and a radius *r*. For collision avoidance purposes, additional tunable clearance space of size *df* is added around obstacles; in this work we specify $df = 65$ mm. This distance accommodates half of the robot width plus an additional 11 % of the robot width to accommodate estimated motion uncertainty. Thus, the obstacle has a total radius $(r + df)$ in the workspace occupancy map, as shown in Fig.6.2(b). The workspace discretization in the occupancy map is 1 mm.

Two path planning scenarios have been investigated and presented to illustrate the OSCAR autonomous navigation framework. One of the scenario schematics is shown in Fig. 6.3. OSCAR starts with a zero-degree orientation with respect to the positive x-axis at its initial configuration, and it must plan an S-shaped path to reach a goal configuration while avoiding static obstacles in the workspace. In the goal configuration, OSCAR should have a zero-degree orientation as well. The second scenario is a mirrored version of the first one about the x-axis. These scenarios are chosen as they allow demonstration of OSCAR's ability to complete complex maneuvers.

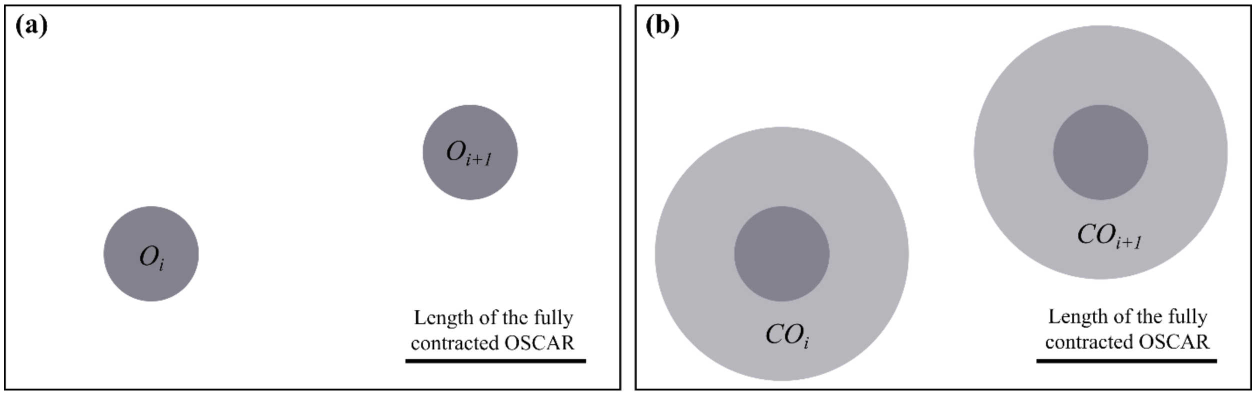

(a)
O_{i+1}
O_i
Length of the fully contracted OSCAR
(b)
CO_{i+1}
CO_i
Length of the fully contracted OSCAR

Figure 6.2 Obstacles: (a) real size in a workspace and (b) in an occupancy map. In (b), obstacles have added clearance around them for collision avoidance purposes during the planning

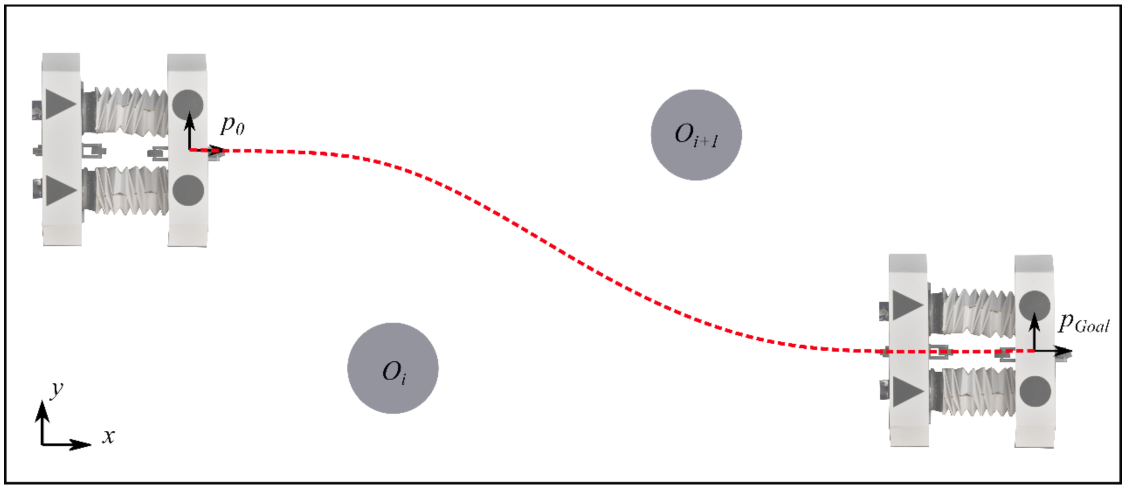


Figure 6.3 Autonomous navigation scenario schematics

6.4.1 Scenario 1

The OSCAR initial and goal configurations are $p_0 = [10 \quad 200 \quad 0]^T$ and $p_{Goal} = [1200 \quad 5 \quad 0]^T$, respectively. Units are mm and radians. The obstacle center locations and radii are specified in Table 6.1. The resulting planned path $P_{ref,\exp 1}$ is shown in Fig. 6.4. As can be seen, due to the robot's achievable workspace limitations, the planned path is close to the second obstacle. However, accommodated clearance space will allow safe collision avoidance, as will be shown in the experimental results.

Table 6.1 Obstacles in scenario 1

	x (mm)	y (mm)	R (mm)
Obstacle 1	150	70	20
Obstacle 2	550	150	20

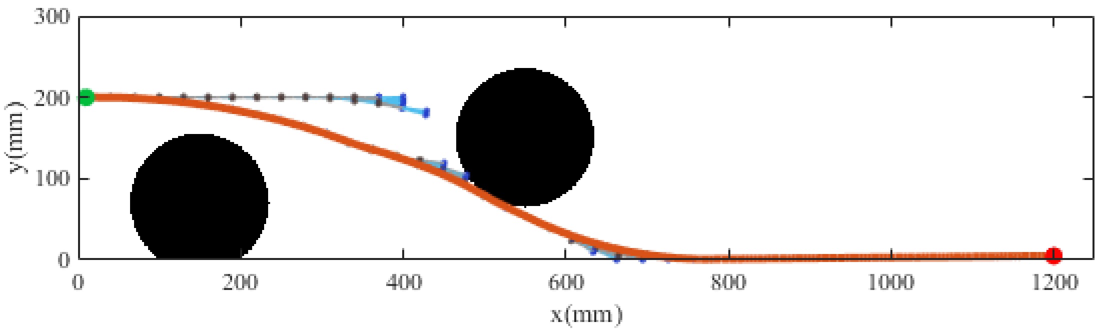


Figure 6.4 Path planned for the first scenario from the start (green) to goal (red) configurations. Black shaded circles are the obstacles with added clearance space, and light blue lines are nodes expanded by the planner

6.4.2 Scenario 2

The OSCAR initial and goal configurations are $p_0 = [10 \quad 5 \quad 0]^T$ and $p_{Goal} = [1200 \quad 200 \quad 0]^T$, respectively. The obstacles are specified in Table 6.2. In this experiment, a third obstacle is added to obtain a straight path for $x > 800$ mm, similar to Fig. 6.4. The resulting planned path $P_{ref,\exp 2}$ is shown in Fig. 6.5.

Table 6.2 Obstacles in scenario 2

	x (mm)	y (mm)	R (mm)
Obstacle 1	150	110	20
Obstacle 2	550	50	20
Obstacle 3	800	285	20

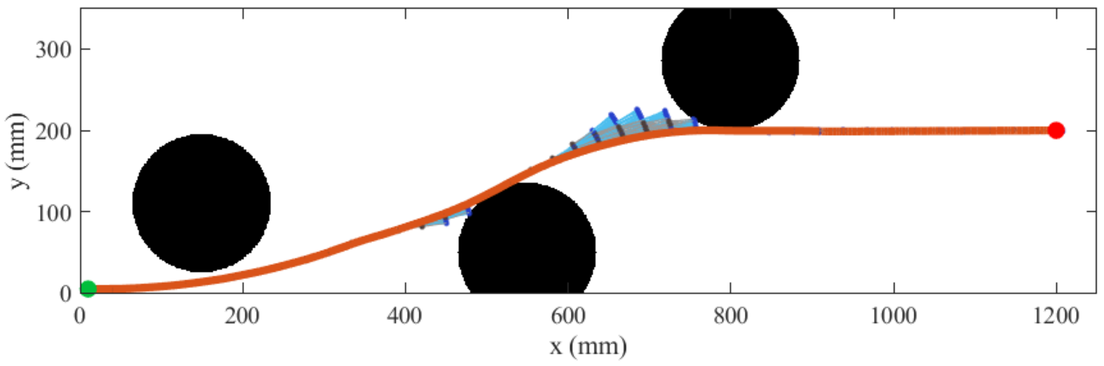


Figure 6.5 Path planned for the second scenario from the start (green) to goal (red) configurations. Black shaded circles are the obstacles with added clearance space, and light blue lines are nodes expanded by the planner

6.5 OSCAR Autonomous Navigation Experiments

In the OSCAR autonomous navigation framework, path planning is done in advance, as shown in Section 6.4. The planned path is then tracked by the feedback path following controller, designed in Chapter 5. The results of the OSCAR's autonomous navigation are presented below. The obstacles are not present in the experiments, but the clearance space accommodates a safe distance to the obstacles.

As part of the future work, OSCAR's autonomous navigation framework could be extended to include navigation in the presence of dynamic obstacles. These scenarios would correspond to the real-world environment, where the obstacles are not static. In this case, the path planning has to be modified. If the obstacles and their trajectories are known, the free configuration space can be modified to accommodate these trajectories. Then, the path planning using the above method could be performed [63]. Instead, if the obstacle motion is unknown, the initial path should be planned first and local replanning is required for collision avoidance as the robot moves to the goal [53].

6.5.1 Scenario 1

Due to the experimental setup's workspace limitation, the planned path $P_{ref,\exp1}$ has been divided into two parts, as shown in Table 6.3. Two-part experiments have been conducted and merged, similar to Chapter 5. During part 1 experiments, the reference path has been extended to include the remainder of $P_{ref,\exp1}$ as the virtual path. The virtual path is used as a reference by the path following controller to calculate control inputs when the actual path ends, but the robot has not reached a point $p_{ref,1,g}$. Similarly, during part 2 experiments, the reference path has a virtual path: a straight line of length 400mm with the orientation of p_{Goal}.

Table 6.3 Reference paths for two parts of scenario 1

	Start Configuration	Goal Configuration
Part 1	$p_0 = \begin{bmatrix}10 & 200 & 0\end{bmatrix}^T$	$p_{ref,1,g} = \begin{bmatrix}636.8 & 29.21 & -0.2353\end{bmatrix}^T$
Part 2	$p_{ref,1,s} = \begin{bmatrix}636.8 & 29.21 & -0.2353\end{bmatrix}^T$	$p_{Goal} = \begin{bmatrix}1200 & 5 & 0\end{bmatrix}^T$

The autonomous navigation results are shown in Fig. 6.6 to Fig. 6.9. Here, the simulation and experimental results for part 1 are shown first separately in Fig. 6.6 and Fig. 6.7, and then together in Fig. 6.8 to demonstrate the similarities and draw conclusions. As can be seen, the robot avoids obstacles and follows the reference path accurately in both simulation and experiment, see Fig. 6.6 and Fig. 6.7, respectively. In Fig. 6.7, the robot body (grey shaded area), corresponding to front plate motion, is plotted to demonstrate obstacle avoidance. The obstacles in their actual size are shown in pink color.

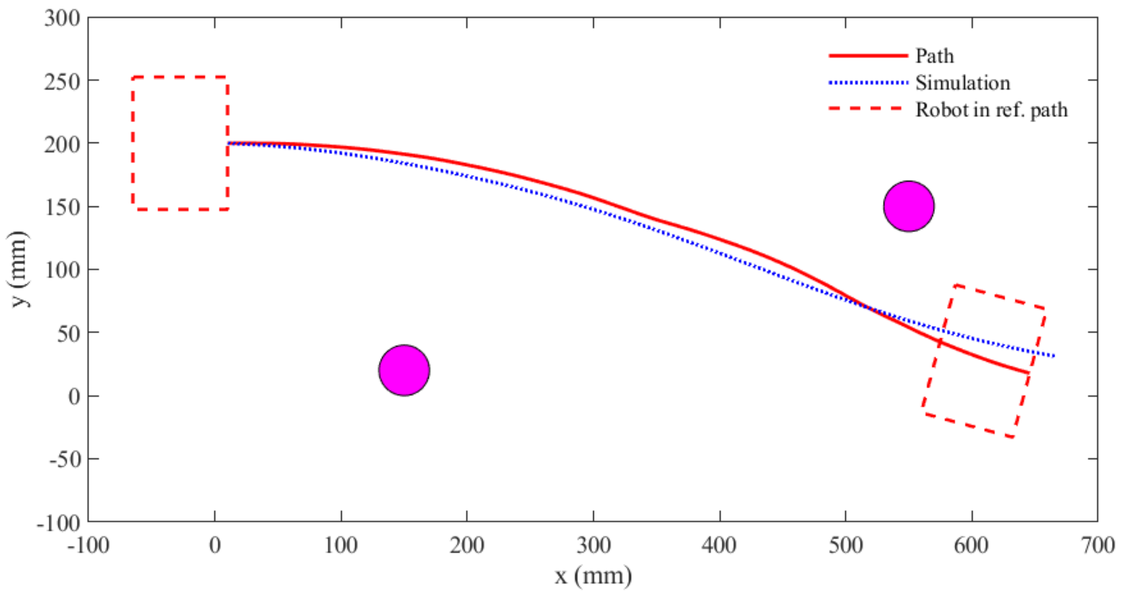


Figure 6.6 OSCAR trajectory in simulation in part 1 of scenario 1. The red dashed lines show the robot's initial and final configurations in the reference path; a fully contracted robot state is shown. Obstacles in actual size (without clearance space) are shown in pink.

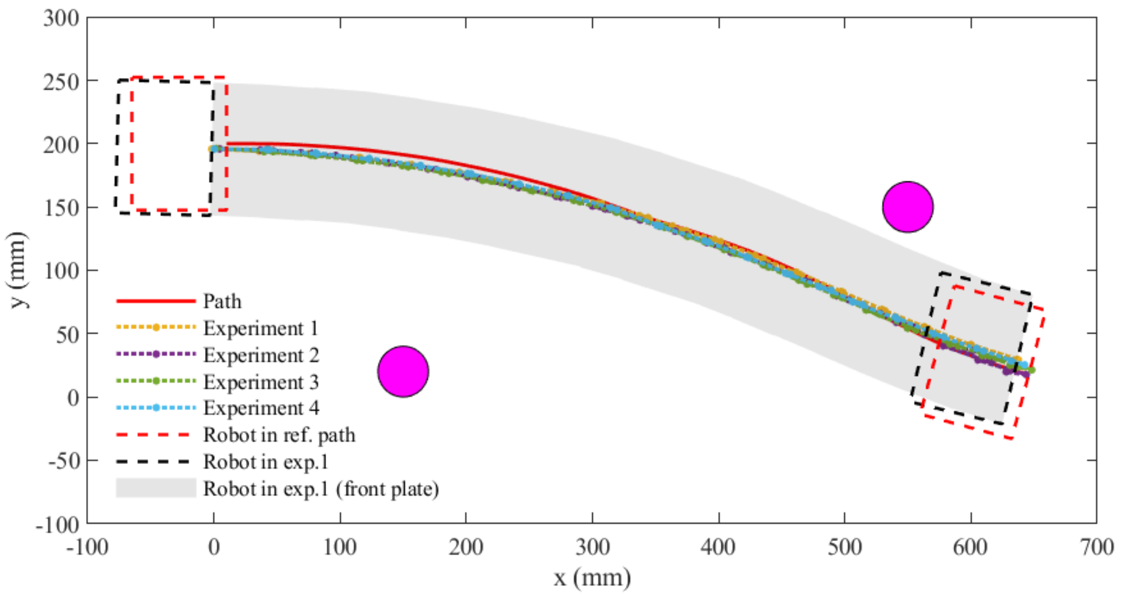


Figure 6.7 OSCAR trajectory in experiments in part 1 of scenario 1. The red and black dashed lines show the robot's configurations in the reference path and experiment 1. The robot body motion (grey shaded area) in experiment 1 demonstrates obstacle avoidance. Four trials show the repeatability of the results

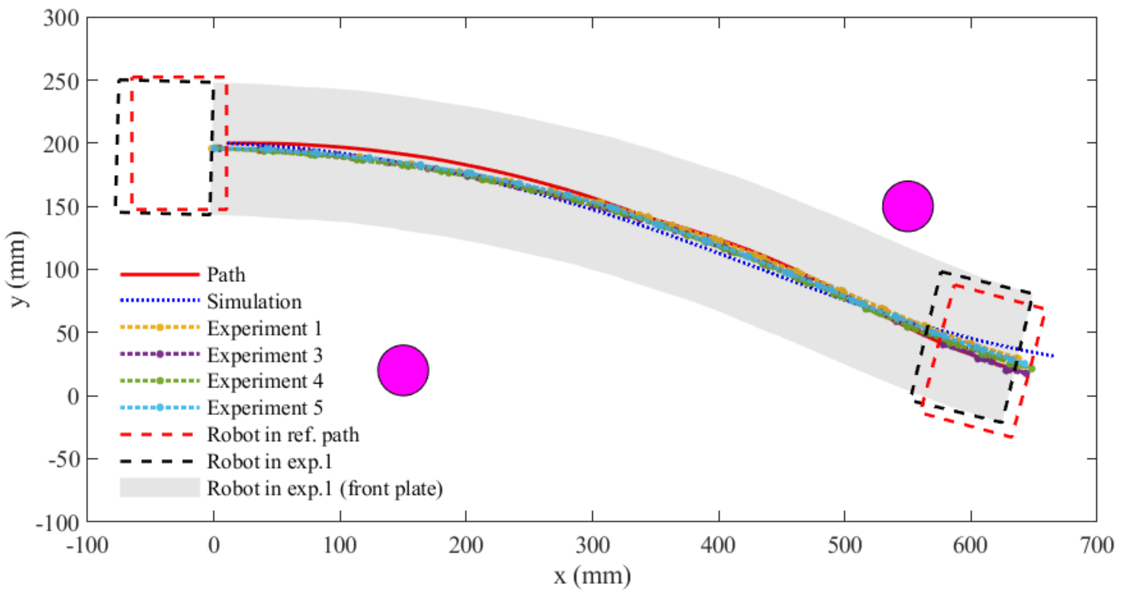


Figure 6.8 Combined simulation and experimental results for part 1 of scenario 1

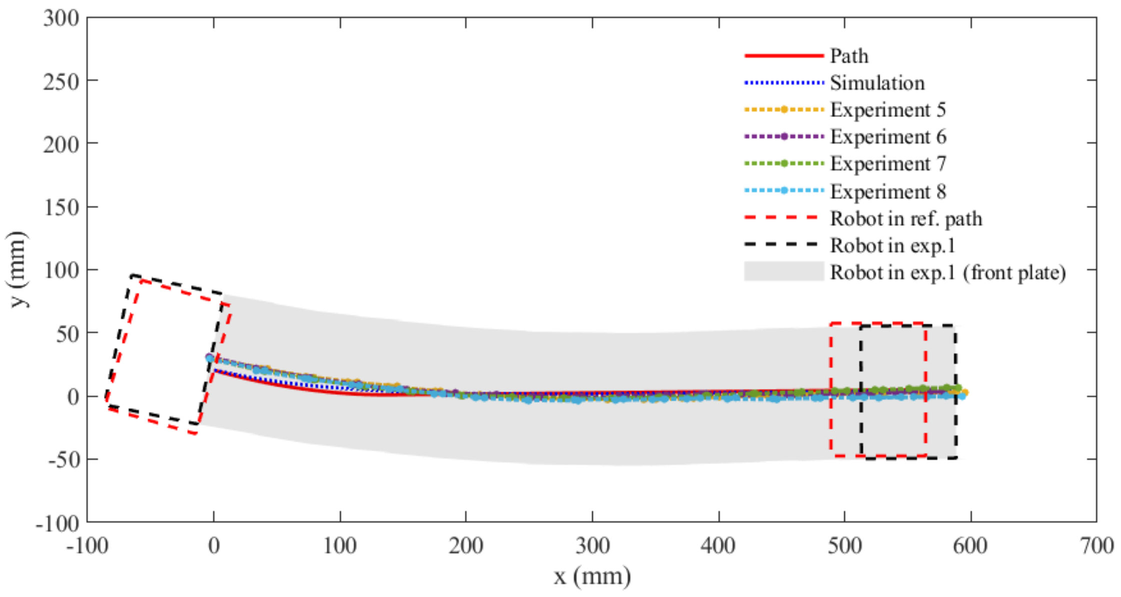


Figure 6.9 Simulation and experimental results for OSCAR trajectory in part 2 of the scenario 1. Here, red and black dashed lines show the robot configuration in reference path and experiment 5, respectively. The robot body motion (grey shaded area) in experiment 5 demonstrates obstacle avoidance. Four trials show the repeatability of the results

Additionally, the robot configurations in the reference path (red dashed line) and experiment 1 (black dashed line) are shown at the initial and final states. These demonstrate the similarities between robot position and orientation in the reference path and experiment. The combined experimental and simulation results are shown in Fig. 6.8. As can be seen, the OSCAR trajectory matches the simulation. The same is shown in Fig. 6.9 for part 2 of scenario 1. Four trials have been conducted in each case to show the results' repeatability.

