MARMARA UNIVERSITY
FACULTY OF ENGINEERING

ORIGAMI INSPIRED PARALLEL ROBOT DESIGN

Ahmet KAYIKÇI, Mehmet Salih HAMARAT

GRADUATION PROJECT REPORT

Department of Mechanical Engineering

Supervisor

Assoc. Prof. Dr. İbrahim Sina KUSEYRİ

ISTANBUL, 2023

MARMARA UNIVERSITY
FACULTY OF ENGINEERING

ORIGAMI INSPIRED PARALLEL ROBOT DESIGN

By

Ahmet KAYIKÇI, Mehmet Salih HAMARAT

July 3, 2023, ISTANBUL

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

BACHELOR OF SCIENCE

AT

MARMARA UNIVERSITY

Signature of Author(s)

Department of Mechanical Engineering

Certified By Assoc. Prof. Dr. İbrahim Sina KUSEYRİ

Project Supervisor, Department of Mechanical Engineering

Accepted By Prof. Dr. BÜLENT EKİCİ

Head of the Department of Mechanical Engineering

ACKNOWLEDGEMENT

First of all, we would like to express our heartfelt gratitude to our supervisor, Assoc. Prof. Dr. İbrahim Sina KUSEYRİ, for his unwavering support and invaluable guidance throughout the journey of preparing this thesis. His expertise, encouragement, and constructive feedback have been instrumental in shaping this research work.

We extend our sincere appreciation to Asst. Prof. Dr. Uğur TÜMERDEM for his invaluable insights and critical feedback at various stages of this endeavor. His expertise has played a pivotal role in refining the scope and methodology of our research. We are truly grateful for his patience, understanding, and unwavering dedication in providing us with the necessary guidance and knowledge to navigate the complexities of this project. His mentorship has been a constant source of inspiration and has greatly contributed to the overall quality of this thesis.

Furthermore, we would like to acknowledge the valuable contributions of Ph.D. student Nural YILMAZ, whose collaboration and prior studies in this field have been of immense assistance in advancing our research.

Lastly, we express our heartfelt appreciation to our family and friends for their unwavering support and encouragement throughout this project.

July 2023 **Ahmet KAYIKÇI, Mehmet Salih HAMARAT**

CONTENTS

ABSTRACT

Origami-inspired engineering has gained significant importance as it offers several advantages and challenges, especially in the field of robotics. Developing origami robots with 3 degrees of freedom (DOF) using a parallel robot design is the focus of our graduation project. Our goal is to create an optimal origami robot by determining the best origami pattern, maximizing the workspace, enabling an average 90-degree rotation like a wrist mechanism, and ensuring minimal robot size.

Throughout the project, we will primarily investigate the kinematics and workspace analysis of various 3-DOF RSR origami robots with distinct designs. We will conduct these analyses in a computer environment, considering static structural aspects as well. The outcomes of this study will serve as a guide for designing, analyzing kinematics and structures, and eventually fabricating the desired RSR 3-DOF parallel origami robots, tailored to specific requirements and applications.

As we move forward, our future plans involve delving deeper into fabrication techniques and including a control system for the robot. Furthermore, we aim to optimize the robot's size further and explore different origami configurations to broaden the scope of possibilities.

SYMBOLS

α, β	: Orientation angles top platform
r	: Thrust distance
$\theta_i; i = 1,2,3$	: Angles between base and links connected to the base
$E_i; i = 1,2,3$	: Intersection points of revolute joint
$F_i; i = 1,2,3$	: The fixed locations of base
${}^{B}l_i; i = 1,2,$	: Unit vector directions
$\varphi_i; i = 1,2,3$	: Rotation angles of ${}^{B}l_i$ onto base
$D_i; i = 1,2,3$	: Length of links
c_i, s_i	: $\cos(\theta_i)$, $\sin(\theta_i)$
$\hat{n}_p$	: The normal to the top platform
$\hat{n}_B$	: The normal to the base
$n_{P_x}, n_{P_y}, n_{P_z}$	: Components of $\hat{n}_p$
P	: Length from base to platform
x_i	: Vector which is perpendicular to $\hat{u}$
x_0, y_0, z_o	: Components of origin of base
$\hat{u}$	: Midplane normal
$\hat{u}_x, \hat{u}_y, \hat{u}_z$	: Components of $\hat{u}$
Y_i, Z_i, V_i	: Inverse kinematics coefficients
ξ	: Bending angle
$\delta_i; i = 1,2,3,4,5,6$	: Joint angles of origami patterns
λ	: Design angle of origami patterns

γ	: Rotation angle
P_{ij}	: Constant positions of upper link
F_{ij}	: Constant positions of bottom link
U_{ij}	: Constant positions of RTT mechanism in upper part
B_{ij}	: Constant positions of RTT mechanism in lower part
$\overrightarrow{BU_{ij}}$	: Direction vector from U_{ij} to B_{ij}
$\overrightarrow{UE_{ij}}$	: Direction vector from E_i to U_{ij}
$\overrightarrow{EB_{ij}}$	: Direction vector from B_{ij} to E_i
R	: Inner radius of RTT mechanism
${}^{B}_{P}T$	: Transformation matrix from base to top platform
P_i	: End effector position of T matrix
P_{ix}, P_{iy}, P_{iz}	: Components of P_i
lb	: Lower boundary of δ_1 and δ_4
ub	: Upper boundary of δ_1 and δ_4
T	: Thickness of the links

ABBREVIATIONS

DOF	: Degree of Freedom
RSR	: Revolute-Spherical-Revolute
RTT	: Resch Triangular Tessellation
GA	: Genetic Algorithm
TPU	: Thermoplastic Polyurethane
PLA	: Polylactic Acid
JW	: Joint Width
JT	: Joint Thickness
DH	: Denavit-Hartenberg

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Origami, with its unique principles, offers numerous advantages in robotics, particularly for the production of small-sized robots. While there is a significant body of research exploring the application of origami principles in robotics, studies on 3-DOF revolute-spherical-revolute (RSR) parallel robots have been relatively limited, and a comparative analysis of origami robots using different patterns is lacking.

In this project, we have undertaken a comprehensive study on three origami robots, each designed with distinct origami patterns, in order to evaluate their respective advantages and disadvantages based on two fundamental design criteria: maximum workspace volume and maximum bending angle. To facilitate the analysis, we have developed a formulation that allows us to adjust the joint limits of the origami pattern, thereby determining the robot's workspace. We have utilized Simulink to validate and refine the obtained workspace.

Fabricating 3-DOF parallel origami robots can pose challenges. While existing literature primarily employs composite techniques, we have explored the potential of 3-D printing as a faster and more cost-effective alternative. Although our origami conversion formulation adheres to rigid origami principles, incorporating flexible materials in 3-D printing for joints and links contradicts this principle. To address this concern, we conducted a finite element analysis using ANSYS prior to manufacturing, allowing us to gain insights into the structural behavior for the selected material. Additionally, we have considered the possibility of employing multi-material 3-D printing as an alternative approach for producing origami robots.

Finally, we emphasize the importance of incorporating optimization into the robot design. In that way, we enhance the overall performance and functionality of the origami robots. We believe that our study will serve as a valuable guide for the design, analysis, and fabrication of 3-DOF origami robots, providing a solid foundation for their application in various domains.

1.1 Literature Survey

Origami is the ancient art of folding paper, where a single sheet is carefully manipulated to create a wide array of intricate shapes. What makes origami truly special is its fundamental principle of relying solely on folding techniques. It strictly avoids stretching, cutting, or adding extra pieces of paper [1]. Origami, originally valued for its aesthetics, has now transcended its artistic origins, and found practical applications in numerous fields, thanks to advancements in computer science and computational geometry. The principles of origami have been successfully integrated into diverse domains, including architecture [2] [3], robotics [4] [5], biomedical devices [6] [7], optical systems [8] [9], space telescope [10], and many more. This versatile folding technique offers not only complexity but also cost-effective advantages, making it an attractive choice for various industries. The expansion of origami into different disciplines has opened new avenues for innovation, allowing for the exploration of groundbreaking possibilities.