In order to merge the results of part 1 and part 2, the final state at experiment 1 in part 1 is used as an initial state for all part 2 experiments. It should be noted that based on the final state position in experiment 1, the reference path for part 2 has been chosen in Table 6.3. The merged results of experiment 1 of part 1 and experiment 5 of part 2 are shown in Fig. 6.10. These results demonstrate a complete OSCAR trajectory for scenario 1.

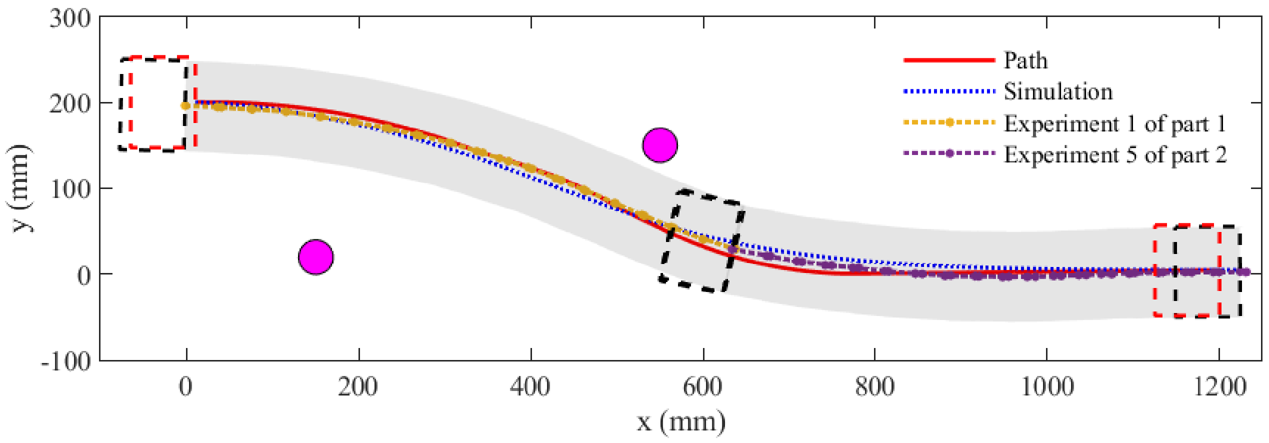


Figure 6.10 Coupled results of OSCAR trajectory in scenario 1. The final state in experiment 1 matches the initial state of experiment 5

The comparison of angular inputs for simulation and experiments 1 and 5 are shown in Fig. 6.11 and Fig. 6.12, respectively. As can be seen, the experimental angular inputs align well with the simulation. However, as expected, the experimental inputs are slightly larger than in simulation, as the robot needs to compensate for some unknown uncertainties.

In more detail, in Fig. 6.11, to follow a curved path in scenario 1, the robot initially makes right turns ($\varphi_1 > \varphi_2$) until $k = 13$ locomotion cycle in the experiment and $k = 15$ in simulation, then it makes left turns $(\varphi_2 > \varphi_1)$. As the robot converges to the straight-line path in scenario 1, the difference between angular inputs decreases, as shown in Fig. 6.12.

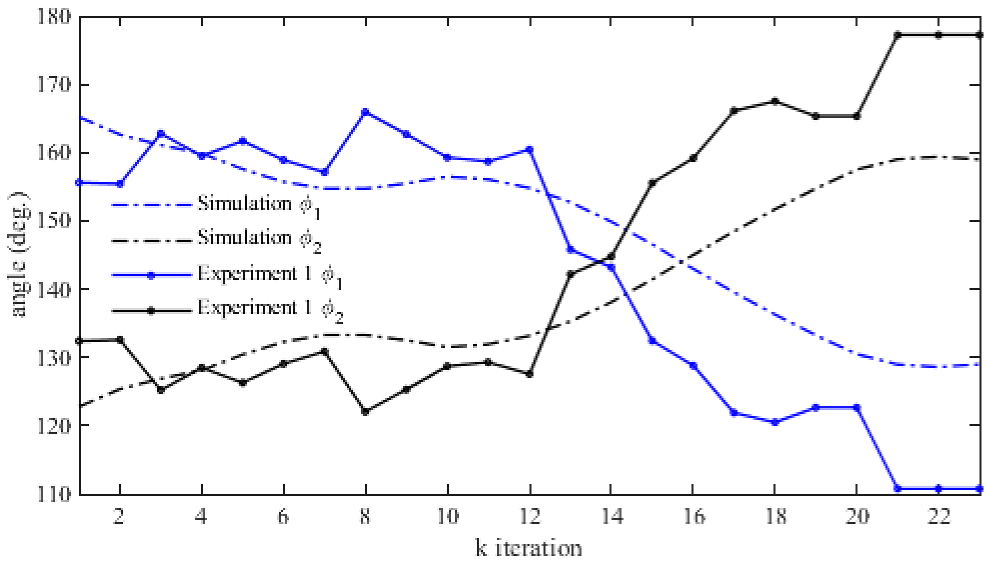


Figure 6.11 Angular inputs for part 1 in scenario 1

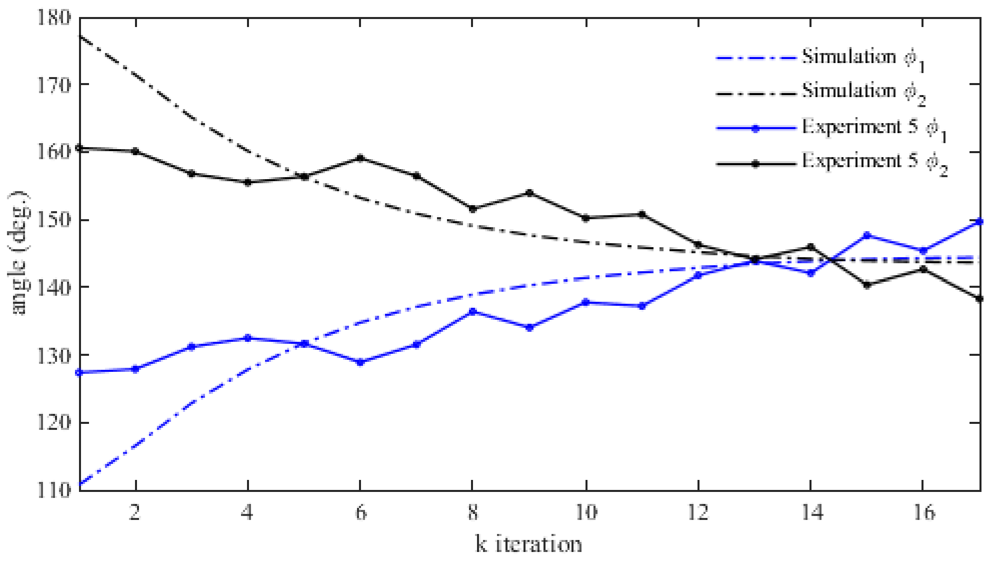


Figure 6.12 Angular inputs for part 2 in scenario 1

6.5.2 Scenario 2

As in Scenario 1, the planned path $P_{ref,\exp 2}$ has been divided, as shown in Table 6.4. Here, the initial state of the reference path for part 2 is chosen based on the final state in experiments of part 1, as described in previous section. It should be noted that the reference path has been shifted by *y*=200 mm to make the robot ‘centered’ to the experimental setup’s camera.

Table 6.4 Reference paths for two parts of scenario 2

	Start Configuration	Goal Configuration
Part 1	$p_0 = \begin{bmatrix}10 & -195 & 0\end{bmatrix}^T$	$p_{ref,2,g} = \begin{bmatrix}642 & -33.89 & 0.2587\end{bmatrix}^T$
Part 2	$p_{ref,2,s} = \begin{bmatrix}642 & -33.89 & 0.2587\end{bmatrix}^T$	$p_{Goal} = \begin{bmatrix}1200 & 0 & 0\end{bmatrix}^T$

The OSCAR's autonomous navigation results for two parts are shown in Fig. 6.13 and Fig. 6.14. The final state in experiment 9 in Fig. 6.13 is used as the initial state for all part 2 experiments in Fig. 6.14. The virtual paths are taken similar to Scenario 1. As can be seen, the robot (grey shaded area) safely avoids the obstacles and follows the reference path in both simulation and experiment. Four trials demonstrate the results' repeatability in each part. The coupled results of parts 1 and 2 are shown in Fig. 6.15.

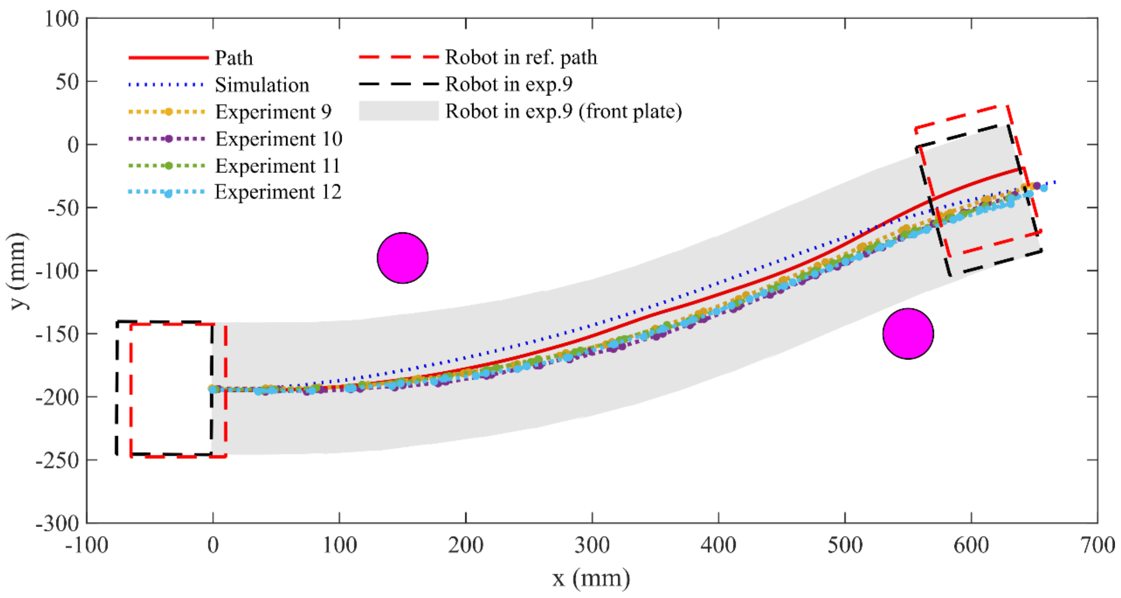


Figure 6.13 OSCAR trajectory in part 1 of the scenario 2. Here, red and black dashed lines show the robot configuration in reference path and experiment 9. The robot body motion (grey shaded area) in experiment 9 demonstrates the obstacle avoidance. Four trials show the results repeatability

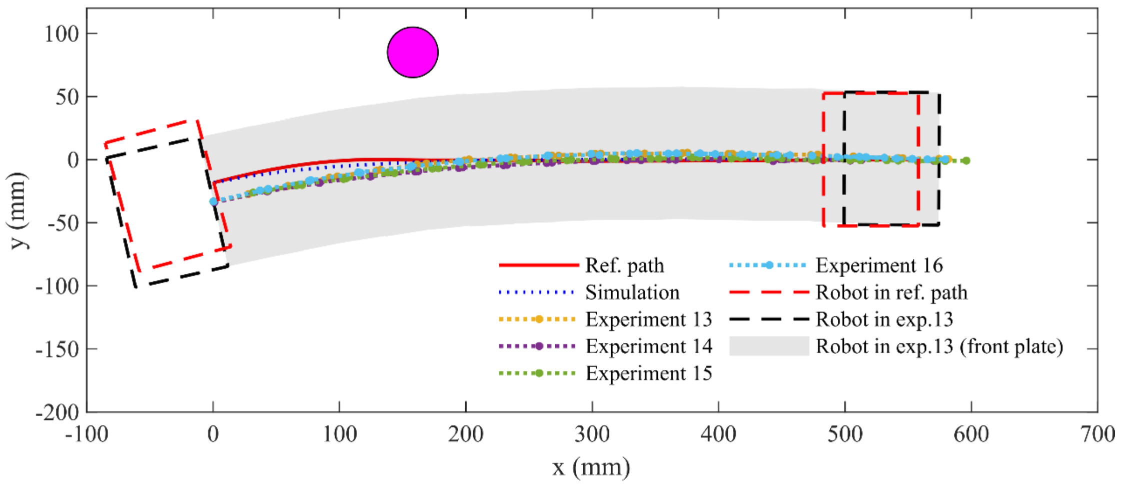


Figure 6.14 OSCAR trajectory in part 2 of the scenario 2. Here, red and black dashed lines show the robot configuration in reference path and experiment 13. The robot body motion (grey shaded area) in experiment 13 demonstrates the obstacle avoidance. Four trials show the results repeatability

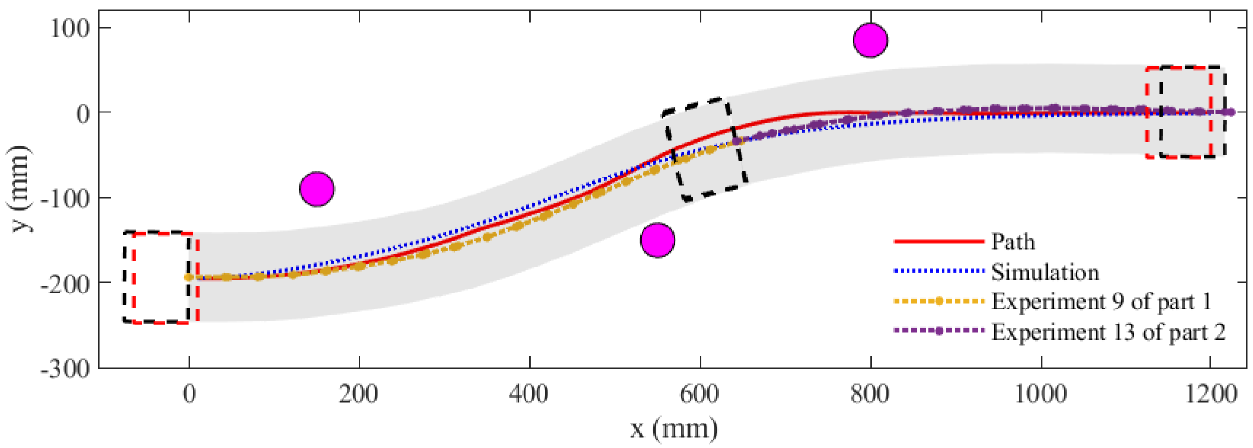


Figure 6.15 Coupled results of OSCAR trajectory in scenario 2. The final state in experiment 9 of part 1 matches the initial state of experiment 13 of part 2

The comparison of the angular inputs for simulation and experiments 9 and 13 are shown in Fig. 6.16 and Fig. 6.17, respectively. Simular conclusions as in Scenario 1 could be drawn.

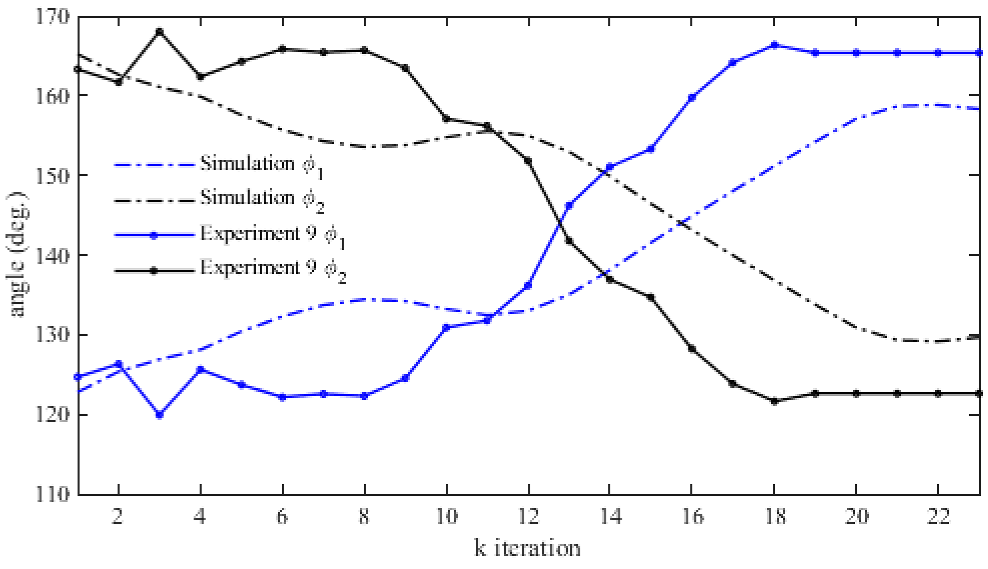


Figure 6.16 Angular inputs for part 1 in scenario 2

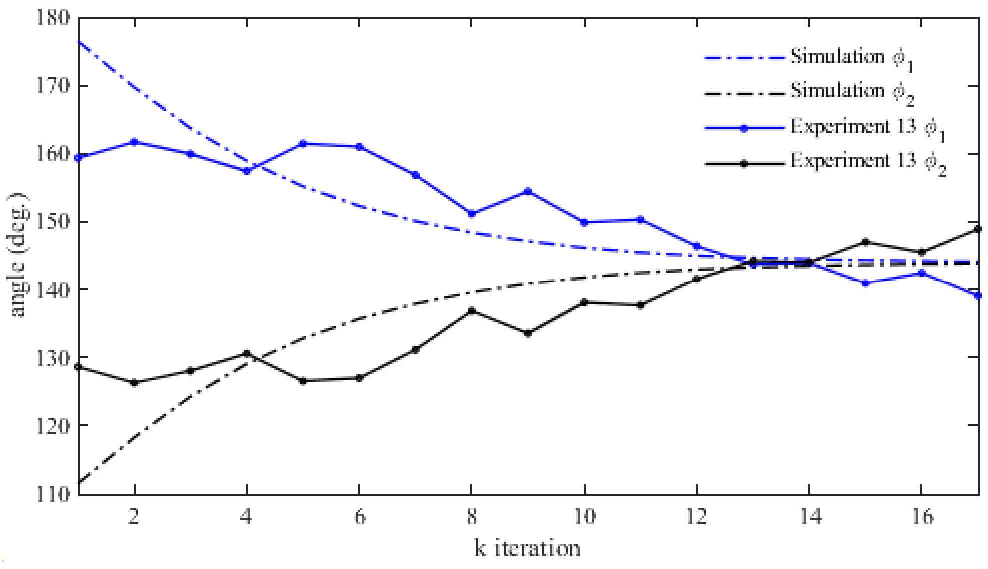


Figure 6.17 Angular inputs for part 2 in scenario 2

6.6 Chapter Summary

This chapter concludes the autonomous navigation framework for OSCAR and demonstrates its ability to autonomously navigate complex paths in the 2D terrain with static obstacles. It is one of the main contributions of this work, showing the soft mobile robots' ability to accurately follow the complex reference paths.

The presented framework uses a hybrid A* algorithm for path planning and the proportional feedback controller for path following. The perception is accomplished externally in the experimental setup. The hybrid A* planner accommodates the robot motion constraints, and the feedback controller allows to accurately follow the path due to the presented robot design accurately. Additionally, clearance space around obstacles allows safe collision avoidance in the planned path. Future work may include autonomous navigation in the presence of dynamic obstacles.

Chapter 7 expands the presented framework to include multi-segment OSCAR coupled locomotion, as it would expand and augment OSCAR functionality.

Chapter 7
Coupled Locomotion Strategy

7.1 Motivation

The modular self-reconfigurable robots have attracted researchers in the robotics field for their high versatility and robustness [76],[77]. These robots consist of multiple segments (modules) of the same or different functionality, e.g., [78]–[80]. With each segment being independent, the modular robot can reconfigure itself to adapt its shape for a task, e.g., it can make a chain structure for crawling and rolling and a lattice structure for walking. Additionally, the modularity allows the robot to self-repair. If one of the segments is faulty, the robot can disconnect it and reconfigure itself to include only the functional segments to continue the mission. Hence, these robots could be viable and efficient in such applications as search-and-rescue or unknown area exploration.

Similarly, multiple OSCARs can be arranged into a metameric robot to utilize the modular robots' benefits. Metameric means multiple similar segments arranged in series. In the metameric robot, an individual OSCAR is referred to as a segment. Many of the existing soft metameric robots, such as the origami-ball earthworm-like robot in [6], have rigidly attached segments that cannot be decoupled. Each OSCAR segment can move independently, or they can reconfigure and assemble into a metameric robot. This would allow greater task adaptability in certain real applications, such as search-and-rescue or area exploration, compared to the existing soft robots.

This chapter describes and validates the coupled locomotion strategy of the two-segment robot as the basic building block to metameric locomotion with the OSCAR concept. It could readily be extended to include a larger number of segments in the future. Section 7.2 describes the locomotion strategy for the coupled OSCAR segments. The passive docking mechanism used to connect the segments is described in Section 7.2.2. Section 7.3 demonstrates the effectiveness of the locomotion strategy.

7.2 Proposed Coupled Locomotion Strategy

7.2.1 Coupled Locomotion Strategy

The proposed coupled locomotion strategy is inspired by the earthworm locomotion. This locomotion has been widely used in soft mobile robots [22]; examples include an origami-ball earthworm-like robot in [6], meshworm [13], and softworm [81]. An earthworm is a true metameric animal. Separated by septa, its segments can be actuated independently. Each segment has two antagonistic groups of muscles: longitudinal and circular muscles [82]. When the longitudinal muscles contract, the segment shortens, and its diameter increases. Instead, when the circular muscles contract, the segment elongates, and its diameter decreases. Each segment has the bristle-like setae that perform an anisotropic friction function, and these setae were the inspiration for OSCAR’s foot designs. When the segment contracts, the setae anchor it to the ground providing high friction. When the segment elongates forward, they slide and provide low friction.

The earthworm locomotes by coordinated segments’ expansion and contraction, called the retrograde peristalsis wave [82]. The wave travels from the head to tail segment along the body, resulting in the forward motion. It is schematically shown in Fig. 7.1(a) for the three-segment earthworm moving to the right. The figure is adapted from [23]. As can be seen, when the head segment 1 elongates, the rear segment 3 contracts and anchors to the ground with setae. The middle segment 2 remains at rest. Then, the head segment 1 contracts, and adjacent to it segment 2 elongates simultaneously. This wave of contraction and elongation of two adjacent segments, called a peristalsis wave, travels back to the tail segment 3, facilitating a total forward displacement Δx. Then, the cycle repeats.

Similarly, the coupled two-segment OSCAR locomotion is schematically presented in Fig. 7.1(b). Unlike the actual earthworm, OSCAR does not actuate in the radial direction.

Instead, high ground friction is provided by the anisotropic friction feet. Therefore, the segment height in OSCAR's schematics does not change. In Fig. 7.1, (b), to show a coordinated transition of low and high friction in the feet, low and high feet friction are shown by black shaded circles and triangles, respectively.

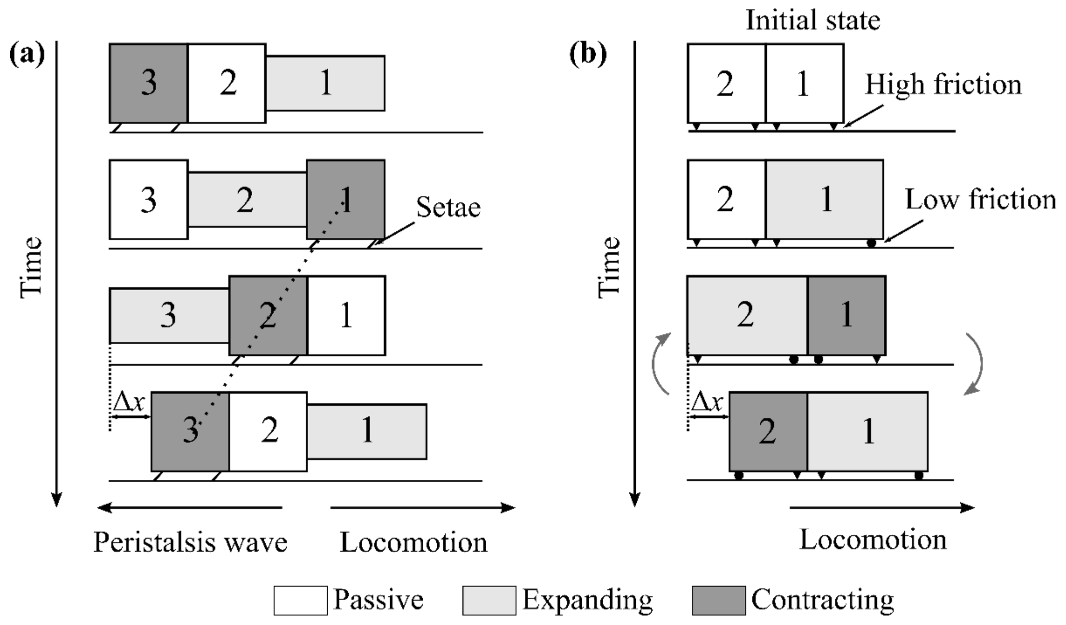


Figure 7.1 (a) Schematics of the earthworm-like locomotion, adapted from [23]**; (b) coupled two-segment OSCAR locomotion strategy**

The locomotion can be described as follows. Starting from the initial state, where both segments are at rest, the head segment 1 expands. Due to the generated forces from the origami towers, the front plate moves forward due to its feet having low friction, while the back plate stays fixed due to its feet having high friction. Then, segment 1 contracts, and segment 2 expands simultaneously. During that motion, the front plate of segment 1 and the back plate of segment 2 stay fixed due to the feet's high friction. Both connected plates in the middle have low friction and, thus, they move forward. Following that, segment 2 contracts, while segment 1 expands. During this motion, the two connected plates in the middle have high friction at their feet and, thus, anchor to the ground. It allows the front plate of segment 1 and the back plate of segment 2 to move forward. The last two steps then iterate in a cycle, each time resulting in the displacement Δx.

Let denote the fully contracted segment state as 0, and the fully expanded state as 1. Then, transition $0 \to 1$ corresponds to expansion and $1 \to 0$ corresponds to contraction. The coupled locomotion can then be described as the following sequence $(0,0) \to (0,1) \to (1,0) \to (0,1) \to \ldots$, where the states $(1,0)$ and $(0,1)$ repeat in a cycle.

The locomotion strategy is experimentally validated in Section 7.3.

7.2.2 Docking Mechanism

The passive docking mechanism for the two segments' connection is shown in Fig. 7.2. To realize it, the front and back plates at the connection are modified from those presented in Fig. 2.3. The front plate with connection mechanism has six strong permanent neodymium magnets evenly spaced on two disks (Fig. 7.2 (a)), and the back-plate has a detachable cover with the same number of magnets (Fig. 7.2 (b)) oriented in the opposite north-south magnet pole orientation from the front plate. When two robots are close to each other, they can passively dock due to the magnetic attraction. The resulting pulling force between connected plates is 11.2 N.

This docking mechanism could be extended to include a segmentation functionality, which could be realized by the actuation of the front plate's disks. Disks are placed on bearings for low friction. Since the back plate is static at the connection, the front plate's disks could rotate and detach two connected segments. Such segmentation is energy-efficient, as the magnet sheer force is significantly lower than the pulling force. The disks rotate in the opposite directions for stability during the possible segmentation. The proposed docking and segmentation mechanism has been realized previously in other robotic applications. It was done for a single tower robot in [46], where an active plate had a single disk and shape memory alloy (SMA) actuators were used for segmentation. A similar magnetic docking and segmentation mechanism has been implemented in [5]. Finally, in [78], magnetic docking and SMA-based segmentation by translation have been demonstrated. These prior efforts add to the confidence in the chosen design for docking and segmentation used here.

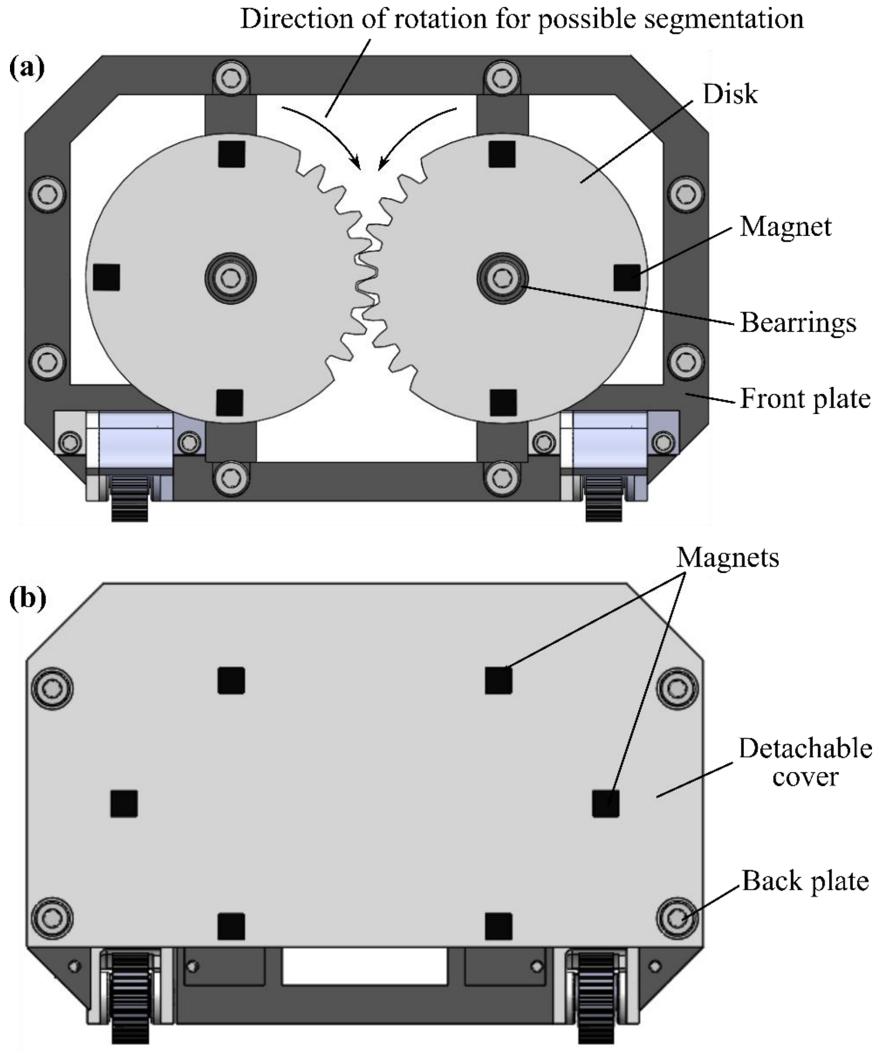


Figure 7.2 Docking mechanism: (a) front plate of segment 2 and (b) back plate of segment 1

7.3 Coupled Locomotion Strategy Validation

The two-segment OSCAR robot is shown in Fig. 7.3. It has the markers on the front plate of segment 1 and on the back plate of segment 2 to characterize its total displacement. Each segment has an offboard microcontroller. The low-level servo position controller from Chapter 3 controls segment expansion and contraction. For the coupled 1D locomotion, the selected angular expansion inputs have been set at $[\varphi_1 \ \varphi_2]^T = [170 \ 170]^T$ degrees, and the angular contraction inputs have been set at $[\varphi_1 \ \varphi_2]^T = [0 \ \ 0]^T$ degrees.

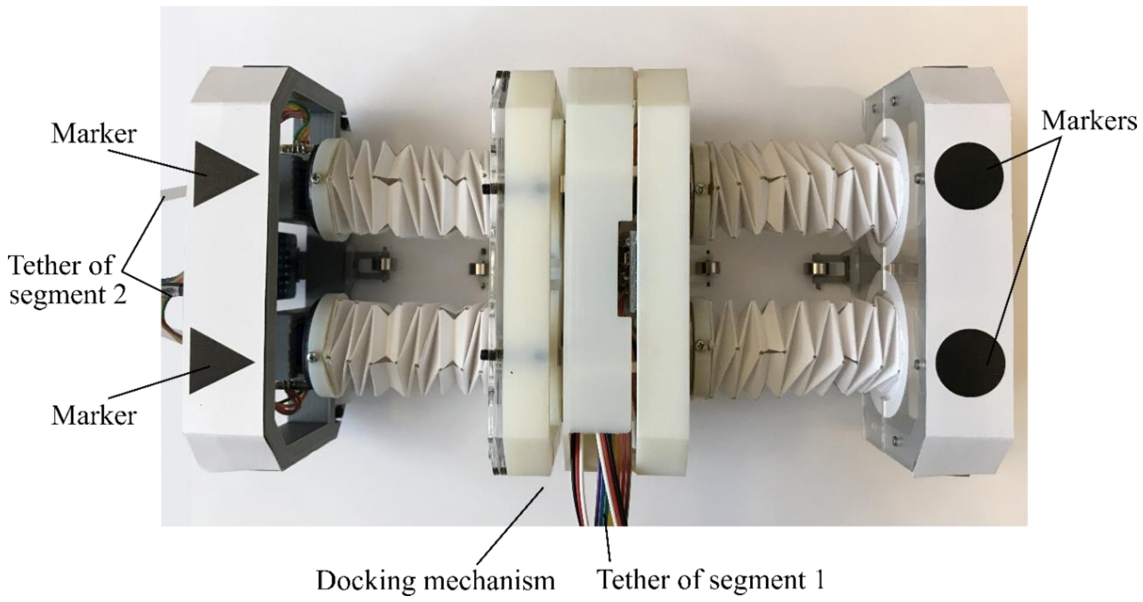


Figure 7.3 Two-segment robot at the fully contracted state (top view)

The coupled locomotion data has been collected in the experimental setup previously described. However, the LabVIEW VI sent the angular inputs to each of the separate OSCAR microcontrollers through separate tethered connections. An additional GoPro camera has been used to record the video at 30 frames per second. It was located next to the testbed's primary camera and had a 'god's eye' view of the workspace.

The resulting coupled locomotion is shown as a sequence of video frames in Fig. 7.4. Each frame shows the segments at either fully expanded and fully contracted states. Starting from the fully contracted initial state (Fig. 7.3), the expansion inputs are sent to segment 1, i.e., $(0,0)\to(0,1)$. This is illustrated in Fig. 7.4 (top). Following that, segment 2 expands, and segment 1 contracts, i.e., $(0,1)\to(1,0)$, as shown in Fig. 7.4 (middle). Then, segment 2 contracts, and segment 1 expands, i.e., $(1,0)\to(0,1)$, as shown in Fig. 7.4 (bottom). The last two steps repeat in a cycle while the robot locomotes.