There are the most known 4 basic origami patterns in literature. These are the Miura-ori pattern, Waterbomb pattern, the Yoshimura pattern, and the Diagonal pattern, respectively. The Miura-ori pattern is a notable origami pattern renowned for its unique property of rigid foldability. Initially employed in solar panels [11], this pattern possesses the remarkable ability to transition from a folded state to a fully expanded, flat state. The distinctive characteristic of rigid foldability ensures that the folded structure maintains its shape and rigidity throughout the folding and unfolding process [12]. The Waterbomb pattern is a remarkable origami pattern characterized by a single-vertex bistable structure. Its distinctive geometry and wide range of motion provide flexibility for various design possibilities [13]. A stent designed with the Waterbomb base pattern exhibits a unique feature: it can collapse into a smaller size when navigating through veins [14]. Furthermore, the kinematic and mechanical characteristics of the waterbomb pattern are investigated in [15]. The kinematic formulation of the waterbomb cell is derived from the Denavit Hartenberg (DH) formulation and incorporates loop closure equations, which act as an objective function to determine the specific joint angles of the cell. In addition to analytical investigation, the study also encompasses a comprehensive analysis including Workspace and Finite Element Analysis. The Yoshimura pattern, known for its diamond-shaped folds along one of its diagonals, is commonly employed in cylinder folding techniques. This pattern exhibits a notable buckling behavior when subjected to axial compression [16].

An origami tessellation is a folding design where repeated elements are used in both the crease pattern and the folded state. The study specifically highlights the valuable contributions made by David A. Huffman, a distinguished mathematician, computer scientist, and artist, in the realm of straight crease designs, particularly origami tessellations [17]. The Resch Triangular Tessellation (RTT), patented by Ronald D. Resch [18], is an exemplary representation of origami tessellations. It exhibits a characteristic structure that shares similarities with the Waterbomb pattern, albeit with distinct triangular patterns and crease configurations.

The application of origami as an engineering tool has proven to be immensely valuable in the field of robotics. By leveraging the principles of this ancient art, engineers have been able to transform 2-D structures into dynamic 3-D structures, giving rise to a new category of robots known as origami robots [19]. The process of folding along specific lines within an origami structure exhibits a revolute joint behavior, resulting in rotational motion. This unique rotational capability has been leveraged for the development of robotic systems based on origami folding principles [20]. By combining the foldability property with the minimization principle, the design space for micro-scale robots is expanded [21]. This attribute is particularly valuable in the field of surgical systems, where miniaturization is of paramount importance [22]. Additionally, compliant mechanisms can be designed using origami techniques due to their inherent similarities [23]. Although conventional methods are often the initial approach when creating robots, they encounter challenges such as resistance to rotation due to link and joint inertia [24], as well as size limitations imposed by the numerous components, such as nuts and bolts, involved in the assembly. The use of high-strength materials further restricts the rotation of flexure hinges to small angles of a few degrees [25], despite the high precision in motion provided by compliant mechanisms within the micrometer range.

A robot design is made from 2 triangular bases and 3 waterbomb bases which act as a parallel origami structure [26]. This mechanism shares similarities with the revolute-spherical-revolute (RSR) mechanism, commonly referred to as the spherical wrist [27]. The kinematic analysis of the 3-RSR wrist is conducted by [28]. A new mathematical method to analyze the singularity of a 3-RSR parallel mechanism is presented in fixed and all-attitude multiple motion modes in [29]. By employing reciprocal screw methods and linear geometry theory, the study provides theoretical guidance for the selection of mechanism drive and time-sharing control, aiming to

avoid singularity and optimize the performance of the parallel mechanism. The design, kinematic analysis, control architecture, and experimental validation of a novel 3-DOF robotic surgical instrument for minimally invasive surgery is developed in [30]. For the same purpose, a novel 4-DOF origami grasper is studied that is developed for Minimally invasive surgery (MIS) [31]. The paper puts an investigation on kinematics mapping of the origami parallel structure.

Parallel manipulators, known as parallel robots or parallel mechanisms, are advanced robotic systems consisting of multiple arms or limbs connected in parallel to a common base and end effector. Their workspace determination poses significant challenges due to the closed kinematic chains that connect the base and top platform. These challenges are further compounded when dealing with origami robots, as the joint limits of the folding structure restrict their motion. Accurate workspace definition for origami parallel manipulators requires meticulous analysis of the joint limitations inherent in their unique folding design. Extensive research has been dedicated to determining and optimizing the workspace of parallel manipulators, employing various techniques in robotics. One notable approach utilizes Computer-Aided Design (CAD) software, such as CATIA. This involves individually determining the reachable region for each leg of a 3-RPR Planar manipulator through CAD modeling in CATIA [32]. Simulating the motion of each leg allows for obtaining the reachable regions of the end effectors. By applying a Boolean intersection function to the individual reachable regions of each leg, the overall workspace of the parallel manipulator can be determined. This methodology is also employed in [33], using CATIA Software to determine and parameterize the workspace of RRRR planar and Delta parallel robots. Additionally, the workspace volume of the Delta robot is incorporated into an optimization algorithm as an objective function, aiming to maximize the robot's capabilities.

Another study focuses specifically on a 3-DOF Delta parallel robot actuated by prismatic joints, employing an analytical solution for workspace analysis. An optimization process is then conducted, comparing the CAD features and the analytical solutions [34]. In [35], CAD-based analysis and optimization methods are discussed for a 6-DOF parallel kinematics machine (PKM), highlighting its efficiency for complex structures. Furthermore, genetic algorithm optimization is employed for a 3-DOF parallel manipulator used in drilling operations [36].

Origami's applicability extends to embedded haptic feedback systems, enriching user experience with tactile sensations. The 3-DOF RSR mechanism, comprising a desktop, actuators, torque transmission mechanism, and parallel origami mechanism, is specifically designed for this purpose, incorporating fabrication considerations [37]. The robot body consists of three adhesive-bonded layers: fiberglass layers on top and bottom for structural rigidity, and a polyamide layer between them to facilitate flexibility along the creases of the revolute and spherical waterbomb mechanism. Notably, the creases on the fiberglass layer have been intentionally removed, and this fabrication was extended for Delta mechanism robots. The same manufacturing technique also employed exclusively for the link of the 3-DOF RSR origami robot, known as Ori-pixel [38], has been utilized in a distinct operational context. Each leg is seamlessly affixed to the actuators via a transmission mechanism, enabling them to contribute to the robot's overall movements. These actuators are interconnected with slave units, which are centrally controlled by a master board. This arrangement facilitates the precise manipulation of the robot's shape, encompassing all pixels, thereby facilitating effective flag interaction. Furthermore, an exploration of the impact of both high and low P controllers has been undertaken to ascertain optimal control strategies for individual pixels. In addition, two different fabrication methods which were examined in a study to assess their effectiveness in producing the Multi-Modal Locomotion (MML) origami robot, capable of executing both jumping and crawling modes [39]. These are layer-by-layer fabrication and 3-D printer replacing for producing the feet pads, stopper, and glass fiber. The investigation involved comparing the weight, fabrication time, and layer count of the two methods while maintaining identical dimensions and SMA spring actuator fabrication.

2 KINEMATIC ANALYSIS

Kinematic position analysis plays a crucial role in developing an accurate mathematical representation of a robot's architecture. This analysis involves examining the system's geometry and constraints to establish a strong relationship between input parameters, such as joint angles, and the resulting changes in output parameters, including the position and orientation of the end effector.

To begin, we will first derive the kinematics of the revolute-spherical-revolute (RSR) robot. Once we have established the kinematic behavior of the RSR robot, we will then introduce and analyze the constraints required for incorporating RSR parallel origami robots. This step-by-step approach will allow us to gain a comprehensive understanding of the kinematics involved in the entire system.

2.1 Kinematics of RSR Robot

Forward kinematics involves calculating the values of α, β and r based on given input angles θ_1, θ_2 and θ_3. This process enables us to determine the resulting position and orientation of the end effector. In contrast, inverse kinematics aims to determine the desired input angles θ_1, θ_2 and θ_3 that will yield a specified α, β and r, as illustrated in Figure 2.1.