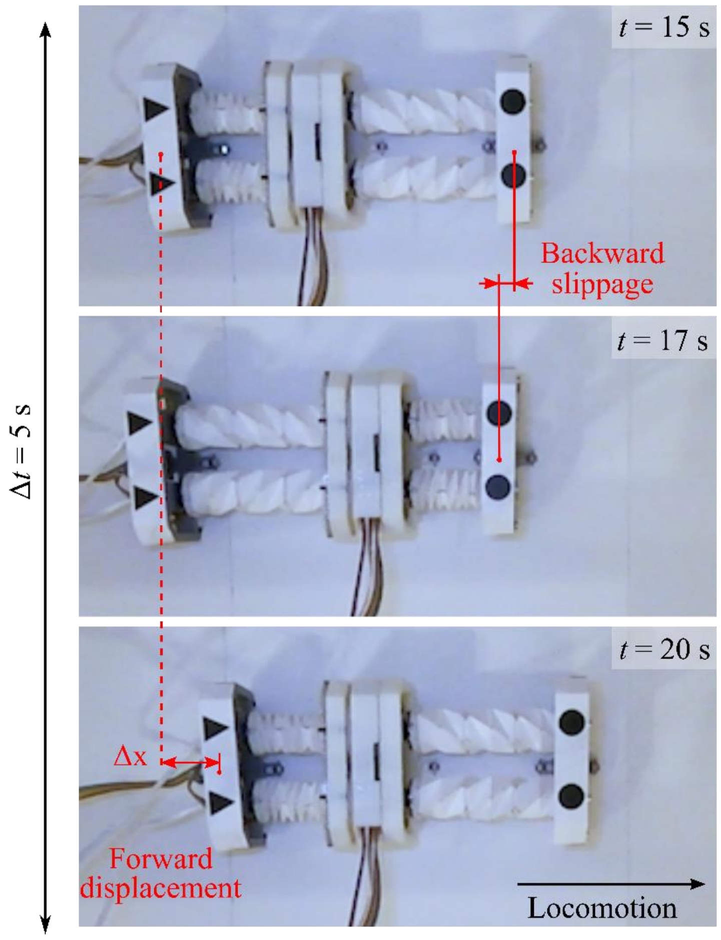


Figure 7.4 Sequence of video frames depicting the coupled locomotion

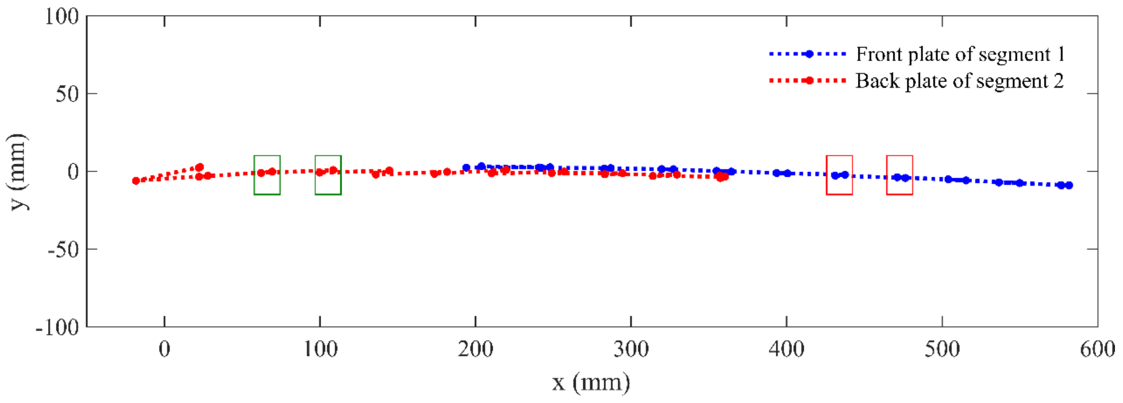


Figure 7.5 Coupled robot trajectory. Here, the highlighted areas show examples of backward slippage of front plate during contraction (red boxes) and backward slippage of back plate during expansion (green boxes)

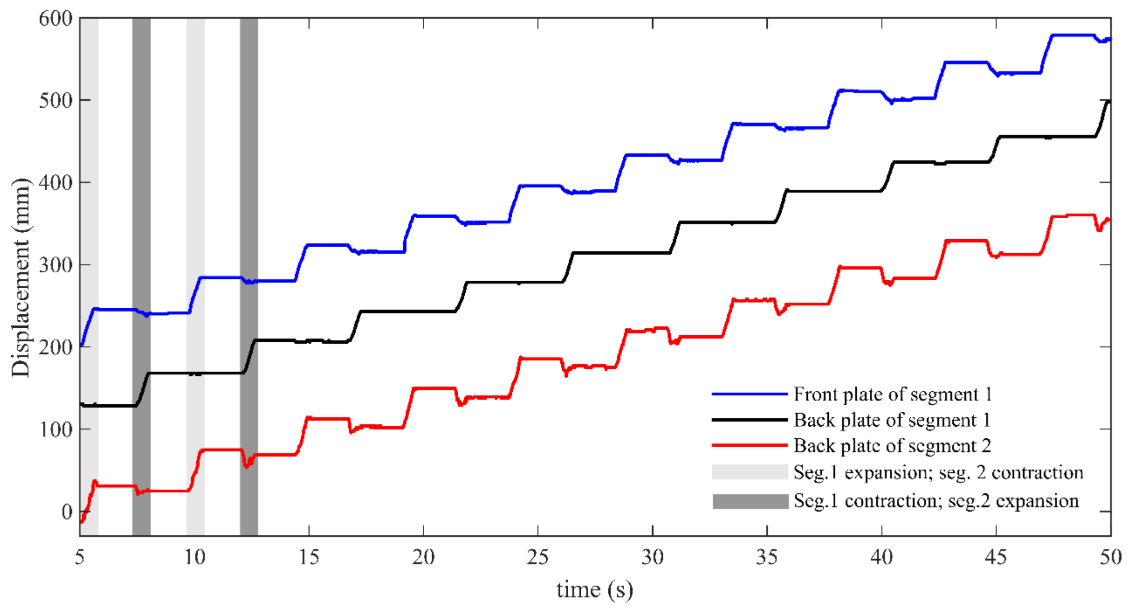


Figure 7.6 Displacement time history. Highlighted areas shown for two cycles only correspond to the actuation times during two alternating states: (a) segment 1 - expansion and segment 2 - contraction; (b) segment 1 - contraction and segment 2 – expansion

Figure 7.5 shows the resulting robot trajectory, which corresponds to the centroids of the front plate of segment 1 and the back plate of segment 2. As can be seen, although not constrained in the y-direction, the robot moves straight. The total maximum deviation in the y-direction for the segment 1 front plate is 12 mm, and for the segment 2 back plate it is 7 mm during a total 400 mm displacement along the x-axis.

The displacement time history is shown in Fig. 7.6, starting from $t = 5s$. It is obtained by processing recorded video frames in MATLAB. The time history plot shows the centroids of the front and back plates of segments 1 and 2, respectively, as in Fig. 7.5. Also, it presents the displacement of the back plate of segment 1. The highlighted areas correspond to actuation time periods during each locomotion state: (a) the expansion of segment 1 and contraction of segment 2, and (b) the contraction of segment 1 and the expansion of segment 2. The flat areas correspond to wait times for the next state. As can be seen, during (a), both front and back plates of segments 1 and 2 move forward, while the connected plates in the middle remain. As explained in subsection 7.2.1, due to segment 1 expansion, its front plate moves forward caused by low feet friction, and the back plate remains fixed due to high feet friction. For similar reasons, the back plate of segment 2 moves forward, and its front plate remains fixed during contraction. Then, during (b), the connected plates in the middle move forward, caused by both contraction of segment 1 and the expansion of segment 2. The front and back plates of segments 1 and 2 are supposed to remain fixed. However, as can be seen, they have some backward slippage, as also shown in Fig. 7.4 and Fig. 7.5. The amount of backward slippage is comparable to a single uncoupled OCSAR.

7.4 Chapter Summary

The coupled multi-segment locomotion presented in this chapter is a valuable study for the OSCAR autonomous navigation framework. With the proposed coupled locomotion strategy, OSCAR is able to navigate both autonomously and in a metameric configuration. Thus, it would potentially increase OSCAR's flexibility and usefulness compared to existing soft mobile metameric robots in real applications. As stated earlier, the existing soft metameric robots in the literature, as the writing of this thesis, cannot be decoupled. As demonstrated here, the two-segment robot can effectively locomote with the proposed coupled locomotion strategy. Moreover, the coupled robot can be expanded to include more segments. Similarly, two segments should be actuated simultaneously, and the resulting wave of expansion and contraction should travel from the head to tail segment.

Chapter 8
Conclusions and Future Work

8.1 Summary of Research Contributions

In the current state of the art in soft robotics, autonomous navigation is a goal that has not been demonstrated well or often. Unlike rigid-bodied robots, soft robots suffer from motion uncertainties caused by their compliance and interaction with the environment. Therefore, the implementation of autonomous navigation for soft robots is considerably more challenging than for their rigid-bodied counterparts. This research develops, implements, and demonstrates an autonomous navigation approach for the novel origami-enabled soft crawling robot OSCAR.

The overall research approach can be divided into five main parts: (i) control-oriented iterative robot design; (ii) kinematic model development; (iii) path following controller design; (iv) path planning; and (v) OSCAR's multi-segment locomotion. The first four parts develop and experimentally validate the autonomous navigation for a single OSCAR. The fifth part extends OSCAR's functionality by introducing a modular approach to combining multiple OSCARs. This lays the foundation for more complex soft robotic efforts in the future.

Chapters 2 and 3 cover the iterative robot design, which was a significant contribution to this thesis, as it allowed for effective OSCAR autonomous navigation. As described in Chapter 2, initial OSCAR designs suffered from significant motion uncertainties caused by its foot-ground interaction, the low-level servo control, and the assembly process induced misalignment. The presented iterative design approach mitigated these uncertainties. It can be summarized here in three main efforts:

4. *Iterative foot design (Chapter 2)* allowed for a reliable and robust switching between a high and low friction ground interaction. After several iterations, the final design utilized a sliding ratchet foot to maximize effectiveness. These feet minimized undesirable backward slippage due to enhanced traction and improved OSCAR's locomotion capabilities both in straight line motion and turning.
5. *The low-level closed-loop servo position control (Chapter 3)* significantly reduced the motion uncertainties caused by the non-uniform expansion and contraction of two origami towers. The initial low-level controllers were simple proportional-integral (PI) controllers. They control the angular position of the servos, thereby actuating the towers to expand and contract. When implemented on the tower actuation, the initial controllers did not achieve repeatable tower response which led to significant challenges in OSCAR control. To allow uniform and repeatable actuation of the origami towers, the feedback controllers for the servos implemented ramp reference positional inputs instead of the original step reference. This can be thought of as rate-limiting any reference positions that would come to the servos. Additionally, the controller's identified the existence of significant servo dead-band that had to be compensated in the controller to achieve adequate functionality.
6. *OSCAR assembly process (Chapter 2)* addresses uncertainties due to the non-uniform foot-ground interaction caused by misalignment of OSCAR's front and back plates. This is, effectively, a manufacturing issue which arose because each OSCAR is a hand-built custom robot and there was significant variation. The refined assembly process, with the custom-designed and custom-built assembly guide, mitigated these uncertainties to provide a robot with consistent performance necessary for path following control.

The resulting OSCAR has a robust and repeatable performance validated by the symmetric motion for the left and right turns with the same angular inputs, shown in Chapter 3. It enabled an application of high-level path following control. All the experiments in this work are conducted in the experimental setup presented in Chapter 3. The setup's primary purpose is robot localization. It is done with the developed marker localization algorithm that was detailed in the chapter.

Chapter 4 presents the two-part OSCAR kinematic model, which includes the lumped kinematic submodel (LKS) and the segmented kinematic submodel (SKS). These two models work together to relate origami tower actuation to robot motion in the planar workspace. The LKS is a simplified model that converts the given desired radius of turn and displacement into OSCAR position increments. The SKS is a detailed model that considers the origami cell geometry. It converts the position increments from the LKS into the angular inputs required to achieve the desired displacement. This kinematic model is ideal and does not account for the actual robot losses. To accommodate these losses, this chapter introduces the empirically based correction. The correction aligns model predictions with the experimental data. The resulting corrected model has been used for the simulation in Chapters 5 and 6.

Chapter 5 presents the path following control. Its purpose is to calculate the OSCAR angular inputs to follow a reference path. Due to the foot-ground interaction uncertainties, the path following is crucial for autonomous navigation in soft robots. This chapter presents two controllers: a model-based pure pursuit and a feedback controller. Adapted from the well-known pure pursuit method, a model-based pure pursuit is an open-loop controller. It uses the kinematic model to calculate angular inputs. The proportional feedback controller is based on the measured lateral error to the path. This chapter validates both controllers in simulation and experiment. The chosen case study is a straight path with a robot having an initial offset from the path. As shown experimentally, the pure pursuit has a poor performance in experiments due to OSCAR’s uncertainties in the terrain interactions. The feedback controller, due to direct output measurement, demonstrates a very accurate path following in comparison with the open-loop controller.

Chapter 6 presents the path planning approach and demonstrates complete autonomous navigation for a single OSCAR. The path planning is done by the hybrid A* planner. Unlike other path planners, hybrid A* accounts for OSCAR motion constraints and outputs a feasible path. In this framework, the hybrid A* planner plans the path offline. Then, OSCAR uses a feedback path-following controller to follow it. The autonomous navigation has been validated for two case scenarios with static obstacle avoidance in both simulation and experiment.

Finally, Chapter 7 presents the two-segment OSCAR coupled locomotion. The locomotion strategy is bio-inspired by the behavior of an earthworm. It is done by simultaneous expansion of one segment and the other segment's contraction that are then alternated. As shown experimentally, this gait allows effective locomotion of the coupled robot. The chapter also presents a passive docking mechanism for the segment connection. The modular approach allows OSCAR to navigate both separately and in a coupled configuration. Thus, the OSCAR has higher adaptability for future practical applications, where the OSCAR needs to reconfigure its shape.

In conclusion, this framework presents effective autonomous navigation for OSCAR, a novel origami-enabled soft crawling autonomous robot. Thus, this work narrows the gap between soft robots and their practical applications. The final OSCAR has a robust and reliable performance under the conditions used in this research. It can accurately navigate a 2D space while avoiding static obstacles. It can follow a complex path and it can converge to that path if it is offset from the path to start with.

8.2 Future Work

This research is an initial work validating the OSCAR concept for autonomous navigation. As such, it is acknowledged to be a significant step in what could be many future investigations for this class of robots. One of the shortcomings of the current robot, due to complexity management concerns is that OSCAR was tethered and had the sensing information coming from offboard. This was all done to minimize the complexity since it was challenging enough to achieve reliable locomotion in the physical OSCAR. Also, this work considers cases with static obstacle avoidance only. This research could be built upon and extended to achieve the untethered OSCAR, navigating a 2D or 3D terrain with multiple moving obstacles. Therefore, future work could be organized as follows:

1. From the design perspective, untethered locomotion should be added by embedding the power sources and microcontroller on board. Additionally, actuation for segmentation should be added to fully utilize the modularity capabilities. With added segmentation, coupled segments could be separated on demand. Moreover, the structural components, such as origami towers, could be fabricated from different material to increase robot durability for possible

applications in harsh conditions. Finally, the additional onboard sensors, e.g., vision, could be added to enable truly autonomous navigation without the need for the external components of the current experimental setup. Since the robot is origami-based, it could be easily scaled up or down as needed; although this may necessitate a redesign of the material used in the towers and the actuation mechanisms.

2. From the autonomy perspective, navigation and collision avoidance in the presence of moving obstacles could be investigated to advance OSCAR's capabilities. In this case, path planning should be done online to allow active re-planning for obstacle avoidance. Moreover, autonomous navigation could be studied for multiple robots simultaneously. Additionally, OSCAR could be used as a soft robotic research platform to investigate the efficiency of other path-following and path planning methods in soft mobile robots. These studies could be done both in simulation and experiment, using either the robot kinematic model or the actual OSCAR. Finally, navigation through 3D terrain could be conducted experimentally to investigate OSCAR effectiveness in managing terrain changes.
3. From the modular robotics perspective, complex scenarios, where both decoupled and coupled locomotion are present, could be investigated. In this initial attempt, these actions are shown in separate experiments. Additionally, only a coupled straight locomotion is demonstrated this thesis. Having a high-level controller for OSCAR modularity combined with path planning and following could be investigated in future work. This could include 2D trajectories where the robot segments disengage and engage to navigate around obstacles. Modularity could be also extended to include larger number of coupled segments, which would allow to add additional locomotion gaits to the robot. With sufficient segments, multi-module OSCARs could even surround objects and perform manipulation tasks as well as locomotion tasks. This is a very rich and open area for research in multi-module soft robotics.

4. Finally, due to its cost-efficiency, OSCAR could be adapted to be used as a research or educational platform for origami-enabled soft mobile robots. For example, this would be an outstanding way to perform STEM outreach to K-12 students because of the multiple different aspects of engineering that would be involved.

References

[1] C. Laschi, B. Mazzolai, and M. Cianchetti, "Soft robotics: Technologies and systems pushing the boundaries of robot abilities," *Sci. Robot.*, 2016.

[2] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, 2015.

[3] C. Lee *et al.*, "Soft robot review," *Int. J. Control. Auton. Syst.*, 2017.

[4] M. T. Tolley *et al.*, "A resilient, untethered soft robot," *Soft Robot.*, 2014.

[5] J. Shintake, V. Cacucciolo, D. Floreano, and H. Shea, "Soft robotic grippers," *Adv. Mater.*, 2018.

[6] H. Fang, Y. Zhang, and K. W. Wang, "Origami-based earthworm-like locomotion robots," *Bioinspir. Biomim.*, 2017.

[7] S. I. Rich, R. J. Wood, and C. Majidi, "Untethered soft robotics," *Nat. Electron.*, 2018.

[8] D. Rus and M. T. Tolley, "Design, fabrication and control of origami robots," *Nat. Rev. Mater.*, 2018.

[9] C. Laschi, M. Cianchetti, B. Mazzolai, L. Margheri, M. Follador, and P. Dario, "Soft robot arm inspired by the octopus," *Adv. Robot.*, 2012.

[10] R. F. Shepherd *et al.*, "Multigait soft robot," *Proc. Natl. Acad. Sci.*, 2011.

[11] T. Umedachi, V. Vikas, and B. A. Trimmer, "Softworms : the design and control of non-pneumatic , 3D-printed , deformable robots," *Bioinspir. Biomim.*, 2016.

[12] H. T. Lin, G. G. Leisk, and B. Trimmer, "GoQBot: A caterpillar-inspired soft-bodied rolling robot," *Bioinspiration and Biomimetics*, 2011.

[13] S. Seok, C. D. Onal, K.-J. Cho, R. J. Wood, D. Rus, and S. Kim, "Meshworm: A peristaltic soft robot with antagonistic nickel titanium coil actuators," *IEEE/ASME Trans. Mechatronics*, 2013.

[14] C. D. Onal and D. Rus, "Autonomous undulatory serpentine locomotion utilizing body dynamics of a fluidic soft robot," *Bioinspiration and Biomimetics*, 2013.

[15] M. Luo *et al.*, "Slithering towards autonomy: A self-contained soft robotic snake platform with integrated curvature sensing," *Bioinspiration and Biomimetics*, 2015.

[16] N. W. Bartlett *et al.*, "A 3D-printed, functionally graded soft robot powered by combustion," *Science (80-.).*, 2015.

[17] H.-Y. Chen *et al.*, "RUBIC: An untethered soft robot with discrete path following," *Front. Robot. AI*, 2019.

[18] E. W. Hawkes, L. H. Blumenschein, J. D. Greer, and A. M. Okamura, "A soft robot that navigates its environment through growth," *Sci. Robot.*, 2017.

[19] K. Suzumori, S. Endo, T. Kanda, N. Kato, and H. Suzuki, “A bending pneumatic rubber actuator realizing soft-bodied manta swimming robot,” in *IEEE International Conference on Robotics and Automation*, 2007.

[20] M. Sfakiotakis, A. Kazakidi, N. Pateromichelakis, and D. P. Tsakiris, “Octopus-inspired eight-arm robotic swimming by sculling movements,” in *IEEE International Conference on Robotics and Automation*, 2013.

[21] A. D. Marchese, C. D. Onal, and D. Rus, “Autonomous soft robotic fish capable of escape,” *Soft Robot.*, 2014.

[22] S. Kim, C. Laschi, and B. Trimmer, “Soft robotics: A bioinspired evolution in robotics,” *Trends Biotechnol.*, 2013.

[23] M. Calisti, G. Picardi, and C. Laschi, “Fundamentals of soft robot locomotion,” *J. R. Soc. Interface*, 2017.

[24] M. Luo, “Pressure-operated soft robotic snake modeling, control, and motion planning,” Worcester Polytechnic Institute, 2017.

[25] S. Miyashita, S. Guitron, M. Ludersdorfer, C. R. Sung, and D. Rus, “An untethered miniature origami robot that self-folds, walks, swims, and degrades,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015.

[26] S. M. Felton, M. T. Tolley, C. D. Onal, D. Rus, and R. J. Wood, “Robot self-assembly by folding: A printed inchworm robot,” *IEEE Int. Conf. Robot. Autom.*, 2013.

[27] M. Luo *et al.*, “OriSnake: Design, fabrication, and experimental analysis of a 3-D origami snake robot,” *IEEE Robot. Autom. Lett.*, 2018.

[28] D. Jeong and K. Lee, “Design and analysis of an origami-based three-finger manipulator,” *Robotica*, 2018.

[29] T. Liu, Y. Wang, and K. Lee, “Three-dimensional printable origami twisted tower: design, fabrication, and robot embodiment,” *IEEE Robot. Autom. Lett.*, 2018.

[30] J. Santoso and C. D. Onal, “An origami continuum robot capable of precise motion through torsionally stiff body and smooth inverse kinematics,” *Soft Robot.*, 2020.

[31] S. Miyashita, S. Guitron, K. Yoshida, S. Li, D. D. Damian, and D. Rus, “Ingestible, controllable, and degradable origami robot for patching stomach wounds,” in *2016 IEEE International Conference on Robotics and Automation*, May 2016.

[32] Z. Zhakypov, K. Mori, K. Hosoda, and J. Paik, “Designing minimal and scalable insect-inspired multi-locomotion millirobots,” *Nature*, 2019.

[33] H. Fang, Y. Zhang, and K. W. Wang, “An earthworm-like robot using origami-ball structures,” 2017.

[34] A. Rafsanjani, Y. Zhang, B. Liu, S. M. Rubinstein, and K. Bertoldi, “Kirigami skins make a simple soft actuator crawl,” *Sci. Robot.*, 2018.

[35] B. Liu, Y. Ozkan-Aydin, D. I. Goldman, and F. L. Hammond, “Kirigami skin improves soft earthworm robot anchoring and locomotion under cohesive soil,” in *2019 IEEE International Conference on Soft Robotics (RoboSoft)*, 2019.

[36] P. Polygerinos *et al.*, “Soft robotics: review of fluid-driven intrinsically soft devices; manufacturing, sensing, control, and applications in human-robot interaction,” *Adv. Eng. Mater.*, 2017.

[37] M. W. Spong, S. Hutchinson, and M. Vidyasagar, “Robot modeling and control,” *IEEE Control Systems*. 2006.

[38] D.A. Bristow ; M. Tharayil ; A.G Alleyne., “A survey of iterative learning,” *IEEE Control Syst. Mag.*, 2006.

[39] J. D. Greer, L. H. Blumenschein, A. M. Okamura, and E. W. Hawkes, “Obstacle-aided navigation of a soft growing robot,” in *2018 IEEE International Conference on Robotics and Automation*, 2018.

[40] A. Pagano, T. Yan, B. Chien, A. Wissa, and S. Tawfick, “A crawling robot driven by multi-stable origami,” *Smart Mater. Struct.*, 2017.

[41] O. Angatkina, K. Gustafson, A. Wissa, and A. G. Alleyne, “Path following for the origami crawling robot,” in *Proc. ASME Dyn. Syst. Control Conf.*, 2019.

[42] K. Gustafson, O. Angatkina, and A. Wissa, “Model-based design of a multistable origami-enabled crawling robot,” *Smart Mater. Struct.*, 2020.

[43] B. Kresling, “Origami-structures in nature: lessons in designing ‘smart’ materials,” *MRS Proc.*, 2012.

[44] C. B. K. Zimmermann, I. Zeidis, *Mechanics of terrestrial locomotion*. Springer, 2009.

[45] A. Pagano, B. Leung, B. Chien, T. Yan, A. Wissa, and S. Tawfick, “Multi-Stable Origami Structure for Crawling Locomotion,” in *ASME 2016 Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, 2016.

[46] O. Angatkina *et al.*, “A metameric crawling robot enabled by origami and smart materials,” in *ASME 2017 Smart Materials, Adaptive Structures and Intelligent Systems*, Sep. 2017.

[47] E. Olson, “AprilTag: A robust and flexible visual fiducial system,” 2011.

[48] S. M. Abbas, S. Aslam, K. Berns, and A. Muhammad, “Analysis and improvements in apriltag based state estimation,” *Sensors*, 2019.

[49] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems Eighth Edition*, 8th ed. Pearson, 2019.

[50] K. J. Gustafson, “Study of the compliance and system integration of bistable structures for use as actuation mechanisms in bioinspired adaptive systems,” University of Illinois at Urbana-Champaign, 2019.

[51] A. P. Aguiar and J. P. Hespanha, “Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty,” *IEEE Trans. Automat. Contr.*, 2007.

[52] N. H. Amer, H. Zamzuri, K. Hudha, and Z. A. Kadir, “Modelling and control strategies in path tracking control for autonomous ground vehicles: a review of state of the art and challenges,” *J. Intell. Robot. Syst.*, 2017.

[53] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *IEEE Trans. Intell. Veh.*, 2016.

[54] J. M. Snider, “Automatic Steering Methods for Autonomous Automobile Path Tracking,” *Work*, 2009.

[55] R. C. Coulter, “Implementation of the pure pursuit path tracking algorithm,” Pittsburgh, Pensilvania, 1992.

[56] O. Amidi, “Integrated Mobile Robot Control,” 1990. Accessed: Feb. 03, 2019. [Online]. Available:

https://www.ri.cmu.edu/pub_files/pub3/amidi_omead_1990_1/amidi_omead_1990_1.pdf.

[57] S. Thrun *et al.*, “Stanley: The robot that won the DARPA Grand Challenge,” *J. F. Robot.*, 2006.

[58] A. Carvalho, Y. Gao, A. Gray, H. E. Tseng, and F. Borrelli, “Predictive control of an autonomous ground vehicle using an iterative linearization approach,” *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, 2013.

[59] J. Morales, J. L. Martínez, M. A. Martínez, and A. Mandow, “Pure-pursuit reactive path tracking for nonholonomic mobile robots with a 2D laser scanner,” *EURASIP J. Adv. Signal Process.*, 2009.

[60] R. Rajamani, *Vehicle dynamics and control*, 2nd ed. Springer Science & Business Media, 2012.

[61] D. González, J. Pérez, V. Milanés, and F. Nashashibi, “A review of motion planning techniques for automated vehicles,” *IEEE Trans. Intell. Transp. Syst.*, 2016.

[62] S. D. Pendleton *et al.*, “Perception, planning, control and coordination for autonomous vehicles,” *Machines*, 2017.

[63] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[64] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Springer Science & Business Media, 2012.

[65] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer (Long. Beach. Calif).*, 1989.

[66] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, 2014.

[67] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, 1959.

[68] P. E. Hart, N. J. Nilsson, and B. Raphael, “Formal basis for the heuristic determination of minimum cost paths,” *IEEE Trans. Syst. Sci. Cybern.*, 1968.

[69] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Practical search techniques in path planning for autonomous driving introduction and related work,” *Ann Arbor*, 2008, Accessed: Apr. 02, 2020. [Online]. Available: www.aaai.org.

[70] A. Stentz, “Optimal and efficient path planning for unknown and dynamic environments,” *Int. J. Robot. Autom.*, 1995.

[71] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, 1996.

[72] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *Int. J. Rob. Res.*, 2011.

[73] J. T. Betts, “Survey of numerical methods for trajectory optimization,” *J. Guid. Control. Dyn.*, 1998.

[74] J. Petereit, T. Emter, C. W. Frey, T. Kopfstedt, and A. Beutel, “Application of hybrid A* to an autonomous mobile robot for path planning in unstructured outdoor environments,” in *ROBOTIK 2012; 7th German Conference on Robotics*, 2012, [Online]. Available: https://pdfs.semanticscholar.org/6e00/16024b257040db590d2de352556f64f46787.pdf.

[75] K. Kurzer, “Path planning in unstructured environments: a real-time hybrid A * implementation for fast and deterministic path generation for the KTH research concept vehicle,” KTH Royal Institute of Techhnoloy, 2016.

[76] M. Yim *et al.*, “Modular self-reconfigurable robot systems [Grand challenges of robotics],” *IEEE*

Robot. Autom. Mag., 2007.

[77] S. S. R. Chennareddy, A. Agrawal, and A. Karuppiah, “Modular self-reconfigurable robotic systems: a survey on hardware architectures,” *J. Robot.*, 2017.

[78] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, “M-TRAN : self-reconfigurable modular robotic system,” *IEEE/ASME Trans. Mechatronics*, 2002.

[79] B. Salemi, M. Moll, and W. M. Shen, “SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system,” in *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[80] J. Davey, N. Kwok, and M. Yim, “Emulating self-reconfigurable robots - design of the SMORES system,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[81] K. a. Daltorio, A. S. Boxerbaum, A. D. Horchler, K. M. Shaw, H. J. Chiel, and R. D. Quinn, “Efficient worm-like locomotion: slip and control of soft-bodied peristaltic robots,” *Bioinspir. Biomim.*, 2013.

[82] C. A. Edwards and J. R. Lofty, *Biology of Earthworms*. Springer, 1972.

Appendix A

LabVIEW VIs for the Robot Operation

This appendix provides an in-detail description of the LabVIEW-based testbed control. It first details the camera calibration process. The camera is used for robot localization. Then, it describes the steps for the image processing algorithm used in robot localization. Finally, it details the main VI for the robot control in the testbed.

A.1 Camera calibration

Camera calibration is required in order to measure the robot's position in the global coordinate frame. Calibration must be updated when the distance between the workspace and the camera or the camera itself is changed. The required LabVIEW toolboxes are NI Vision and NI Vision Development Module.

Step 1. For new calibration, the calibration grid covering the whole workspace area should be placed in the testbed. The standard LabVIEW grid template 20x20 cm can be found in the following directory:

'Box>ARG_Student_Reports>Oyuna Angatkina>Dissertation Supplementary> LabVIEW> Camera Calibration'

The calibration grid resolution is 10x10 mm.

Step 2. To create new calibration open 'Camera control.vi' in the above directory.

This VI uses two Express VIs: *Image Acquisition* and *Vision Assistant*, as shown in Fig. A.1. The *Image Acquisition Express VI* configures camera settings. It is used in all LabVIEW

VIs designed for the robot. The *Vision Assistant Express VI* calibrates camera and configures the robot localization algorithm.

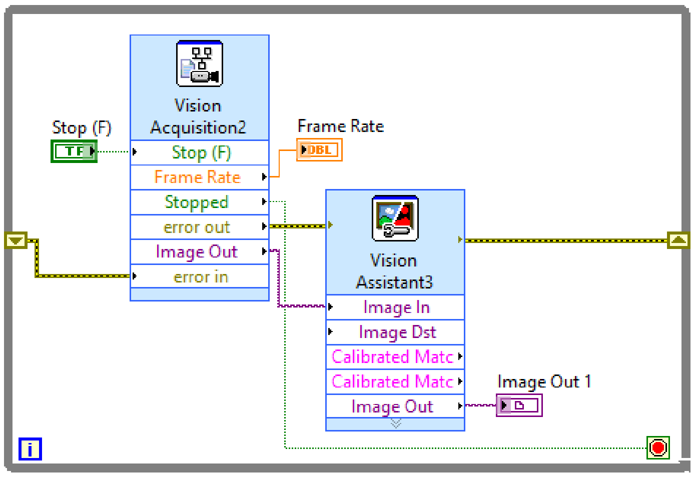


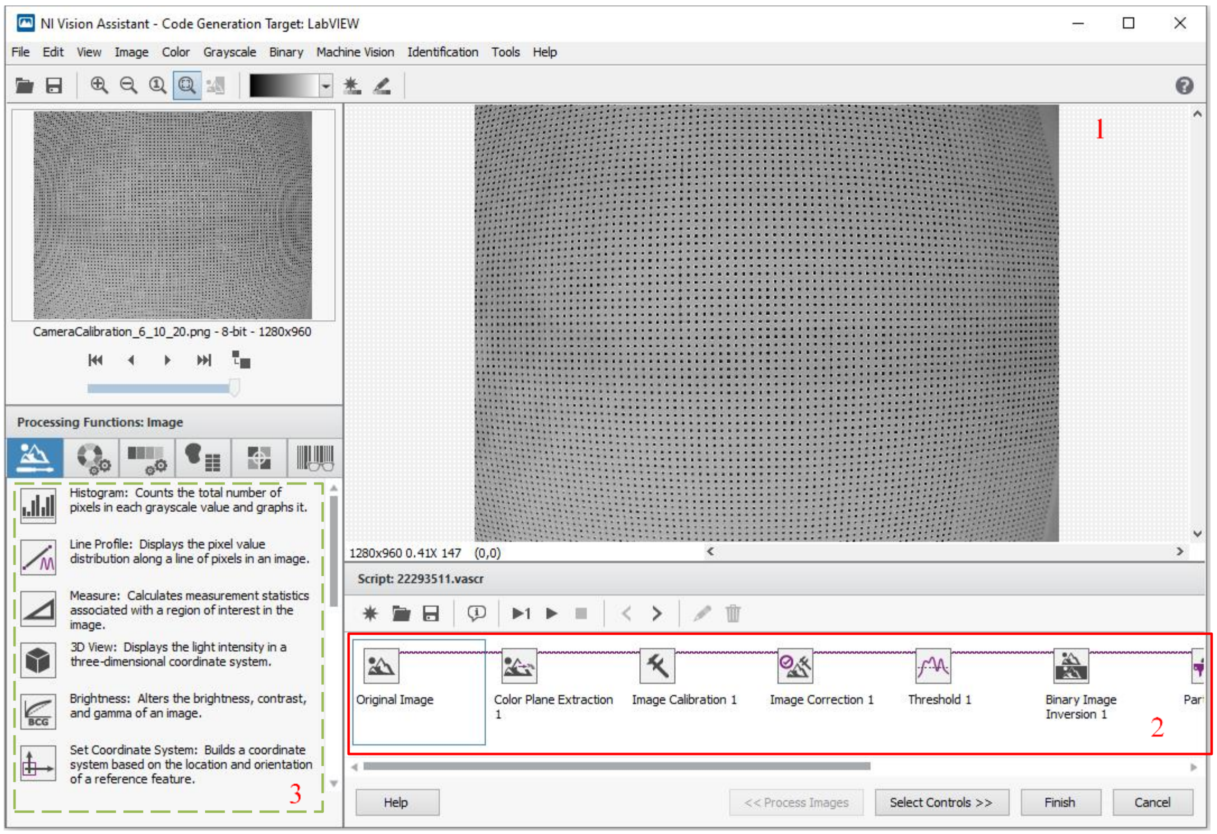
Figure A.1 Block diagram of 'Camera control.vi'

Step 3. Double click and open *Image Acquisition Express VI*. In the configuration box, the following settings are selected to set up the camera:

- Select Acquisition Source: a current camera cam0 : IZONE UVC 5M CAMERA
- Select Acquisition Type: 'Continuous Acquisition with inline processing' with 'Acquire most recent image' in the acquire image type box.
- Configure Acquisition Settings: current image quality is 1280x960 in MJPG format; the speed is 30 frames per second; mode is manual.
- Configure Image Lodging settings:

  By default, 'Enable Image Lodging' is not selected. For the new calibration, enable 'Enable Image Lodging' to save images from the camera ('PNG' format) and select the file path to store them. Run the Camera Control VI to acquire images of the calibration grid.
- Select Controls/Indicators: 'Frame Rate' could be optionally selected in the 'Indicator area' box.

Step 4. Double click and open *Vision Assistant Express VI*. This *Express VI* configures an image processing algorithm for robot localization, and its interface is shown in Fig. A.2.



Histogram: Counts the total number of pixels in each grayscale value and graphs it.

Line Profile: Displays the pixel value distribution along a line of pixels in an image.

Measure: Calculates measurement statistics associated with a region of interest in the image.

3D View: Displays the light intensity in a three-dimensional coordinate system.

Brightness: Alters the brightness, contrast, and gamma of an image.

Set Coordinate System: Builds a coordinate system based on the location and orientation of a reference feature.

Figure A.2 Vision Assistant Express VI window

1. Workspace image; this image will be updated after each step of the algorithm is applied.
2. Image processing algorithm steps.
3. When selected, each step opens its configuration dialog box on the left, shown in 3.

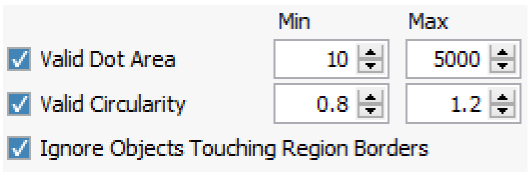
Step 5. Open ‘Image Calibration 1’ and select ‘New Calibration’ to set up new calibration. Otherwise, the path to the existing calibration is selected. It can be edited in the ‘Edit Calibration.’

In the new calibration dialog toolbox, configure the following:

- Select Calibration Type: select ‘Distortion Model (Grid)’ to remove fisheye lens distortion.
- Select Image Source: the image of the acquired calibration grid (see Fig. A.2).

- Extract Grid Features: select look for ‘Dark objects’ for the grid dots filtering and ‘Local Threshold: BG Correction’ as a method.

  The extracted grid dots become highlighted in blue. To tune the detected grid dots, adjust ‘Dot Area’ and ‘Valid Circularity.’



	Min	Max
✓ Valid Dot Area	10	5000
✓ Valid Circularity	0.8	1.2
✓ Ignore Objects Touching Region Borders		

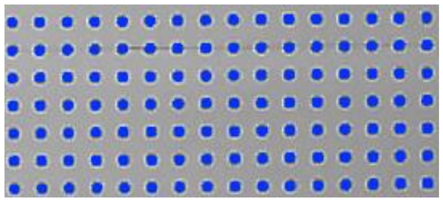
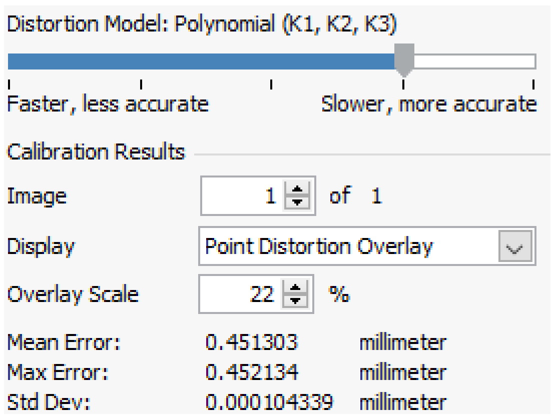


Figure A.3 Settings for calibration grid dots (left) and extracted grid dots (right)

- Specify Grid Parameters: set according to the calibration grid resolution to 10x10 mm.
- Review Calibration Results: adjust the distortion model by moving a slider and looking at the mean error (see Fig. A.4, left). The calibration grid with the applied distortion model, represented by the red vectors at each dot, is shown in Fig. A.4, right.