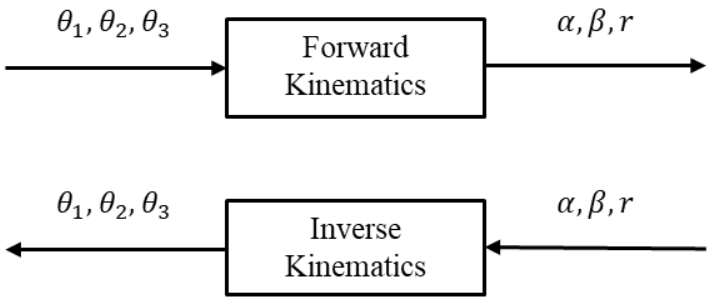


Figure 2. 1: Diagram of forward and inverse kinematic of RSR Parallel Robot

2.1.1 Forward Kinematics

In this section forward kinematics model of typical RSR parallel robot will be derived. The parameters of robot are as shown in Figures 2.2 and 2.3.

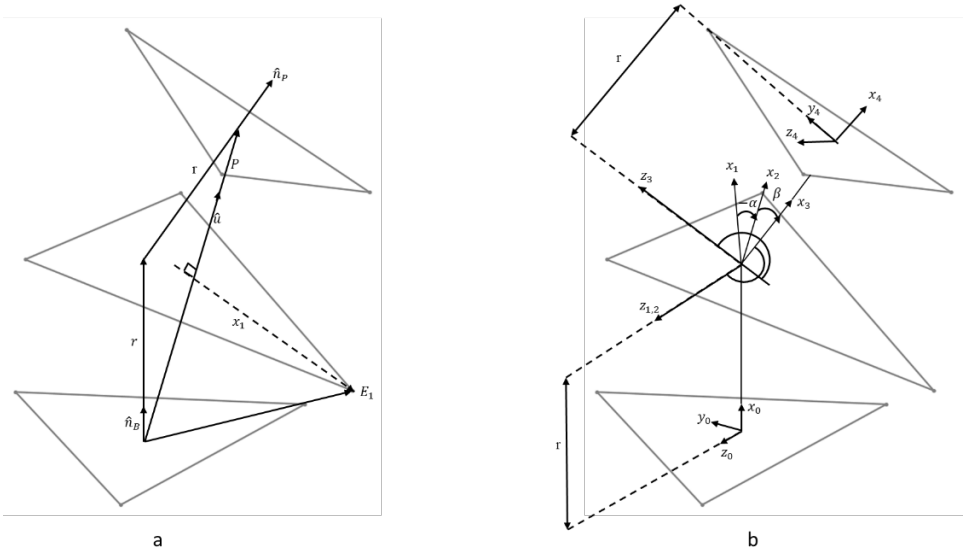


Figure 2. 2: Parameters of wrist mechanisms [30]

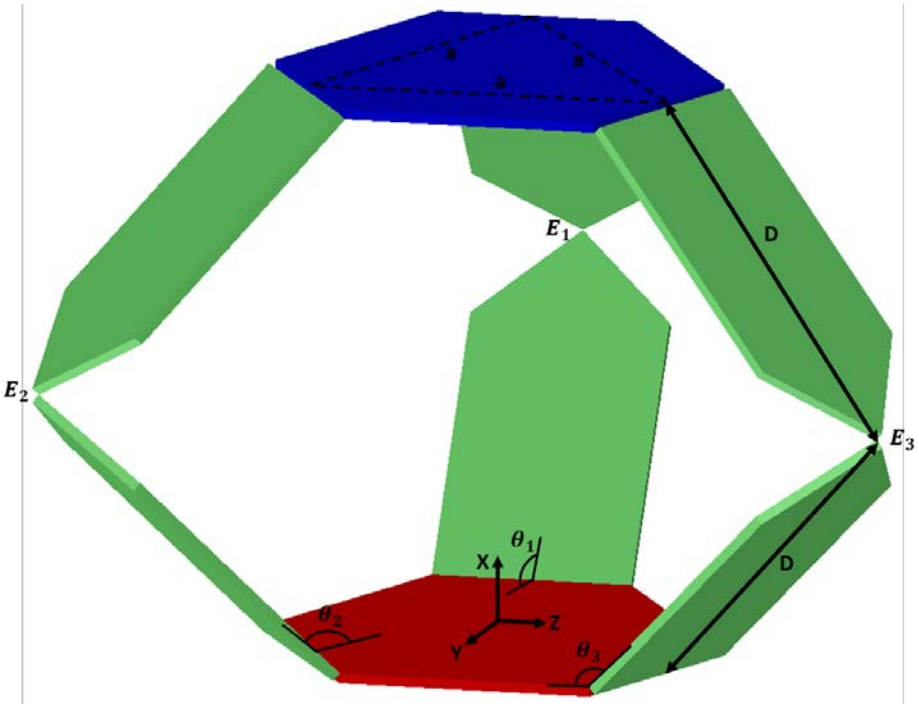


Figure 2. 3: General model of RSR robot

From the geometry base frame can be got as follows:

$$E_i = F_i + D_i l_i \tag{2.1}$$

Where F_i is the fixed location and D_i is the link length. l_i is the variable unit vector which is a function of θ_i.

$$F_i = \begin{bmatrix} F_{ix} \\ F_{iy} \\ F_{iz} \end{bmatrix} \quad \text{for } (i = 1,2,3) \tag{2.2}$$

$$l_i = \begin{bmatrix} \sin(\theta_i) \\ -\cos(\theta_i)\cos(\varphi_i) \\ -\cos(\theta_i)\sin(\varphi_i) \end{bmatrix} \tag{2.3}$$

Where $\varphi_1 = 180°$, $\varphi_2 = 300°$ and $\varphi_3 = 60°$

From the Figure [a] the unit vector directed from base origin to the top origin can be calculated by:

$$\hat{u} = \frac{(E_2 - E_1)x(E_3 - E_1)}{|(E_2 - E_1)x(E_3 - E_1)|} = \begin{bmatrix} \hat{u}_x \\ \hat{u}_y \\ \hat{u}_z \end{bmatrix} \tag{2.4}$$

The projection of r along $\hat{u}$ is $P/2$. Therefore:

$$r = \frac{P}{2\hat{n}_B \cdot \hat{u}} \tag{2.5}$$

Where $\hat{n}_B = [1 \quad 0 \quad 0]^T$ is the unit vector perpendicular to the base platform.

P can be calculated from the projection of E_i onto $\hat{u}$:

$$P = 2(E_1 \cdot \hat{u}) \tag{2.6}$$

$$\hat{n}_B r + \hat{n}_P r = \hat{u}P \tag{2.7}$$

$$\hat{n}_P = \frac{\hat{u}P}{r} - \hat{n}_B \tag{2.8}$$

First column of the transformation matrix is equal to n since the x direction is chosen perpendicular to the base platform. DH table of the wrist is obtained in Table 2.1.