Distortion Model: Polynomial (K1, K2, K3)

Calibration Results

Image 1 of 1

Display Point Distortion Overlay

Overlay Scale 22 %

```
Mean Error:   0.451303      millimeter
Max Error:    0.452134      millimeter
Std Dev:      0.000104339   millimeter
```

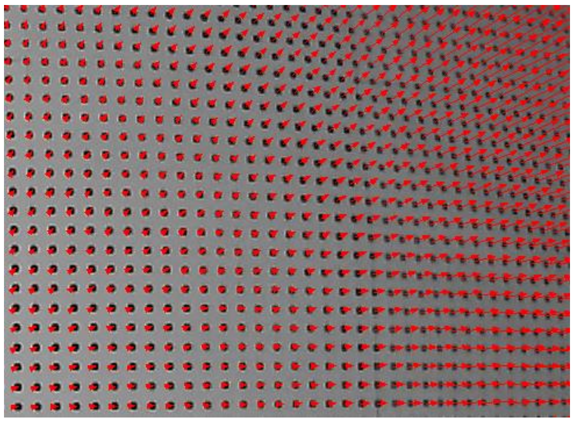


Figure A.4 Calibration model settings (left) and workspace image (right)

Save the resulting calibration file in a ‘PNG’ format.

A.2 Image Processing Algorithm

The image processing for the robot localization is first performed in *Vision Acquisition Express VI* (Fig. A.2). The algorithm includes the following operations, see Fig. A.5.

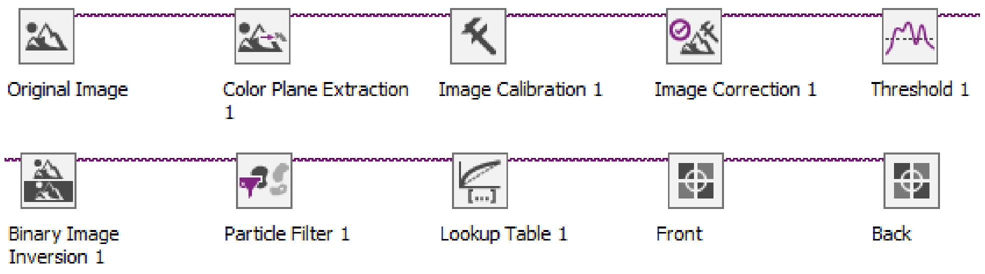


Figure A.5 Block diagram of the image processing algorithm operations

Step 1. Configure the image processing algorithm in *Vision Acquisition Express VI* and create the marker templates.

As shown in Fig. A.5, the algorithm has the following steps, starting from the original camera image:

- Color Plane Extraction

  It converts the original RGB camera image (Fig. A.4) to a gray scale (8-bit) image. Select ‘HSL Luminance Plane’.

- Image calibration

  Select a path to a calibration file from Section A.1.

- Image correction

  This step applies image correction based on the current calibration and removes camera distortion. Select ‘bi-linear’ interpolation type. The corrected image is shown in Fig. A.4 (right).

- Threshold

  Thresholding converts a grayscale image to a binary format (0 or 1). It isolates the objects that need to be kept in the processed image (makes value 1) and removes everything else (makes value 0). We need to keep only the robot markers, as shown in Fig. A.6 (right). Select look for ‘Bright Objects,’ ‘Manual Threshold’ method, and adjust the lower bound value for thresholding objects until the markers only are left.

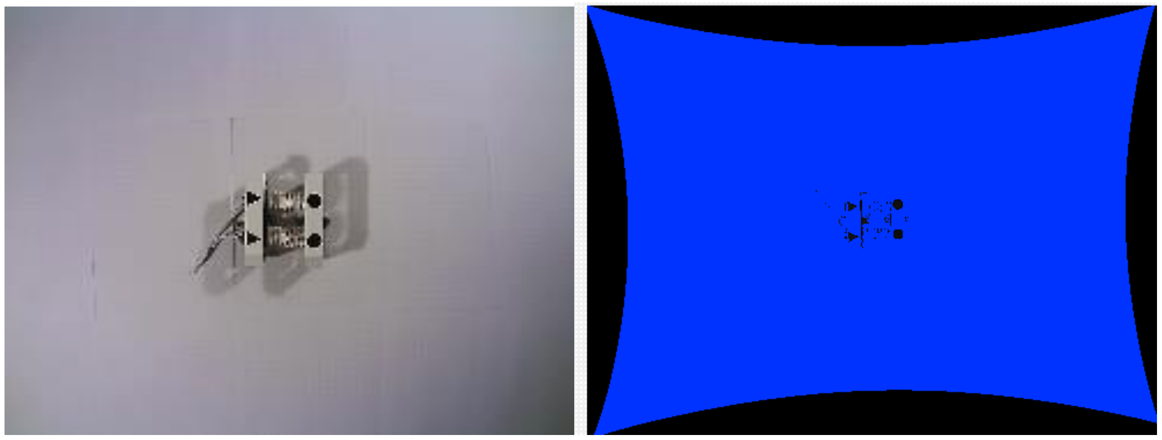


Figure A.6 RGB image of workspace (left); binary image after correction and threshold steps (right)

- Binary image inversion

  This step flips values of 0 and 1 to apply further steps of the algorithm, i.e., it converts Fig. A.6 (right) to Fig. A.7 (left).
- Particle filter

  This step filters the remaining particles (areas to keep in the image) to keep only front and back plates' markers. Select 'Area,' 'Real World,' and enter the area's minimum and maximum values. The particles in this range will be left, as shown in Fig. A.7 (right).

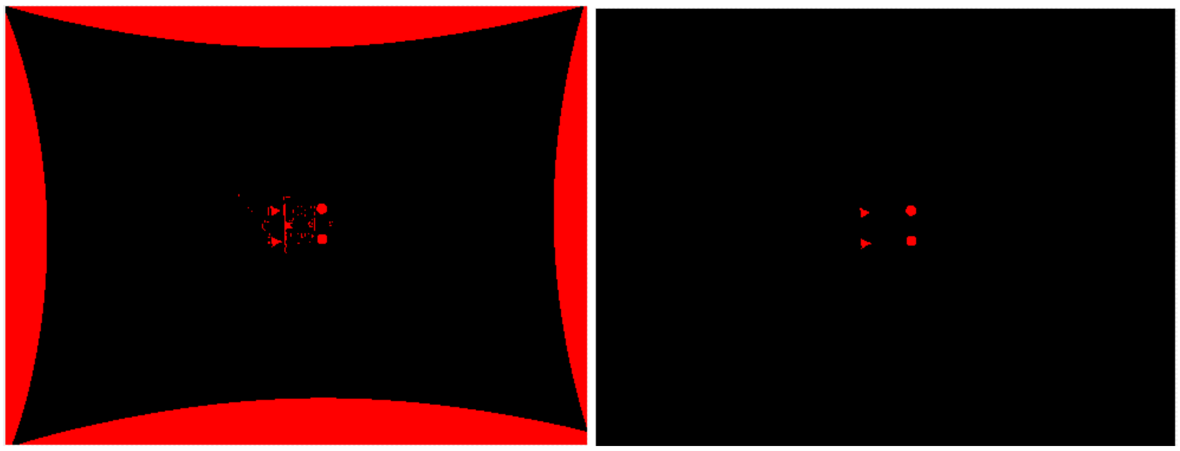


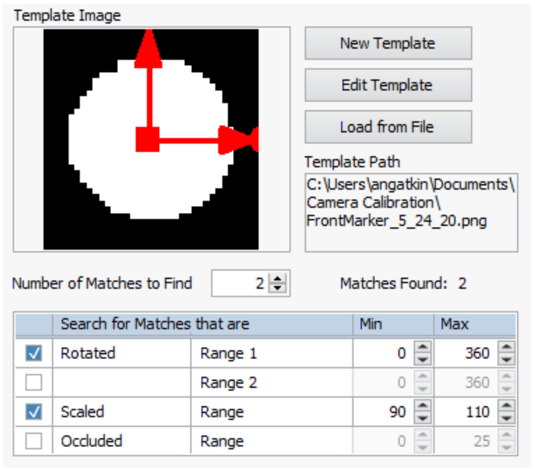
Figure A.7 Image after binary inversion (left); image after applied particle filter (right)

- Lookup Table

Select 'Equalize' in the dialog box. After this step, marker areas will become white color on a black background.

- **Front and back markers (Geometric Matching)**

  These steps create or adjust the robot marker templates. To create or adjust the template for the front plate marker, open the Specification tab (Fig. A.8). When creating a new marker template, align the marker contour (green) with the marker shape, and place the origin in the center; see Fig. A.9. The same steps hold for the back plate marker. Save marker template files in 'PNG' format.



Template Path
C:\Users\angatkin\Documents\Camera Calibration\FrontMarker_5_24_20.png

Number of Matches to Find: 2

Matches Found: 2

	Search for Matches that are		Min	Max
✓	Rotated	Range 1	0	360
☐		Range 2	0	360
✓	Scaled	Range	90	110
☐	Occluded	Range	0	25

Figure A.8 Specification tab in geometric matching

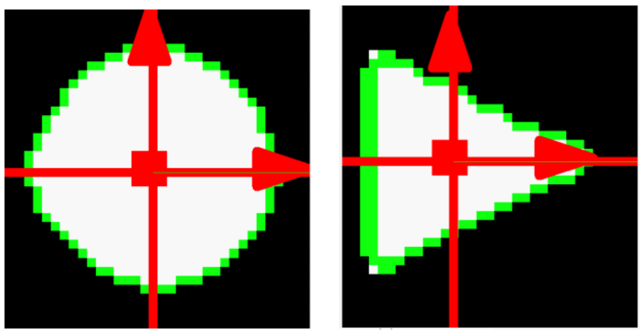


Figure A.9 Marker templates: for front plate (left) and for back plate (right)

Notes:

Simple geometric shape markers of black color on a white background with distinctive shapes should be selected. Here, the circle markers for the front plate and triangle markers for the back plate are used. The front plate markers are our main focus; they always result in the correct detection. The back plate marker may rarely remain undetected even after advanced option tuning. This could be resolved by the marker size increase or a higher resolution camera.

After the marker is created, make the region of interest (ROI), where the algorithm looks for a marker, to the whole image area. Two markers per template are looked for with rotation and scaling in ranges [0 360] degrees and [0.9 1.1], respectively, as shown in Fig. A.8. The *Vision Assistant* uses preset parameters for marker detection. If markers are undetected, tune advanced options in the 'Options' tab. It is done in the robot control VI to improve marker detection.

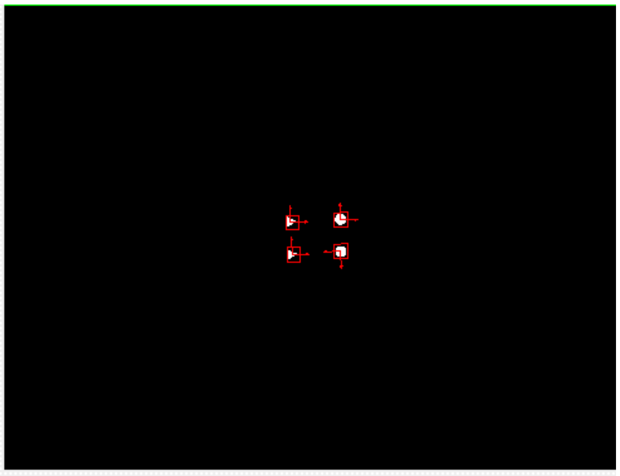


Figure A.10 Processed image showing the identified markers on a black background

The processed image is shown in Fig. A.10. The algorithm outputs are the front and back plate markers' positions in 'Calibrated Matches.' The positions are in the global coordinate frame as defined in the calibration.

Step 2. Convert *Vision Assistant VI* to a subVI and update settings in the *Vision Processing (subVI)* and the robot control VI (described in the next section).

To improve the localization process performance in the main robot control VI, the *Vision Assistant Express VI* is converted to a *Vision Processing (SubVI)*. In this subVI, two markers per plate are located using the following steps:

- First, the algorithm finds one marker (match) per plate, with the ROI being the whole image. The marker's position is **x**.

- Then, it defines a new ROI as a torus with a center in **x**, see Fig. A.11. Its inner radius is slightly larger than the marker radius, and its outer radius is approximately the robot width. Finally, it finds the second marker in this area.

This localization process prevents localization errors and reduces image processing run-time. When two markers are searched simultaneously, the resulting matches could be erroneously located at the same position.

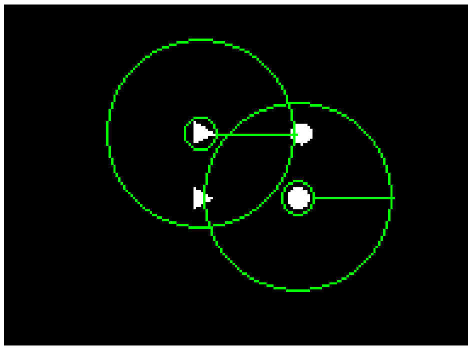


Figure A.11 Fragment of the final processed image with identified markers

To update *Vision Processing (SubVI)*, create a copy of 'Vision_Control.vi.' In the opened copy, right-click on the *Vision Assistant Express VI* and select 'Open Front Panel', which will convert it into a subVI. From a created subVI, update the following settings in the *Vision Processing (SubVI)*: coordinate frame origin, range threshold, and inputs for particle filter. If markers are not detected, lower the minimum bound in the particle filter.

The *Vision Processing (SubVI)* is located in the following directory:

'Box>ARG_Student_Reports>Oyuna Angatkina>Dissertation Supplementary> LabVIEW> Main code V7_2>Vision'

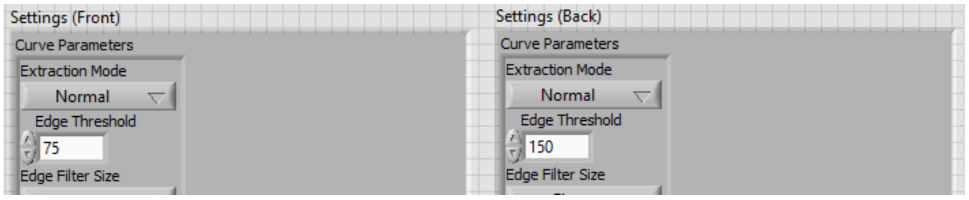
The main robot control VI, called 'Main VI.vi', is located in the following directory:

'Box>ARG_Student_Reports>Oyuna Angatkina>Dissertation Supplementary> LabVIEW> Main code V7_2'

Update the following settings in the front panel of 'Main VI.vi':

- Update the path names for new calibration and front and back plate marker templates. Once updated, right-click and select Data Operation / Make current value Default.

- Update settings for the front and back plate markers. Settings are the clusters with multiple inputs (see their fragment in Fig. A.12); update all changed inputs. To do that, convert ‘Constant’ input into ‘Control’ in the copy of ‘Vision_Control.vi’ and compare with settings in the ‘Main VI.vi’.


Settings (Front)
Curve Parameters
Extraction Mode: Normal
Edge Threshold: 75
Edge Filter Size

Settings (Back)
Curve Parameters
Extraction Mode: Normal
Edge Threshold: 150
Edge Filter Size

Figure A.12 Marker settings inputs on the front panel in ‘Main.vi’

- Change ROI for the whole image if the camera or image quality is changed.

Step 3. Localization Validation

The following sequence of steps could be used to validate the marker localization:

a) Display the corrected image of the calibration grid in ‘Main VI.vi’ and check if the grid dots could be connected into straight lines. If not, update calibration.

b) Run ‘Main VI.vi,’ and if marker positions are not identified, check marker visibility in the processed image. If some markers are missing, lower the ‘min bound’ in the particle filter. However, a small value may result in other objects being present in the processed image, which is undesirable. If markers remain unidentified, adjust advanced settings in *step 1, Front and back markers (Geometric matching),* and copy them to ‘Main VI.vi.’

c) Validate measurements by the following process:

    - Place the center of the robot front plate at (0,0). To do that, with the calibration grid being in the workspace, find origin (0,0) in LabVIEW and the corresponding point in the workspace.
    - Move robot by 100 mm in the x-direction from 0 to 600 mm and verify measurements in LabVIEW. Repeat this process for the y-axis for the range from -200 to 200 mm.

A.3 Robot Control Main VI Operation

The robot in the testbed is operated by the 'Main VI.vi.' This VI implements the described above localization, path following controller and sends the angular inputs to the OSCAR microcontroller, as has been described in Chapter 3 and detailed in Fig. 3.2. The 'Main VI.vi' is located in:

'Box>ARG_Student_Reports>Oyuna Angatkina>Dissertation Supplementary>
LabVIEW> Main Code V7_2'

Its front panel has the following inputs:

- settings for the front and back marker search (see Fig. A.12);
- paths for the calibration and marker template files;
- Arduino serial communication inputs;

and the following main outputs:

- the processed image with identified markers (see Fig. A.11);
- the x-y robot's displacement plot (in mm);
- four marker positions in the global coordinate frame, called 'Calibrated matches.'

The flow of 'Main VI.vi's block diagram is as follows.

1. During initialization, the VI opens the calibration and the front and back marker templates.

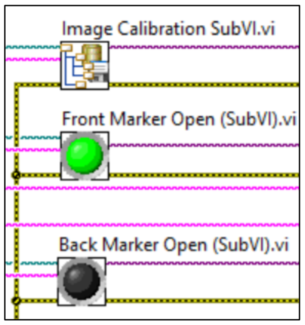


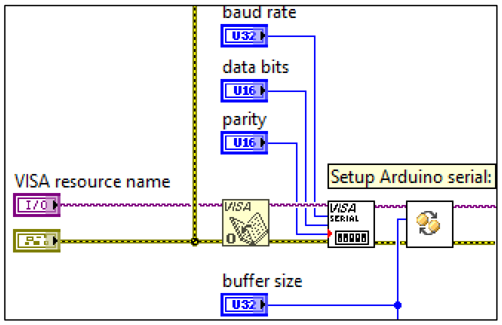
Image Calibration SubVI.vi
Front Marker Open (SubVI).vi
Back Marker Open (SubVI).vi


Figure A.13 Initialization subVIs

Also, it establishes the serial communication with the robot microcontroller. When the following message appears, the robot servos should be reset according to the assembly process described in Section 2.6.2, Chapter 2. After the assembled robot is placed in the testbed workspace, press 'OK' to finish the initialization.

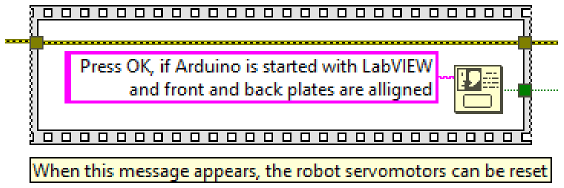

Press OK, if Arduino is started with LabVIEW
and front and back plates are alligned
When this message appears, the robot servomotors can be reset

Figure A.14 Dialog message for completing the robot assembly

2. In the main while loop, following the flow chart in Fig. 3.2, the camera takes the image, and robot localization is performed. The marker positions are scaled to translate measurements from the workspace plane to the robot markers' plane.

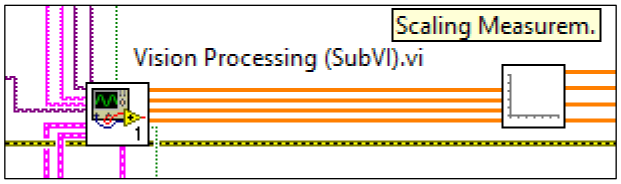
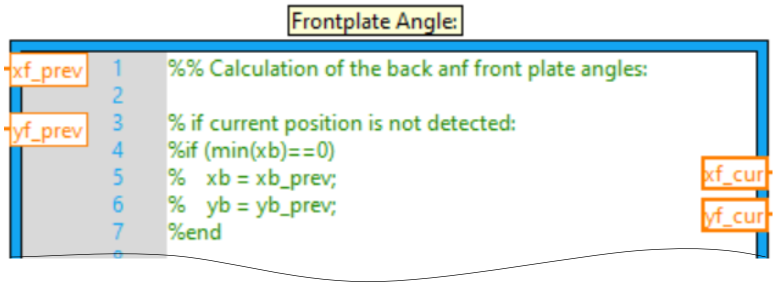


Figure A.15 Image processing and scaling subVIs

The scaling factor is uniform for both the x and y-axis and is equal to 0.865. The factor is defined experimentally by comparing the actual distance between markers and the one obtained in LabVIEW VI. The scaling needs to be updated for changes in the robot height.