Table 2. 1: DH table of the wrist

i	θ_i	d_i	α_i	a_i
1	0	0	0	r
2	α	0	0	0
3	β	0	$-\pi/2$	0
4	0	0	$\pi/2$	r

$$ {}^{B}_{P}T = \begin{bmatrix} c\alpha c\beta & -s\alpha & c\alpha s\beta & r(1+c\alpha c\beta) \\ s\alpha c\beta & c\alpha & s\alpha s\beta & rs\alpha c\beta \\ -s\beta & 0 & c\beta & -rs\beta \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9} $$

Therefore, first column of ${}^{B}_{P}T$ is:

$$ \hat{n}_P = \begin{bmatrix} \cos\alpha\cos\beta \\ \sin\alpha\cos\beta \\ -\sin\beta \end{bmatrix} \tag{2.10} $$

Finally, α, β and r can be calculated as follows:

$$ r = \frac{E_1\hat{u}}{\hat{u}_x} \tag{2.11} $$

$$ \beta = -\sin^{-1}(2\,\hat{u}_x\hat{u}_z) \tag{2.12} $$

$$ \alpha = -\sin^{-1}\left(\frac{2\hat{u}_x\hat{u}_y}{cos\beta}\right) \tag{2.13} $$

In addition to them, bending angle, ξ, can be calculated with following equation:

$$ \xi = \cos^{-1}\left(\frac{\hat{n}_P \cdot \hat{n}_B}{|\hat{n}_P||\hat{n}_B|}\right) \tag{2.14} $$

2.1.2 Inverse Kinematics

The vector locations of E_i are determined by the unknown angles θ_1, θ_2, and θ_3, while ensuring that vector x_i is perpendicular to vector $\hat{u}$ for $i = 1, 2, 3$.

$$x_i \cdot \hat{u} = 0 \tag{2.15}$$

$$\frac{P}{2}\hat{u} + x_i = E_i \tag{2.16}$$

By substituting Equation 2.15 into 2.16.

$$\left(E_i - \frac{P}{2}\hat{u}\right) \cdot \hat{u} = 0 \tag{2.17}$$

$$P = r(\hat{n}_B + \hat{n}_P) \tag{2.18}$$

$$P = r\begin{bmatrix} \hat{n}_{P_x} + 1 \\ \hat{n}_y \\ \hat{n}_z \end{bmatrix} \tag{2.19}$$

$$\hat{u} = \frac{P}{|P|} \tag{2.20}$$

$$|P| = r\sqrt{1 + 2n_{P_x} + n^2{}_{P_x} + n^2{}_{P_y} + n^2{}_{P_z}} \tag{2.21}$$

$$|P| = r\sqrt{2(1 + n_{P_x})} \tag{2.22}$$

Substituting Equations 2.21, 2.22, and 2.23 into equation 2.19 yields the following three constraint equations, which arise from the presence of three serial links:

$$V_1 c_1 + Y_1 s_1 + Z_1 = 0 \tag{2.23}$$

$$V_2 c_2 + Y_2 s_2 + Z_2 = 0 \tag{2.24}$$

$$V_3 c_3 + Y_3 s_3 + Z_3 = 0 \tag{2.25}$$

$$\theta_i = 2tan^{-1}\left(\frac{-Y_i\sqrt{{Y_i}^2 - {Z_i}^2 + {V_i}^2}}{Z_i - V_i}\right) \tag{2.26}$$

2.2 Kinematics of Origami Robots

In addition to evaluating the kinematic equations discussed in Section 2.1, this section delves into exploring the impact of origami constraints on the RSR parallel robot by employing three different pattern designs. Specifically, we examine the kinematic behavior of Waterbomb and Resch triangular tessellation (RTT) crease patterns. These closed chain patterns consist of 6 revolute joints that function similarly to a spherical joint for the parallel robot. Furthermore, we also investigate the kinematic behavior of the serially connected Resch triangular tessellation (RTT).

We employ the genetic algorithm optimization method, due to its high efficiency on finding global minima, to calculate the joint angles of patterns and derive the kinematics of the designed origami robots, which are used to constrain the parallel robot. The problem-solving steps are illustrated in Figure 2.4.

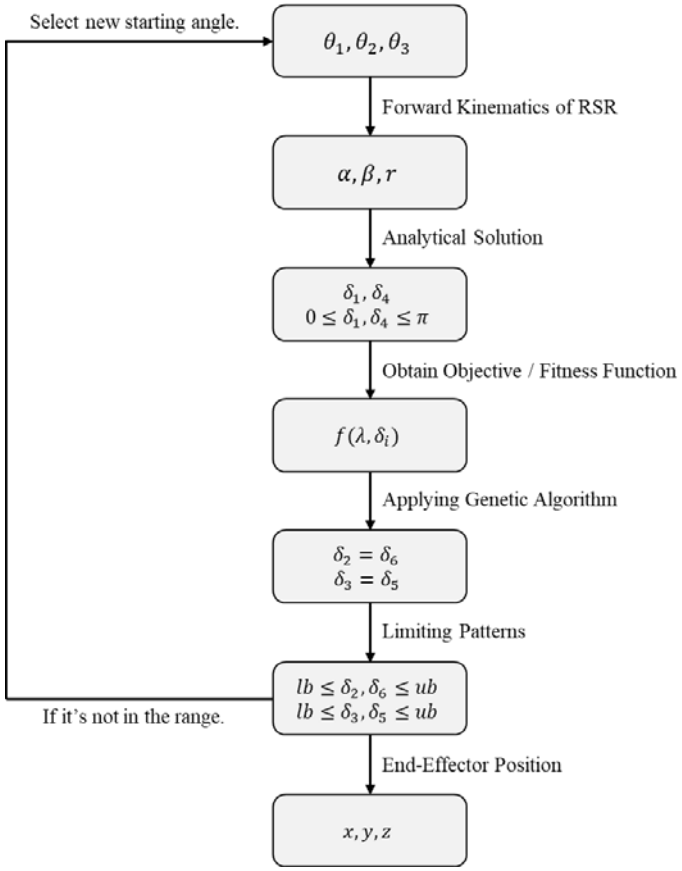


Figure 2. 4: Problem-solving steps of Origami Kinematics

2.2.1 Waterbomb Pattern

The waterbomb mechanism, as shown in Figure 2.5, can be described as the convergence of three folding lines at a central point, forming an intersection. In this case, the design angle, represented by λ, is set to $45°$. Varying the design angle will impact the range of motion of the spherical joint and indirectly affect the workspace of the origami robot. After examining the mechanism on SolidWorks it's seen that optimum design angle is $45°$ for waterbomb.

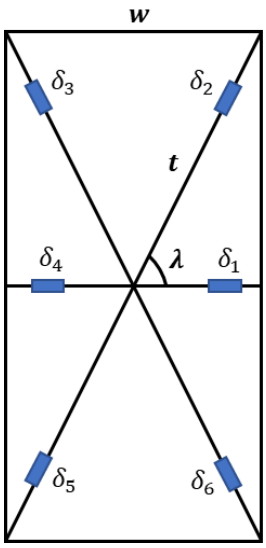


Figure 2. 5: Waterbomb pattern with revolute joint placed at the creases.

Where δ_i refers to waterbomb joint angles for $i = 1,2,\ldots,6$

In the analysis of the waterbomb mechanism, an analytical approach is employed to calculate the 1st and 4th angles. This approach utilizes geometric perspectives to determine these angles. By deriving these angles, we can further understand the behavior and characteristics of the waterbomb mechanism and develop an optimization algorithm using Genetic algorithm (GA) in MATLAB Software by considering the principles of rigid origami where all faces of the structure remain flat and undeformed.

The position of End-Effector is calculated in Section 2.1. That's why P_i, P_{i1} and P_{i2} are known for each links. Only unknowns are E_{i1} and E_{i2}.

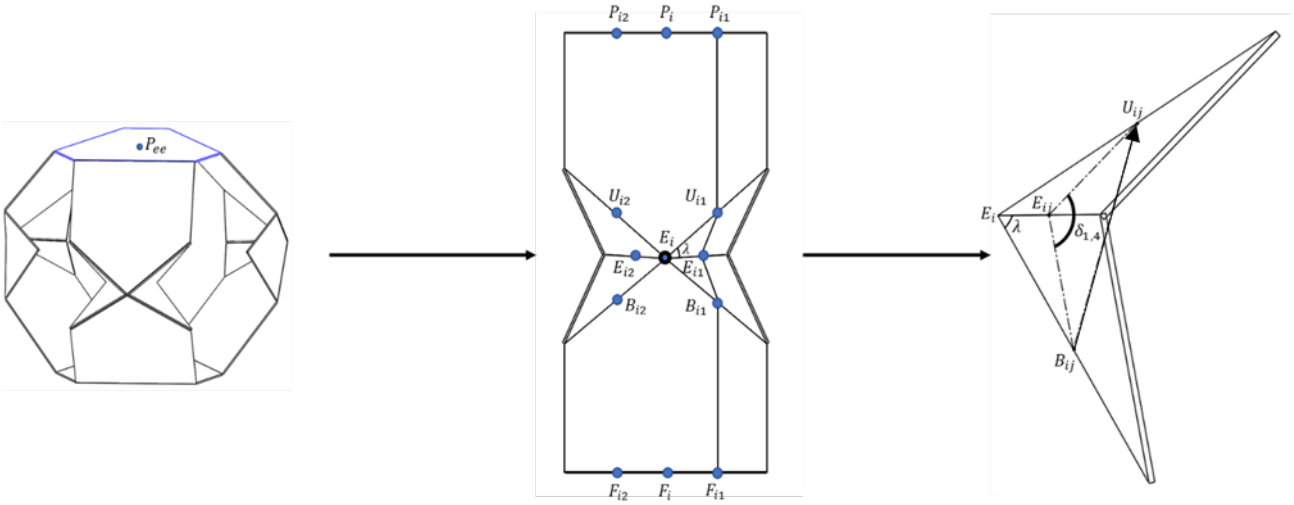


Figure 2. 6: Geometric solution of waterbomb angle 1 and 4

In case where $\lambda = 45°$ the vectors, $\left|\overrightarrow{UE}\right|$ and $\left|\overrightarrow{EB}\right|$ are:

$$\left|\overrightarrow{UE}\right|_{ij} = \left|\overrightarrow{EB}\right|_{ij} = \frac{w}{2} \tag{2.27}$$

Where w is the wide of the link. i refers to the corresponding leg. j represents the right and left half of the leg as shown in Figure 2.6. 1 is for the right half and 2 for the left half of the link. Applying law of cosines onto the BEU triangle results in finding waterbomb angles δ_1 and δ_4.

$$\delta_1 = \cos^{-1}\left(1 - \frac{\left|\overrightarrow{BU}_{i1}\right|^2}{2\left|\overrightarrow{UE}_{i1}\right|^2}\right),\ \delta_4 = \cos^{-1}\left(1 - \frac{\left|\overrightarrow{BU}_{i2}\right|^2}{2\left|\overrightarrow{UE}_{i2}\right|^2}\right) \tag{2.28}$$

Where $0 \leq \delta_{1,4} \leq \pi$

Notice that $\left(\overrightarrow{BU}_{ij}\right)_x \geq 0$should be always satisfied.

The DH table can be formulated using waterbomb angles of, δ_1 and δ_4, each representing one degree of freedom, resulting in a total of six free variables, while the closed loop structure of the waterbomb pattern connects the last joint to the first joint, completing the loop and resulting in three degrees of freedom at the end.

Table 2. 2: DH table for the loop-closure system

i	a_i	α_i	d_i	θ_i
1	0	$\pi - \lambda$	0	δ_2
2	0	$\pi + 2\lambda$	0	δ_3
3	0	$\pi - \lambda$	0	δ_4
4	0	$\pi - \lambda$	0	δ_5
5	0	$\pi + 2\lambda$	0	δ_6
6	0	$\pi - \lambda$	0	δ_1

The loop-closure equation is:

$$ {}^{2}_{1}T\ {}^{3}_{2}T\ {}^{4}_{3}T\ {}^{5}_{4}T\ {}^{6}_{5}T\ {}^{1}_{6}T\ = P(\delta) \tag{2.29} $$

Where $\delta = [\delta_1 \quad \cdots \quad \delta_6]^T$

We didn't include any length parameter into the DH table which means our only consideration is the rotation. The closed chain structure of the waterbomb pattern creates a loop closure equation that must satisfy the identity matrix $P(\delta) - I = 0$.

$$ f(\delta) = [P_{3x3}(\delta) - I_{3x3}] \tag{2.30} $$

The overall system is solved by using Genetic algorithm optimization method by minimizing the objective function obtained.

$$ F(\delta) = [f_1(\delta) f_2(\delta) \cdots f_9(\delta)] \tag{2.31} $$

$$ \min_{\delta} ||F\ (\delta)||^2 = \min_{\delta} \left(\sum_{k=1}^{9} f_k(\delta)^2 \right) \tag{2.32} $$

Achievable lower and upper boundaries for waterbomb angles $\delta_{2,3,5,6}$ can defined as $[0, \pi]$. Extra limitations can be given according to the specific application.

Because we are dealing with parallel robot, midplane acts as symmetric plane which creates following relation that $\delta_2 = \delta_6$ and $\delta_3 = \delta_5$.

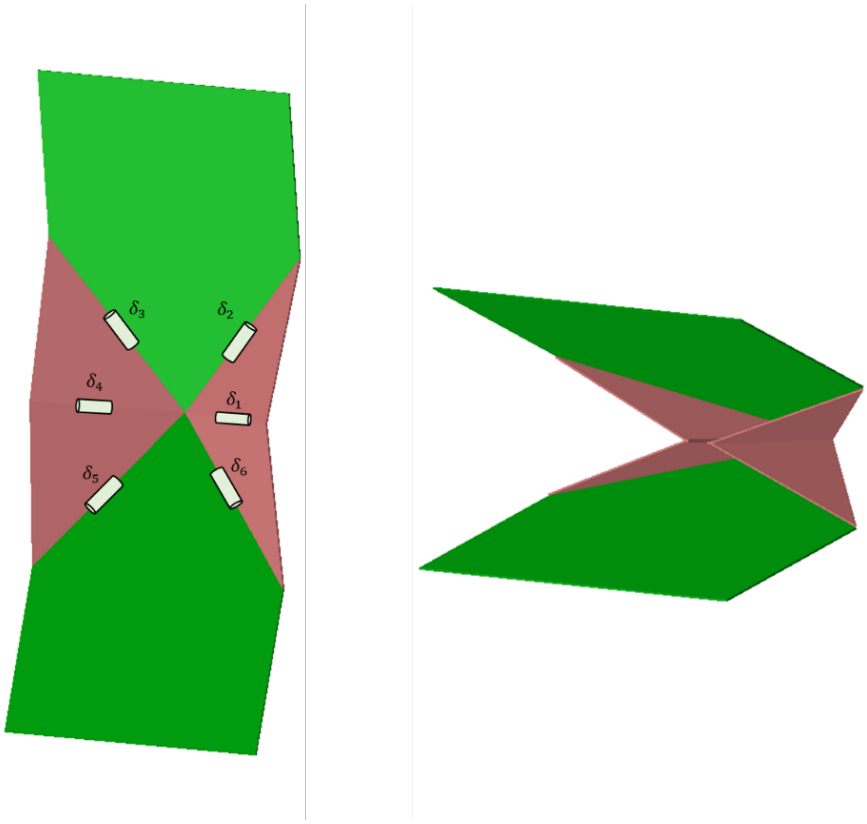


Figure 2. 7: Open (left) and close (right) states of Waterbomb mechanism

Overall process can be expressed as follows:

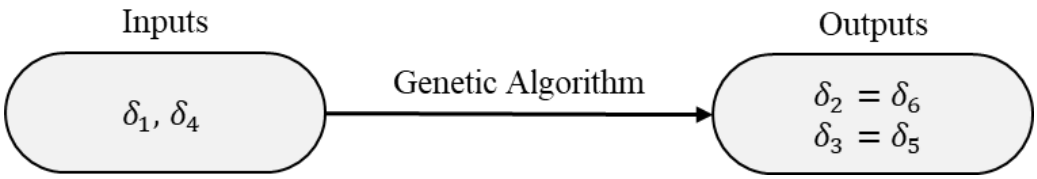


Figure 2. 8: Overall optimization process

2.2.2 Resch Triangular Tessellation (RTT)

On the other hand, the Resch Triangular Tessellation (RTT) exhibits a construction that is very similar to the waterbomb pattern. Hence a parallel methodology will be employed in working with the RTT. The RTT pattern is illustrated in Figure 2.9

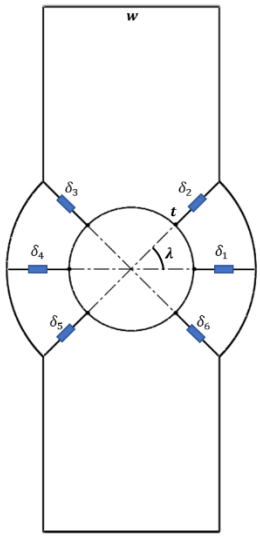


Figure 2. 9: RTT pattern with conventional revolute joints on

In contrast to the waterbomb pattern, the Resch Triangular Tessellation (RTT) offers a notable advantage by enabling a wide range of motion capabilities through varying design angles (λ). The design angle can be adjusted sensibly within the range of 45° to 89°, providing flexibility and adaptability in the system.