3. Then, the robot's front plate orientation is calculated based on the front plate markers' positions, assuming the robot moves in the positive x-axis. The orientation calculation should be adjusted if the robot moves in a different direction.



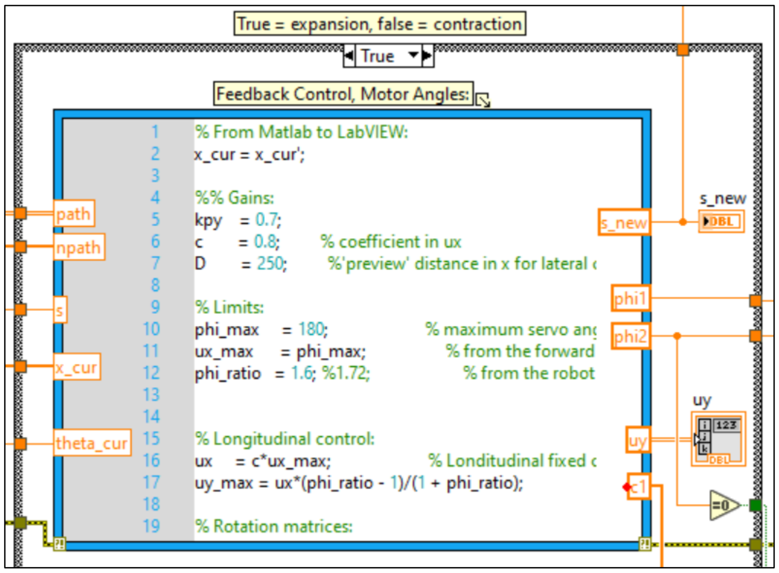
```
%% Calculation of the back anf front plate angles:

% if current position is not detected:
%if (min(xb)==0)
%    xb = xb_prev;
%    yb = yb_prev;
%end
```

Figure A.16 Front plate orientation calculation

4. After that, the angular inputs are defined by either the path following controller or a user input for the robot expansion, which is shown in the 'true' case below. The path following controller is developed in MATLAB Simulink. The robot contraction

corresponds to the ‘false’ case, where the angular inputs are $[\varphi_1\ \varphi_2] = [0\ \ 0]$deg. If the angular inputs need to be constant, disconnect φ_1 (phi1) and φ_2 (phi2) outputs of the feedback path following controller and substitute desired values.


True = expansion, false = contraction
True
Feedback Control, Motor Angles:
path
npath
s
x_cur
theta_cur
s_new
phi1
phi2
uy
c1
s_new
DBL
uy
DBL

```
% From Matlab to LabVIEW:
x_cur = x_cur';

%% Gains:
kpy   = 0.7;
c     = 0.8;      % coefficient in ux
D     = 250;      %'preview' distance in x for lateral

% Limits:
phi_max    = 180;                % maximum servo an
ux_max     = phi_max;             % from the forward
phi_ratio  = 1.6; %1.72;          % from the robot


% Longitudinal control:
ux    = c*ux_max;                 % Londitudinal fixed
uy_max = ux*(phi_ratio - 1)/(1 + phi_ratio);

% Rotation matrices:
```

Figure A.17 Angular inputs the robot calculation

The reference path for the path following controller is obtained in advance by the path planning step, done in MATLAB. The reference path (’path.txt’) and its length (‘npath.txt’) should be provided in the VI folder. The provided reference path is augmented with the virtual path at its end. Therefore, the actual path length (‘npath.txt’) is specified as a second input.

5. The angular inputs are sent to the robot microcontroller via the Visa Write function. When the robot completes expansion or contraction, it sends a flag to the VI through the Visa Read function.
6. Finally, the position and angular inputs data is saved in ‘Saving Data (SubVI).’

Then, steps 2-6 repeat until the ‘Stop’ button is pressed on the front panel. After the experiment, data is saved in the ‘Experimental Data’ folder. The ‘Path Following Experiment Journal.xlsx’ file keeps the information about all experiments.

Appendix B

OSCAR Low-Level Control Code

This following Arduino code implements the OSCAR closed-loop low-level control. It regulates servos angular positions with PI controllers. The code inputs are reference servo angles $[\varphi_1 \;\; \varphi_2]$. The reference inputs to the controllers are ramp signals of slope 5.9 rad/s, as shown in Fig. 3.9 in Chapter 3. The input signal has a first-order filter

$$G(s) = \frac{1}{0.05s + 1} \tag{C.1}$$

Starting at the fully contracted robot state, send expansion angular inputs through serial communication. When the robot expands, send the angular inputs for contraction. The servos are attached to pins 9 and 10 of the robot microcontroller. Pin layout for encoder data reading (from LS7366 encoder counter board) is specified below.

B.1 Arduino Code

```
/* Closed-loop servos' position control of the OSCAR
 * Encoder data reading is based on Dual LS7366 Quadrature Counter Test Code by Jason
Traud on https://github.com/SuperDroidRobots/Encoder-Buffer-Breakout/

   Pins layout:
   LS7366 Breakout    -------------   Arduino Microcontroller
   -----------------                    -------
            MOSI   -------------------   SDO (D11)
            MISO   -------------------   SDI (D12)
            SCK    -------------------   SCK (D13)
            SS1    -------------------   SS1 (D7)
            SS2    -------------------   SS2 (D8)
            GND    -------------------   GND
            VDD    -------------------   VCC (5.0V)
*/
```

```
#include <SPI.h>
#include <Servo.h>

Servo Servo1;
Servo Servo2;

// Slave Select pins for encoders 1 and 2:
const int slaveSelectEnc1 = 7;
const int slaveSelectEnc2 = 8;

// Current encoder count:
signed long encoder1count = 0;
signed long encoder2count = 0;

// Variables:
float f1 = 0.095162581964040;     // Filter gain
float f2 = 0.904837418035960;     // Filter gain

int k1 =  1;     // Origami tower chirality servo 1
int k2 = -1;     // Origami tower chirality servo 2

unsigned long previousMillis = 0;
unsigned long prevMillis = 0;
int var = 0;
boolean LEDstate = 0;

// Servo 1:
unsigned int phiMotor1 = 0;
float angle_des1 = 0;
float angle1 = 0;
float myAng1 = 0;
unsigned int u_PWM1 = 90;
unsigned int u_PWM1_prev = 0;
float ang_print1 = 0;

float e_int_prev1 = 0;
float e_prev1 = 0;

// Servo 2:
unsigned int phiMotor2 = 0;
float angle_des2 = 0;
float angle2 = 0;
float myAng2 = 0;
unsigned int u_PWM2 = 90;
unsigned int u_PWM2_prev = 0;
float ang_print2 = 0;

float e_int_prev2 = 0;
float e_prev2 = 0;

float g11 = 2;     // Servo deadband coefficient
```

```
void setup() {
  Serial.begin(9600);
  initEncoders();
  clearEncoderCount();

  Servo1.attach(9);
  Servo2.attach(10);
  Servo1.write(u_PWM1);
  Servo2.write(u_PWM2);
}

void loop() {

  switch (var) { // Case 0: read angular inputs; Case 1: expansion or contraction.
After case 1 completes, it sends flag '1' to serial port and returns to case 0.
    case 0:
      if (Serial.available() > 0) {
        phiMotor1 = Serial.parseInt();
        phiMotor2 = Serial.parseInt();

        if (phiMotor1 > 190) { // Maximum origami tower input angle = 190 deg.
          phiMotor1 = 190;
        }
        if (phiMotor2 > 190) {
          phiMotor2 = 190;
        }
        // Reference angles:
        angle1 = phiMotor1 * PI / 180;
        angle2 = phiMotor2 * PI / 180;

        var = 1;

        // Current angles:
        ang_print1 = myAng1;
        ang_print2 = myAng2;
      }
      break;
    case 1:
      if ((u_PWM1 - u_PWM1_prev == 0) && (abs(angle1 - myAng1) < 0.05) && (u_PWM2 -
u_PWM2_prev == 0) && (abs(angle2 - myAng2) < 0.05)) {

        // Delay 500 ms after expansion/contraction (to not immediately contract):
        prevMillis = millis();
        while (millis() - prevMillis < 500) {
          ControlPosition(angle1, angle2);
        }

        var = 0;
        Serial.println(1); // Flag

      }
      break;
  }
  u_PWM1_prev = u_PWM1;
  u_PWM2_prev = u_PWM2;

  ControlPosition(angle1, angle2);
}


unsigned int PIcontrol(int k, float angle_d, float myAngf, float &e_int_prev, float
&e_prev) {
  float e = (angle_d - myAngf);
  float e_int = e_int_prev + 0.005 * (e + e_prev) / 2.0;  // Tustin rule
  float u_int = 0.4 * e_int;

  if (abs(u_int) > 0.1) {     // Integral anti-wind-up
    e_int = e_prev;
  }

  float u = 0.6 * e + u_int;

  // Control input saturation:
  if (u > 0.20) {
    u = 0.20;
    e_int = e_int_prev;
  }
  else if (u < -0.20) {
    u = -0.20;
    e_int = e_int_prev;
  }

  int u_c =  k * u * 90;      // Control input to servo

  // Servo deadband compensation (88...92 = '0' speed):
  if (u_c > 0) {
    u_c = u_c + g11;          // Equivalent to 92
  }
  else if (u_c < 0) {
    u_c = u_c - g11;          // Equivalent to 88
  }

  unsigned int u_pwm = 90 + u_c;
  e_int_prev = e_int;
  e_prev = e;

  return u_pwm;
}

void ControlPosition (float ang1, float ang2) {

  if (millis() - previousMillis >= 5) {
    previousMillis = millis();

    // First-order filter on angle:
    float angle_des_updated1 = f1 * ang1 + f2 * angle_des1;
    float angle_des_updated2 = f1 * ang2 + f2 * angle_des2;

    //Rate limiter on desired angle - ramp input:
    if (abs(angle_des_updated1 - angle_des1) * 2000 > 59) { //5.9*10 for integer
      if ((angle_des_updated1 - angle_des1) > 0) {
        angle_des_updated1 = angle_des1 + 0.0295; // 0.0295 = speed*dt = 5.9rad/s*5ms
      }
      else {
        angle_des_updated1 = angle_des1 - 0.0295;
      }
    }
    if (abs(angle_des_updated2 - angle_des2) * 2000 > 59) {
      if (angle_des_updated2 - angle_des2 > 0) {
        angle_des_updated2 = angle_des2 + 0.0295;
      }
      else {
        angle_des_updated2 = angle_des2 - 0.0295;
      }
    }

    myAng1 = k1 * (readEncoder(1) * 2 * PI / 18000.00);
    myAng2 = k2 * (readEncoder(2) * 2 * PI / 18000.00);

    u_PWM1 = PIcontrol(k1, angle_des1, myAng1, e_int_prev1, e_prev1);
    u_PWM2 = PIcontrol(k2, angle_des2, myAng2, e_int_prev2, e_prev2);

    Servo1.write(u_PWM1);
    Servo2.write(u_PWM2);

    angle_des1 = angle_des_updated1;
    angle_des2 = angle_des_updated2;
  }
}


// Following functions from Dual LS7366 Quadrature Counter Test Code by Jason Traud:
void initEncoders() {
  // Set slave selects as outputs
  pinMode(slaveSelectEnc1, OUTPUT);
  pinMode(slaveSelectEnc2, OUTPUT);

  // Raise select pins
  // Communication begins when you drop the individual select signals
  digitalWrite(slaveSelectEnc1, HIGH);
  digitalWrite(slaveSelectEnc2, HIGH);

  SPI.begin();

  // Initialize encoder 1
  //    x4 quatrature count mode (four counts per quadrature cycle)
  digitalWrite(slaveSelectEnc1, LOW);       // Begin SPI conversation
  SPI.transfer(0x88);                       // Write to MDR0
  SPI.transfer(0x03);                       // Configure to 4 byte mode
  digitalWrite(slaveSelectEnc1, HIGH);      // Terminate SPI conversation

  // Initialize encoder 2
  // Same as encoder 1:
  digitalWrite(slaveSelectEnc2, LOW);       // Begin SPI conversation
  SPI.transfer(0x88);                       // Write to MDR0
  SPI.transfer(0x03);                       // Configure to 4 byte mode
  digitalWrite(slaveSelectEnc2, HIGH);      // Terminate SPI conversation
}

long readEncoder(int encoder) {

  // Initialize temporary variables for SPI read
  unsigned int count_1, count_2, count_3, count_4;
  long count_value;

  // Read encoder 1
  if (encoder == 1) {
    digitalWrite(slaveSelectEnc1, LOW);     // Begin SPI conversation
    SPI.transfer(0x60);                     // Request count
    count_1 = SPI.transfer(0x00);           // Read highest order byte
    count_2 = SPI.transfer(0x00);
    count_3 = SPI.transfer(0x00);
    count_4 = SPI.transfer(0x00);           // Read lowest order byte
    digitalWrite(slaveSelectEnc1, HIGH);    // Terminate SPI conversation
  }

  // Read encoder 2
  else if (encoder == 2) {
    digitalWrite(slaveSelectEnc2, LOW);     // Begin SPI conversation
    SPI.transfer(0x60);                      // Request count
    count_1 = SPI.transfer(0x00);           // Read highest order byte
    count_2 = SPI.transfer(0x00);
    count_3 = SPI.transfer(0x00);
    count_4 = SPI.transfer(0x00);           // Read lowest order byte
    digitalWrite(slaveSelectEnc2, HIGH);    // Terminate SPI conversation
  }

  // Calculate encoder count
  count_value = (count_1 << 8) + count_2;
  count_value = (count_value << 8) + count_3;
  count_value = (count_value << 8) + count_4;

  return count_value;
}

void clearEncoderCount() {

  // Set encoder1's data register to 0
  digitalWrite(slaveSelectEnc1, LOW);     // Begin SPI conversation
  // Write to DTR
  SPI.transfer(0x98);
  // Load data
```

```
SPI.transfer(0x00);  // Highest order byte
SPI.transfer(0x00);
SPI.transfer(0x00);
SPI.transfer(0x00);  // lowest order byte
digitalWrite(slaveSelectEnc1, HIGH);    // Terminate SPI conversation

delayMicroseconds(100);  // provides some breathing room between SPI conversations

// Set encoder1's current data register to center
digitalWrite(slaveSelectEnc1, LOW);     // Begin SPI conversation
SPI.transfer(0xE0);
digitalWrite(slaveSelectEnc1, HIGH);    // Terminate SPI conversation

// Set encoder2's data register to 0
digitalWrite(slaveSelectEnc2, LOW);     // Begin SPI conversation
// Write to DTR
SPI.transfer(0x98);
// Load data
SPI.transfer(0x00);  // Highest order byte
SPI.transfer(0x00);
SPI.transfer(0x00);
SPI.transfer(0x00);  // lowest order byte
digitalWrite(slaveSelectEnc2, HIGH);    // Terminate SPI conversation

delayMicroseconds(100);  // provides some breathing room between SPI conversations

// Set encoder2's current data register to center
digitalWrite(slaveSelectEnc2, LOW);     // Begin SPI conversation
SPI.transfer(0xE0);
digitalWrite(slaveSelectEnc2, HIGH);    // Terminate SPI conversation
     }
```

Appendix C

Robot Kinematic Model

The following code is a robot corrected kinematic model implemented in Simulink MATLAB. Given the current robot state, the model calculates the robot state after a single locomotion cycle for provided angular inputs. The robot state is defined as a front plate centroid position and orientation.

C.1 Robot Kinematic Model

```
function [x_new, theta_new] = fcn(phi1, phi2, x_cur, theta_cur, k_loss1)
% Inverse kinematics in the robot coordinate frame (wrt to origin being
current front plate location)
coder.extrinsic('InverseAnalyticalKinematic1');

X = [0;0]; % wrt to origin being current front plate location
Y = [0;0]; % wrt to origin being current front plate location
theta = [0;0]; % Orientation
index = 2;

%% Given constants:
N = 6;                      % Number of cells
cellrelief = 2;             % Number of relief cuts
link_plate = 44;            % Distance between the towers

Lmin = 41; %35-12;          % Length of towers at contracted state
lmax = cellrelief*15;
linitial = Lmin/(N/cellrelief);
deltathetamax = 0.3316;

%Knowns for a single Kresling origami cell (experiment)
alphamax = 0.6283;  % Maximum angle (rad)
stepsize = 200;     % Number of defined increments of cell expansion

% Kresling cell geometry from Pagano Newton-Rhapson geometrical vector loop
solution
```

```
alphas     = linspace(0,alphamax,stepsize);
heights    = 1000*[6.98047095132836e-09, …, 0.015]; % Set of values
heightfunction = pchip(alphas,heights); % Cell height function

%% Calculate robot position and orientation for given input angles:
dx     = 0;
dy     = 0;
theta7 = 0;

if (phi1==0)&&(phi2==0)
    Lleft  = Lmin/(N/cellrelief);
    Lright = Lmin/(N/cellrelief);
else
    Lleft  = (ppval(heightfunction,(phi1/N)))*2;
    Lright = (ppval(heightfunction,(phi2/N)))*2;
end

l1 = Lleft;   % length of vector of two adjacent cells in the left tower
l4 = Lright;  % length of vector of two adjacent cells in the right tower

[theta1,theta2,theta3,theta4,theta5,theta6,theta7,theta8,Lleft, ...
      Lright, dx, dy] = InverseAnalyticalKinematic1(N,cellrelief, ...
      link_plate,X,Y,theta,index, l1,l4);

% Kinematic model correction:
ddx = zeros(2,1);
ddtheta     = (theta7 - pi)*(1 - k_loss1); % k_loss1 = 0.85
ddx(1)      = sqrt((dx + Lmin)^2 + dy^2)*cos(ddtheta/2) - Lmin;
ddx(2)      = sqrt((dx + Lmin)^2 + dy^2)*sin(ddtheta/2);

% Rotation from local robot coord. to global coordinate frame:
R10       = [cos(theta_cur) -sin(theta_cur); sin(theta_cur) cos(theta_cur)];

x_new     = x_cur + R10*ddx;      % Robot position
theta_new = theta_cur + ddtheta;  % Robot orientation
```

C.2 InverseAnalyticalKinematic1.m

```
function [theta1,theta2,theta3,theta4,theta5,theta6,theta7,theta8,Lleft,...
            Lright, x2, y2] = InverseAnalyticalKinematic1(N,cellrelief,...
            link_plate,X,Y,theta,index, l1,l4)
% This function was originally created by Kimberly Gustafson in Fall 2018
Lmin = 41; %35-12;
lmax = cellrelief*15;
linitial = Lmin/(N/cellrelief);
deltathetamax = 0.3316;

% Knowns for a single Kresling origami cell (experiment)
alphamax = 0.6283; % Maximum rotation of a single Kresling origami cell (rad)
stepsize = 200;    % Number of defined increments of Kresling cell expansion
```

```
% Kresling cell geometry from Pagano Newton-Rhapson geometrical vector loop
solution
alphas = linspace(0,alphamax,stepsize);
heights = 1000*[6.98047095132836e-09,…, 0.015];
heightfunction = pchip(alphas,heights);     % Cell height function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Solve Vector Loop Equation %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% With Constraints
theta8 = theta(index-1);       % Previous front plate angle
theta1 = theta8 + (pi/2);

Xback = X(index-1)+Lmin*sin(theta8-(pi/2)); % Back plate center
Yback = Y(index-1)+Lmin*cos(theta8-(pi/2));

fun1 = @(s)((l1*cos(theta1)+l1*cos(s(1))+l1*cos(s(2))- ...
      (link_plate/2)*cos(s(3))-(link_plate/2)*cos(theta8)-s(5)+Yback)^2)+...
      ((l1*sin(theta1)+l1*sin(s(1))+l1*sin(s(2))-(link_plate/2)*sin(s(3))-...
      (link_plate/2)*sin(theta8)-s(4)+Xback)^2)+((l4*cos(theta1)+ ...
      l4*cos(s(1))+l4*cos(s(2))+(link_plate/2)*cos(s(3))+(link_plate/2)*...
      cos(theta8)-s(5)+Yback)^2)+((l4*sin(theta1)+l4*sin(s(1))+...
      l4*sin(s(2))+(link_plate/2)*sin(s(3))+(link_plate/2)*sin(theta8)-...
      s(4)+Xback)^2);

lb = [-pi,-pi,-pi, -10,-100];
ub = [pi,pi,(3/2)*pi, 100,100];
s0 = [0 0 0 0 0];
A = [-1 1 0 0 0; 1 0 0 0 0];
b = [deltathetamax, deltathetamax+theta1];
Aeq = [2,-1,0,0,0;0,-1,1,0,0];
beq = [theta1;pi/2];
options = optimoptions('patternsearch','MeshTolerance',1e-10,...
                                          'StepTolerance', 1e-10);
s = patternsearch(fun1,s0,A,b,Aeq,beq,lb,ub, options);

theta2 = s(1);
theta3 = s(2);
theta7 = s(3);
x2     = s(4);
y2     = s(5);
theta4 = theta1;
theta5 = theta2;
theta6 = theta3;

Lleft = l1;
Lright = l4;

end
```

Appendix D
Path Following Simulation

The following codes are the pure pursuit and feedback path following controller models implemented in Simulink MATLAB. For path following simulation, the controllers are simulated together with the robot kinematic model. For experiments, the codes have been copied to the 'Main VI.vi.'