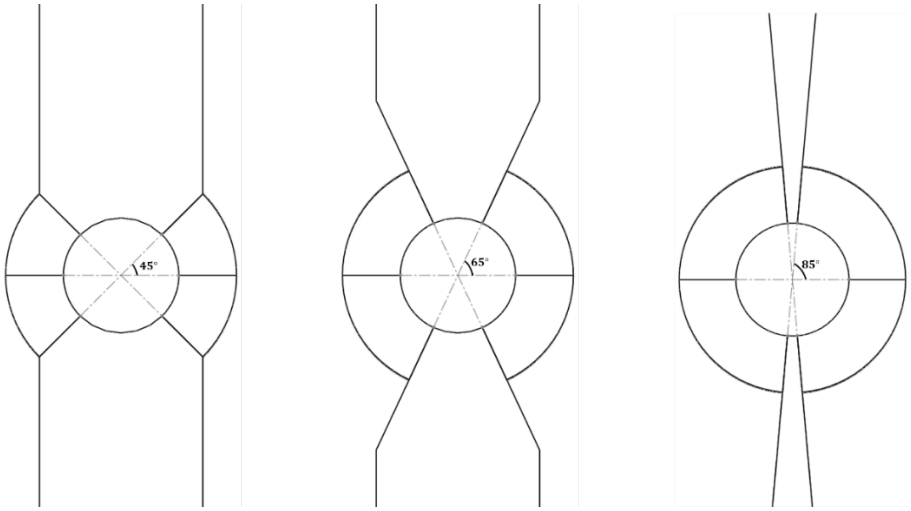


Figure 2. 10: Appearance of RTT mechanism under different λ

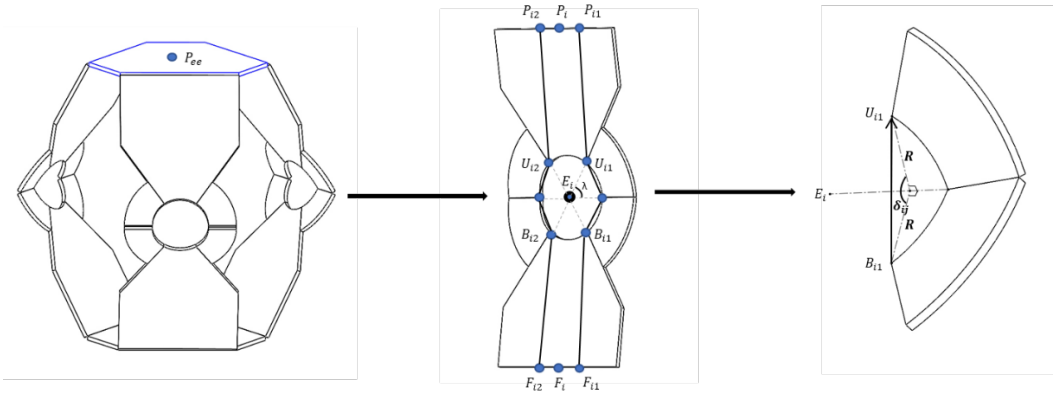


Figure 2. 11: Geometric solution of RTT mechanism

To obtain δ_1 and δ_4, cosine theorem will be applied to BEU triangle.

$$\left(\overrightarrow{BU}\right)_{ij} = U_{ij} - B_{ij} \tag{2.33}$$

$$\delta_1 = \cos^{-1}\left(2R^2 - \left|\overrightarrow{BU}_{i1}\right|^2\right),\ \delta_4 = \cos^{-1}\left(2R^2 - \left|\overrightarrow{BU}_{i2}\right|^2\right) \tag{2.34}$$

Where $0 \leq \delta_{1,4} \leq \pi$

In addition, $\left(\overrightarrow{BU}_{ij}\right)_x \geq 0$ must be satisfied.

Lastly, genetic algorithm can be employed to find the rest of RTT angles in the achievable range of $[\pi/2, 3\pi/2]$. In general, the achievable range of motion for robots is limited by rotational restrictions, which are influenced by the materials used in the joints. The actual range of motion is determined by the rotation angle γ, which defines the maximum extent to which the robot can rotate inside or outside. The updated form of achievable range is:

$$\pi - \gamma \leq \delta_{1,4} \leq \pi + \gamma \tag{2.35}$$

Where $0 \leq \gamma \leq \pi/2$

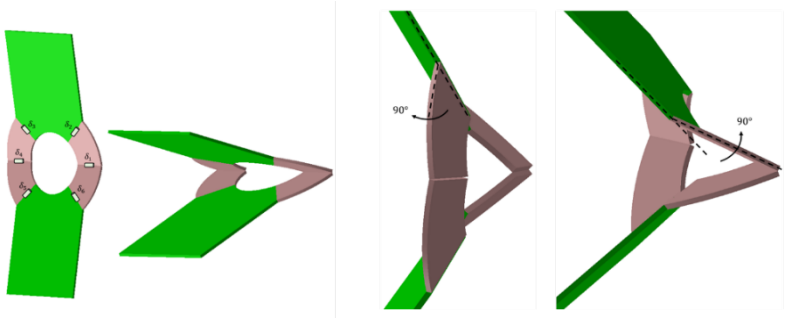


Figure 2. 12: Various positions of RTT mechanism

2.2.3 Serial Resch Triangular Tessellation (Serial RTT)

Despite the wide range of design angles offered by the closed chain Resch Triangular Tessellation (RTT) pattern, it is important to note that adjusting the design angle to 90 degrees is not feasible. In order to achieve this specific angle, we will connect the RTT pattern in a serial manner, as depicted in Figure 2.13.

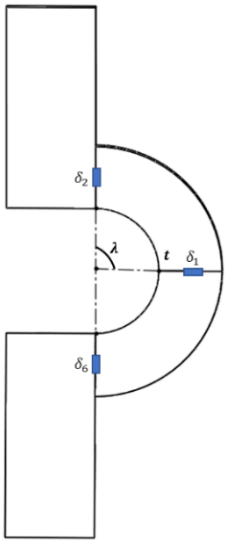


Figure 2. 13: Serial RTT pattern

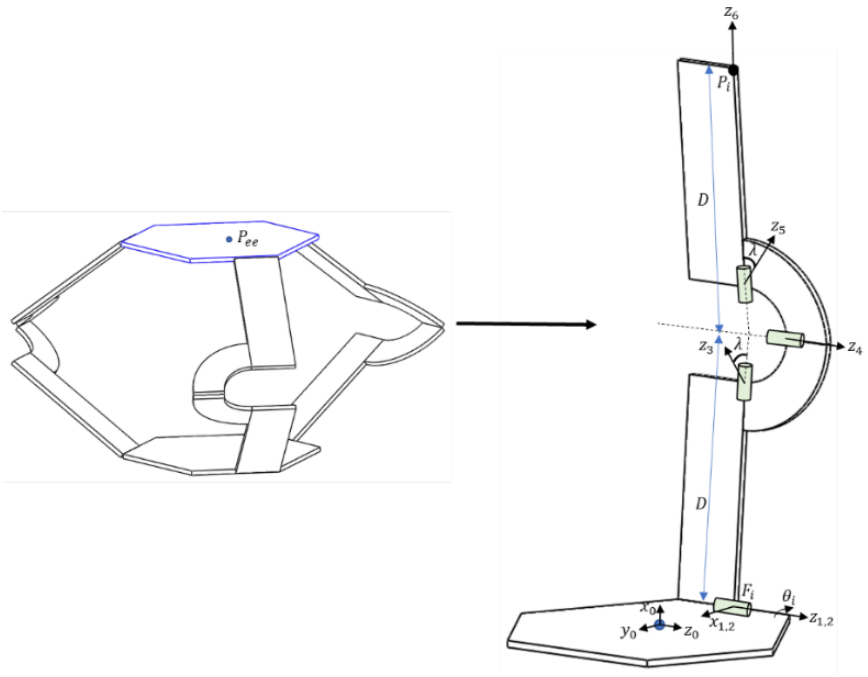


Figure 2. 14: Serial RTT robot and parameter of it to achieve DH table

By utilizing a serially connected RTT mechanism, it becomes feasible to achieve design angles ranging from $45°$ to $90°$ in a practical and logical manner. This configuration allows for increased flexibility and adaptability in adjusting the design angles within this specified range.

Using the cosine theorem allows us to determine the value of δ_1, thereby leaving us with only one unknown variable since δ_2 is equal to δ_6.

Next, we can determine the DH table for serial connected RTT mechanism, following the order as depicted in Figure 2.14.

Table 2. 3: DH table of Serial RTT mechanism

i	a_i	α_i	d_i	θ_i
1	0	φ_i	0	$\pi/2$
2	0	0	0	θ_i
3	0	$\pi/2$	D	0
4	0	$\pi/2-\lambda$	0	δ_6
5	0	$\pi+\lambda$	0	δ_1
6	0	λ	0	δ_2
7	0	$\pi/2-\lambda$	D	0

Where φ represents the rotation vector, which takes values of 180°, 300°, and 60° for each respective link. Additionally, θ_i denotes the input angle that varies between 90 and 180 degrees.

$$T = {}^{1}_{0}T\ {}^{2}_{1}T\ {}^{3}_{2}T\ {}^{4}_{3}T\ {}^{5}_{4}T\ {}^{6}_{5}T\ {}^{7}_{6}T\ = \begin{bmatrix} \cdots & \cdots & \cdots & P_x \\ \cdots & \cdots & \cdots & P_y \\ \cdots & \cdots & \cdots & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = P(\delta) \tag{2.36}$$

From the figure we can determine the P_i after applying forward kinematics and then subtract it from F_i to obtain target position vector $\left(P_{target}\right)$. The serial mechanism must satisfy $P(\delta) - P_{target} = 0$.

After all the objective function is expressed as follows:

$$f(\delta) = \left[P_{3x1}(\delta) - P_{target,3x1}\right] \tag{2.37}$$

At the end genetic algorithm can be applied to the objective function in the range of $[\pi/2, 3\pi/2]$ to obtain δ_2 and δ_6.

3 WORKSPACE ANALYSIS

In the previous section, we provided a detailed kinematic explanation of three origami robots: the waterbomb mechanism, RTT mechanism, and serial RTT mechanism. Our design process initially focused on an origami robot utilizing the waterbomb mechanism, which achieved an optimal design angle of 45 degrees. This angle was found to provide the best workspace due to specific pattern conditions for the waterbomb mechanism. However, if the λ angle exceeds 45 degrees, contact occurs within the triangle of the waterbomb mechanism, limiting its range of motion.

On the other hand, robots utilizing the RTT and serial RTT mechanisms can achieve higher λ angles without any inter-mechanism contact. The RTT mechanism was explored to enhance the workspace of the parallel robot composed of the waterbomb mechanism. Furthermore, the serial RTT mechanism was improved to have a larger workspace than its predecessor by eliminating any chain within each leg.

The input parameters of the robots Figure 2.3 used in this section are shown in Table 3.1.

Table 3. 1: Geometrical properties of analyzed origami robots

a	D	λ	$\delta_{1,4}$	$\delta_{2,3}$
$35\ mm$	$60\ mm$	$45°$	$10° \leq \delta_{1,4} \leq 170°$	No Limit

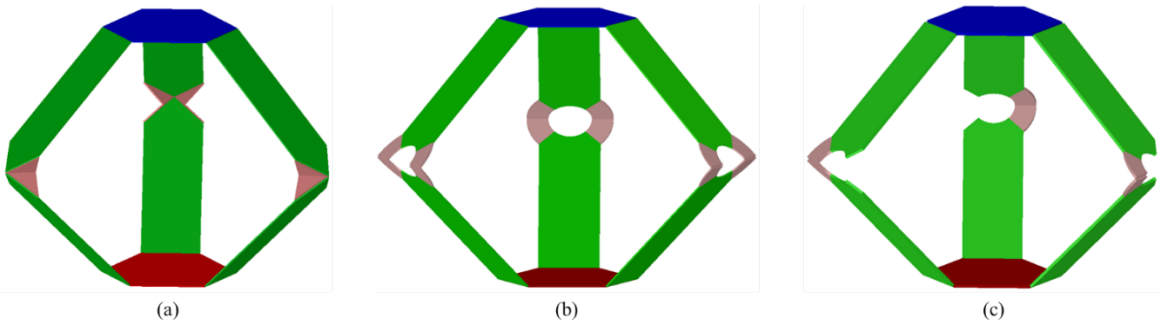


Figure 3. 1: Origami robot designed with waterbomb pattern (a), RTT pattern (b) and, Serial RTT pattern (c) for the given geometrical properties in Table 3.1

To compare the differences in workspaces among the three mechanisms, it is crucial to observe them operating at the same design angle ($\lambda = 45°$). Figure 3.2 provides a visual representation of this comparison.

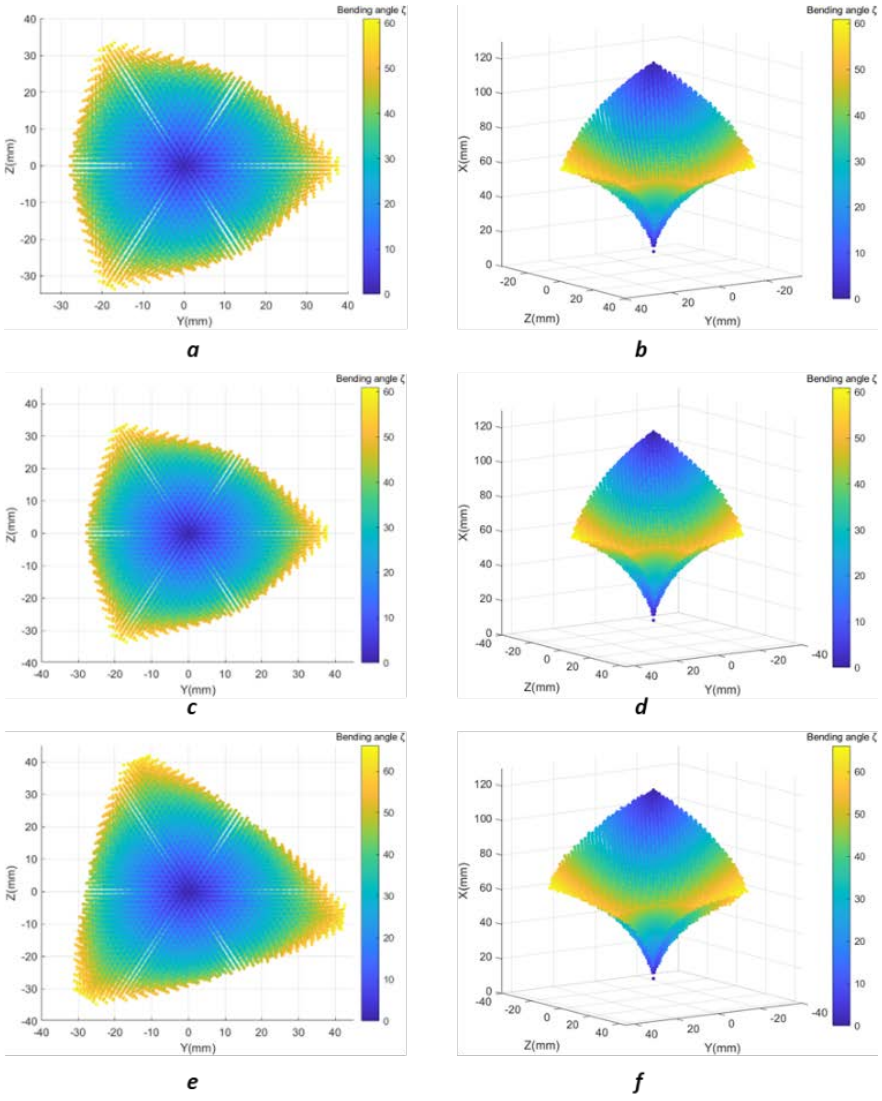


Figure 3. 2: Top and isometric views of Workspace of Origami Robots in MATLAB: Waterbomb (a, b), RTT (c, d), and Serial RTT (e, f)

The results displayed in Figure 3.3 were obtained through an analytical solution conducted in MATLAB. A step input angle of 2.5 degrees was selected for the analysis, considering both time consumption and smooth visualization. To validate the results, the data points obtained from MATLAB were imported as a point cloud file into SolidWorks. This enabled a clear visualization of the workspace for each origami robot, alongside their respective robot bodies. By doing so, we were able to validate the accuracy and correctness of our kinematic formulation.