D.1 PurePursuitController.m

```
function [x_new, theta_new, s_new, phi_left, phi_right, pGoal, R] = ...
                         fcn(x_cur, path, theta_cur, s, k_loss,inputs)
coder.extrinsic('AnalyticalKinematic');

Rmin = inputs(1); % Minimum radius of turn from the robot workspace
ds   = inputs(2); % Max robot forward motion
L    = inputs(3); % Lookahead distance
npath = length(path);

% Initial conditions:
dl      = 41; %35-12;  % Length of the fully contracted origami towers

% Initialization:
pGoalL = [0;0];
phi_left  = 0;
phi_right = 0;
theta7    = 0;

% Rotation matrices:
R01 = [cos(theta_cur)  sin(theta_cur); -sin(theta_cur) cos(theta_cur)];
R10 = [cos(theta_cur) -sin(theta_cur);  sin(theta_cur) cos(theta_cur)];

% Transform from global to local coordinate frame:
xL = R01*x_cur;
pathL = (R01*path')';

if xL(1) <= pathL(end,1) % If the path exists apply pure pursuit
```

```
% Find a goal point on the path:
distNorm = zeros(npath,1);
distNorm(s:end) = sqrt((path(s:end,1)- x_cur(1)).^2 +(path(s:end,2)- ...
                    x_cur(2)).^2);
[c,p] = min(distNorm(s:end));
if c(1)>L
    s = s-1 + p(1);
    s_new = s;
    % Interpolate a virtual point in L from robot (ONLY when robot
    directed toward the path):
    pGoal = x_cur + L*[cos(theta_cur); sin(theta_cur)];
else
    i = 0;
    % Find path point:
    for j = s:length(distNorm)-1
        if distNorm(j)<=L && distNorm(j+1)>L % When robot on path
            i = j;
            break;
        end
    end
    if (i == 0) % When robot close to end of path
        pGoal = path(end,:)';
        s_new = npath;
    else
        if distNorm(j) == L
            pGoal = path(j,:)';
        else
            % Interpolate path point:
            dpGoal=interpPath((path(j,:)'-x_cur),(path(j+1,:)'-x_cur),L);
            pGoal = x_cur + dpGoal;
        end
        s_new = i;
    end
end

e = R01*(pGoal - x_cur);
dy_path = e(2);


%% Apply pure pursuit (calculate radius of turn in local coordinates):
R = L^2/(2*dy_path);
if abs(R)<Rmin       % Turn constraint
    R = sign(R)*Rmin;
end
omega_prime = 2*asin(ds/(2*R));

%% Calculate the next robot position:
dx = ds*cos(omega_prime/2);
dy = ds*sin(omega_prime/2);
dtheta = 2*atan2(dy,(dx+dl));

x_newL = xL+ [dx;dy];
x_new  = R10*x_newL;
theta_new = theta_cur + dtheta;
```

```
    dth = dtheta/(1-k_loss);  % Correction factor k_loss
    dxc = sqrt((dx + dl)^2+dy^2)*cos(dth/2) - dl;
    dyc = sqrt((dx + dl)^2+dy^2)*sin(dth/2);

    %% Calculate angular inputs with segmented kinematic model:
    %% Input Parameters:
    X = [0; dxc];    % x for the center of the front plate (mm)
    Y = [0;-dyc];    % y for the center of the front plate (mm)
    theta = [0;0];   % Orientation of the front plate
    N = 6;             % Number of Kresling cells per tower
    cellrelief = 2;    % Number of cells between relief cuts
    link_plate = 44;   % Distance between origami towers (mm)

    [theta1,theta2,theta3,theta4,theta5,theta6,theta7,theta8,Lleft, ...
      Lright,phi_left,phi_right] = AnalyticalKinematic(N,cellrelief, ...
      link_plate,X,Y,theta,2);

    % Check orientation increment (dtheta) calculation
    % dtheta - (theta7 - pi)
else
    x_new = x_cur;
    theta_new = theta_cur;
    s_new = s;
    phi_left = 0;
    phi_right = 0;
    pGoal = [0; 0];
    R = 0;
end


function p = interpPath(a,b,L)
% Finds the interpolated path point between a and b in lookahead distance L
from the current location, assuming the current location is at [0,0] of local
coord. frame

% Equations (2),(3) are substituted to 1 and solved for u
% p(1)^2 + p(2)^2 = L^2;        (1)
% a(1)*(1-u) + u*b(1) = p(1)    (2)
% a(2)*(1-u) + u*b(2) = p(2)    (3)

a1 = (b(1) - a(1))^2 + (b(2) - a(2))^2;
b1 = 2*(a(1)*b(1) + a(2)*b(2) - a(1)^2 - a(2)^2);
c1 = a(1)^2 + a(2)^2 - L^2;

% Solve a1* u^2 + b1*u + c1 = 0, 0<=u<=1
% Calculate u and p:
u1 = (-b1+ sqrt(b1^2 - 4*a1*c1))/(2*a1);
u2 = (-b1- sqrt(b1^2 - 4*a1*c1))/(2*a1);
if u1>=0
    u = u1;
else
    u = u2;
end
p = [a(1) a(2)]'*(1 - u) + u*[b(1) b(2)]';
```

D.2 AnalyticKinematic.m

```
function [theta1,theta2,theta3,theta4,theta5,theta6,theta7,theta8,Lleft, ...
      Lright,phi_left,phi_right] = AnalyticalKinematic(N,cellrelief, ...
      link_plate,X,Y,theta,index)
% This function was originally written by Kimberly Gustafson, Fall 2018

Lmin = 41;%35-12;
lmax = cellrelief*15;
linitial = Lmin/(N/cellrelief);
deltathetamax = 0.3316;

% Knowns for a single Kresling origami cell (experiment)
alphamax = 0.6283;  % Maximum rotation (rad)
stepsize = 200;     % Number of defined increments of cell expansion

% Kresling cell geometry from Pagano, Newton-Rhapson geometrical vector loop
solution
alphas = linspace(0,alphamax,stepsize);
heights = 1000*[6.98047095132836e-09, …, 0.015];
heightfunction = pchip(heights,alphas); % Cell height

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Solve Vector Loop Equation %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% With Constraints
theta8 = theta(index-1);     % Previous front plate angle
theta1 = theta8 + (pi/2);

Xback = X(index-1)+Lmin*sin(theta8-(pi/2));
Yback = Y(index-1)+Lmin*cos(theta8-(pi/2));

fun1 = @(s)((s(1)*cos(theta1)+s(1)*cos(s(3))+s(1)*cos(s(4))-
(link_plate/2)*cos(s(5))-(link_plate/2)*cos(theta8)-Y(index)+Yback)^2)+ ...
           ((s(1)*sin(theta1)+s(1)*sin(s(3))+s(1)*sin(s(4))-
(link_plate/2)*sin(s(5))-(link_plate/2)*sin(theta8)-X(index)+Xback)^2)+ ...
((s(2)*cos(theta1)+s(2)*cos(s(3))+s(2)*cos(s(4))+(link_plate/2)*cos(s(5))+(li
nk_plate/2)*cos(theta8)-Y(index)+Yback)^2)+ ...
((s(2)*sin(theta1)+s(2)*sin(s(3))+s(2)*sin(s(4))+(link_plate/2)*sin(s(5))+(li
nk_plate/2)*sin(theta8)-X(index)+Xback)^2);

lb = [linitial,linitial,-pi,-pi,-pi];
ub = [100,100,pi,pi,(3/2)*pi];
s0 = [0 0 0 0 0];
A = [];
b = [];
Aeq = [0,0,2,-1,0;0,0,0,-1,1];
beq = [theta1;pi/2];
s = fmincon(fun1,s0,A,b,Aeq,beq,lb,ub);

Lleft = s(1);
Lright = s(2);
theta2 = s(3);
theta3 = s(4);
```

```
theta7 = s(5);

theta4 = theta1;
theta5 = theta2;
theta6 = theta3;

phi_left = ((ppval(heightfunction,(real(Lleft)/cellrelief)))*N);
phi_right = ((ppval(heightfunction,(real(Lright)/cellrelief)))*N);

end
```

D.3 FeedbackController.m

```
function [phi1, phi2, s_new] = fcn(path, x_cur, theta_cur, inputs, s,npath)
% Path following controller calculates angular inputs to robot based on
% constant longitudinal control (ux) and lateral proportional control(uy)
% Lateral error is found in 'preview' distance D
% Path input is path_aug = path + virtual path(interpolation for the path
end), npath=length(path)

% Gains
kpy       = inputs(1);    % p-gain for lateral control
D         = inputs(3);    % preview distance
c         = 0.8;
% Upper bounds
phi_max   = 180;          % maximum servo angle for tower expansion
ux_max    = phi_max;      % maximum longitudinal motion
phi_ratio = 1.6;          % maximum angle ratio from the robot workspace;
                             ideal is 1.8033
% Longitudinal control input
ux        = c*ux_max;
uy_max    = ux*(phi_ratio - 1)/(1 + phi_ratio);  % turn constraint

% Convert to the local coordinate frame:
R01 = [cos(theta_cur) sin(theta_cur); -sin(theta_cur) cos(theta_cur)];
pathL = (R01*path')';
xL = R01*x_cur;

if (s==0)                     % protection if s becomes zero
    s = 1;                    % s is current ref. path point index
end

% Feedback controller:
if (xL(1) <= pathL(npath,1))
    j = s;                   % starting from current path index
    k = -1;                  % set initial value for k

    % Point in 'preview' distance D from robot:
    c0 = x_cur + D*[cos(theta_cur) sin(theta_cur)]';
    while (k<0)||(k>1)
        if path(j+1,1)<path(j,1)
            k = abs((path(j+1, :) - path(j, :))*(c0 - path(j,:)'))/...
```

```
                    (norm(path(j+1, :) - path(j, :)))^2;
        else
            k = (path(j+1, :) - path(j, :))*(c0 - path(j,:)')/...
                    (norm(path(j+1, :) - path(j, :)))^2;
        end
        if (k>1)
            j = j+1;  % update the path point index
        elseif (k<0)
            k = 0;
        end
    end
    c1 = ((1 - k)*path(j,:) + k*path(j+1,:))';
    eyL = [0 1]*R01*(c1 - c0);

    % Lateral error:
    ey = norm(c0 - c1)*sign(eyL);

    s_new = j;

    % Lateral control input:
    uy = kpy * ey;
    % Turn constraint
    if  abs(uy) > uy_max
        uy = sign(uy)*uy_max;
    end

    %% Servo angles (phi):
    A = 1/2*[1 1; -1 1];
    phi = A\[ux; uy];
    phi1 = phi(1);      % (deg.)
    phi2 = phi(2);      % (deg.)

    %% Delete this for LabVIEW implementation:
    phi1 = phi1*pi/180;
    phi2 = phi2*pi/180;
else                          % After path is finished
    phi1 = 0;
    phi2 = 0;
    s_new = s;
end
```

Appendix E

Path Planning Implementation

The path planning is done by a hybrid A* planner in MATLAB. To plan a path, we first create a workspace cost map with obstacles, then call a planner. The reference path is divided into two parts due to workspace limitation. Each part path is augmented with virtual points and transformed into a text file for implementation in 'Main VI.vi.'

E.1 PotentialFieldFunction.mlx

```
% Creating cost map for hybrid A* planner (x,y in mm):
x_map0 = [0; 0]       % min
x_map  = [1300; 400]  % max
res    = 1   % mm
x = x_map0(1):res:x_map(1);
y = x_map0(2):res:x_map(2);
costVal = zeros(length(y), length(x));

% Adding circular obstacles to map:
X_obst = [150, 70; % center (mm)
          550, 150];
R_obst = [20;20]; % radius (mm)

for i = 1:length(R_obst)
    df = 65; % bound around obstacle
    costVal = obstacleToMap(R_obst(i),X_obst(:,i),costVal,res,x_map0,...
                                   x_map, df)
end

surf(costVal, 'EdgeColor', 'none')
xlabel('x(mm)'); ylabel('y(mm)'); zlabel('Cost');

% Obstacle to map:
function costValOut = obstacleToMap(r_obst,x_obst,costVal,res,x_map0,...
                                    x_map, df)
x1 = floor((x_obst(1)-r_obst)/res)-2*res-df :res: ...
                                    floor((x_obst(1)+r_obst)/res)+2*res+df;
```

```
% exclude values out of the map boundaries:
k = (x1>=x_map0(1)) & (x1<=x_map(1));
x = x1(k);

y1 = floor((x_obst(2)-r_obst)/res)-2*res-df :res: ...
                                    floor((x_obst(2)+r_obst)/res)+2*res+df;
% exclude values out of the map boundaries:
k = (y1>=x_map0(1)) & (y1<=x_map(1));
y = y1(k);

for i =1:length(x) % update cost
    for i =1:length(x)
        d_obst = sqrt((x(i)-x_obst(1))^2+(y(j)-x_obst(2))^2)-r_obst;
        if d_obst <=0 % cost value inside obstacle
            costVal(length(costVal(:,1)) - y(j), x(i)) = 1;
        elseif (d_obst >0 && d_obst <= df)
            alpha = 1;
            costVal(length(costVal(:,1)) - y(j), x(i)) = ...
                                    alpha/(alpha + d_obst);
        end
     end
end
costValOut = costVal;
end
```

E.2 Planner_HybridAstar.mlx

```
% Create an obstacle cost map
PotentialFieldFunction
% Create a binaryOccupancyMap with cost values (0 or 1)
map = binaryOccupancyMap(costVal);
% Create a stateValidator object for collision checking
validator = validatorOccupancyMap;
validator.Map = map;
show(map)

% Initialize planner
planner = plannerHybridAStar(validator,'MinTurningRadius', 467.6, ...
            'MotionPrimitiveLength', 30, 'NumMotionPrimitives', 19, ...
            'AnalyticExpansionInterval', 100, 'ReverseCost', 1000000000000);

% Start and goal configurations (mm,mm,rad)
startPose = [10,200,0];
goalPose  = [1200,5,0];

% Plan a path from start to goal
refpath = plan(planner,startPose,goalPose);
refpath = refpath.States; % output path
show(planner)
```

E.3 PathTransform.mlx

```
% Path is divided into two parts due to workspace limitations:
x_lim = 646;
% Path 1:
```

```
k = find(refpath1(:,1)<x_lim)
path1 = refpath1(k,1:3);
path  = path1(:,1:2);
npath = length(path(:,1));
theta_ref = path1(:,3);

% Augmented virtual path is the whole path for left part of path (x<646 mm):
path_aug = refpath1(:,1:2);

% Save path for experiment
dlmwrite('path.txt', path_aug);
dlmwrite('npath.txt', npath)

% Path following simulation
clear out
x0 = path1(1,:);
npath = length(path(:,1));
out = sim('FeedbackOnly')

%% Path 2:
x_lim = 636;
k = find(refpath1(:,1)>x_lim)
path1 = refpath1(k,1:3) - ones(length(k),1)*[x_lim,0,0];
path  = path1(:,1:2);
npath = length(path(:,1));

x_aug = [20:20:400]';
y_aug = zeros(length(x_aug),1);

theta_ref = path1(:,3);
theta_re = theta_ref(end);

R01 = [cos(theta_re) -sin(theta_re); sin(theta_re) cos(theta_re)];
path_aug = [path; path(end, :) + (R01*[x_aug y_aug]')'];

% Save path for experiment
dlmwrite('path.txt', path_aug);
dlmwrite('npath.txt', npath)

% Path following simulation
clear out1
x0 = path1(1,:);
npath = length(path(:,1));
out1 = sim('FeedbackOnly')
```

Appendix F
Supplementary Files

The supplementary files in this dissertation contain the video recordings of the experiments highlighting the main results. They are recorded with an overhead GoPro camera; see Chapter 3. The supplementary files contain the following list:

1. *‘Ch5_Path_Following_Feedback_Part1.mp4’* is the video recording of experiment 1 in Fig. 5.6. It shows the first part of the straight path following with the feedback controller. As can be seen, the robot being initially offset from the path successfully converges to it. Due to workspace limitations, the second experiment has been conducted to show the robot’s ability to follow the path after convergence.
2. *‘Ch5_Path_Following_Feedback_Part2.mp4’* is a video recording of the second part of the straight path following with the feedback controller, which demonstrates OSCAR successfully following the straight reference path. It corresponds to experiment 2 in Fig. 5.6.
3. *‘Ch6_AutonomousNav_Feedback_Part1.mp4’* is a video recording of the first part of the OSCAR’s autonomous navigation experiment with the planned reference path in Scenario 1. It corresponds to experiment 1 in Fig. 6.7.
4. *‘Ch6_AutonomousNav_Feedback_Part2.mpeg’* is a video recording of the second part of the robot autonomous navigation with planned reference path in Scenario 1. It corresponds to experiment 2 in Fig. 6.7. Together this and the previous experiment demonstrate the thesis’s main result, the soft robot autonomous navigation.
5. *‘Ch7_Coupled_Locomotion.mp4’* is a video recording of the coupled locomotion of two OSCARs, presented in Fig. 7.4-7.6.