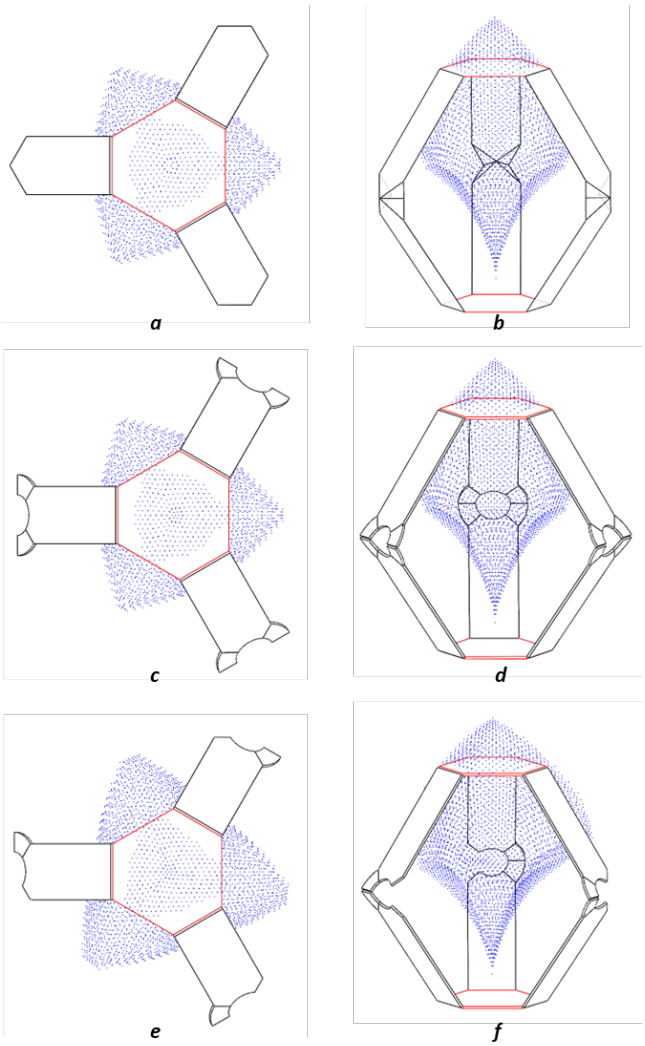


Figure 3. 3: Top and isometric views of Workspace of Origami Robots visualized in SolidWorks: Waterbomb (a, b), RTT (c, d), and Serial RTT (e, f)

Increasing the design angle λ is necessary to expand the workspace. However, it is important to note that the RTT and serial RTT origami robots will exhibit different workspaces even when operating at the same design angle. This discrepancy arises from the fact that the RTT robot eliminates any two-chain configuration within its closed loop, as these chains tend to restrict the robot's motion.

The analysis of the workspaces for these two mechanisms, considering various design angles λ, was performed using MATLAB. The results were visualized from different perspectives to provide a comprehensive understanding. The impact of λ on the workspace can be observed in Figure 3.4 and 3.5

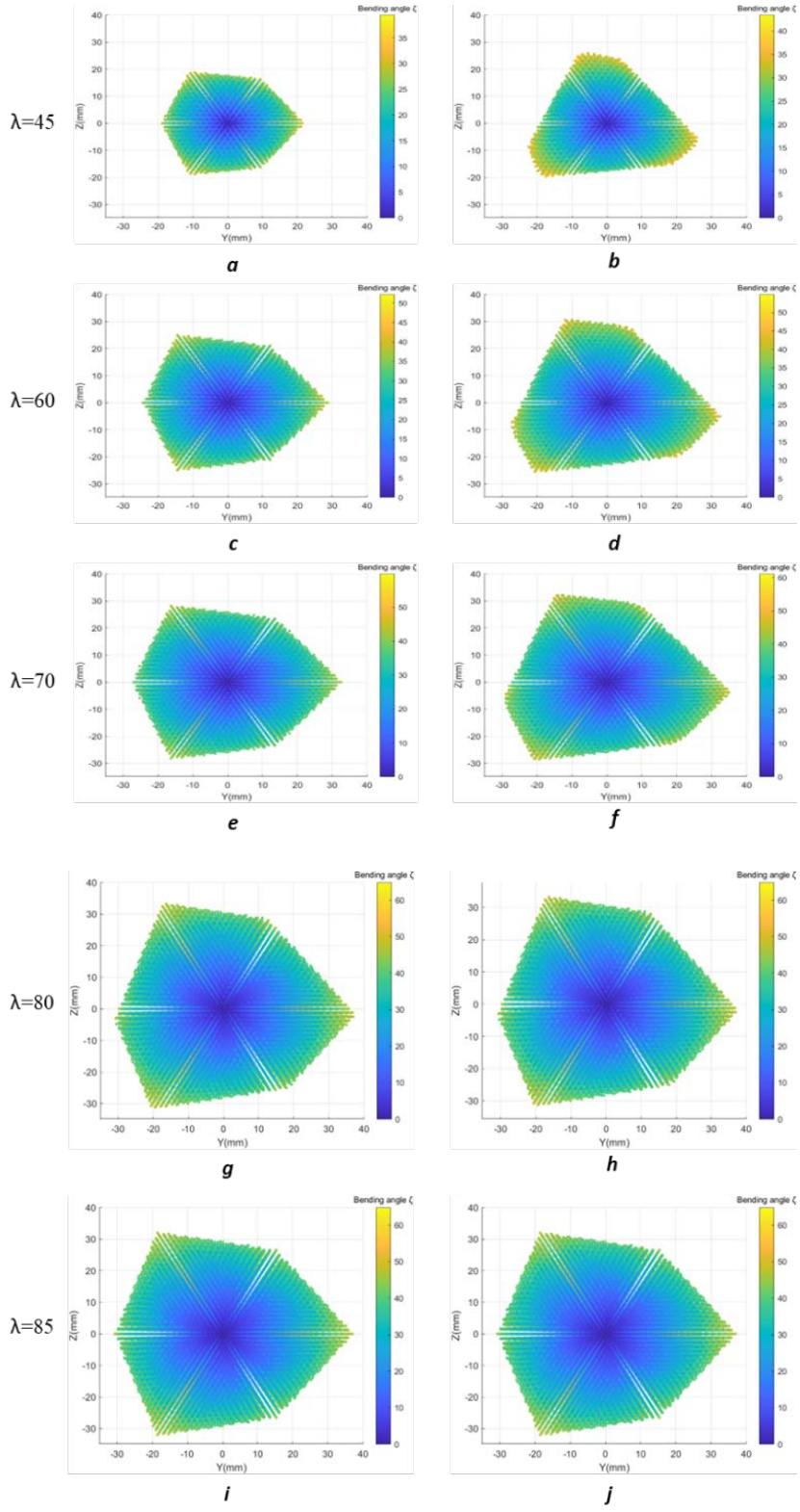


Figure 3. 4: Top views of Workspace of Origami Robots at different design angles: RTT (a, c, e, g, i) and Serial RTT (b, d, f, h, j)

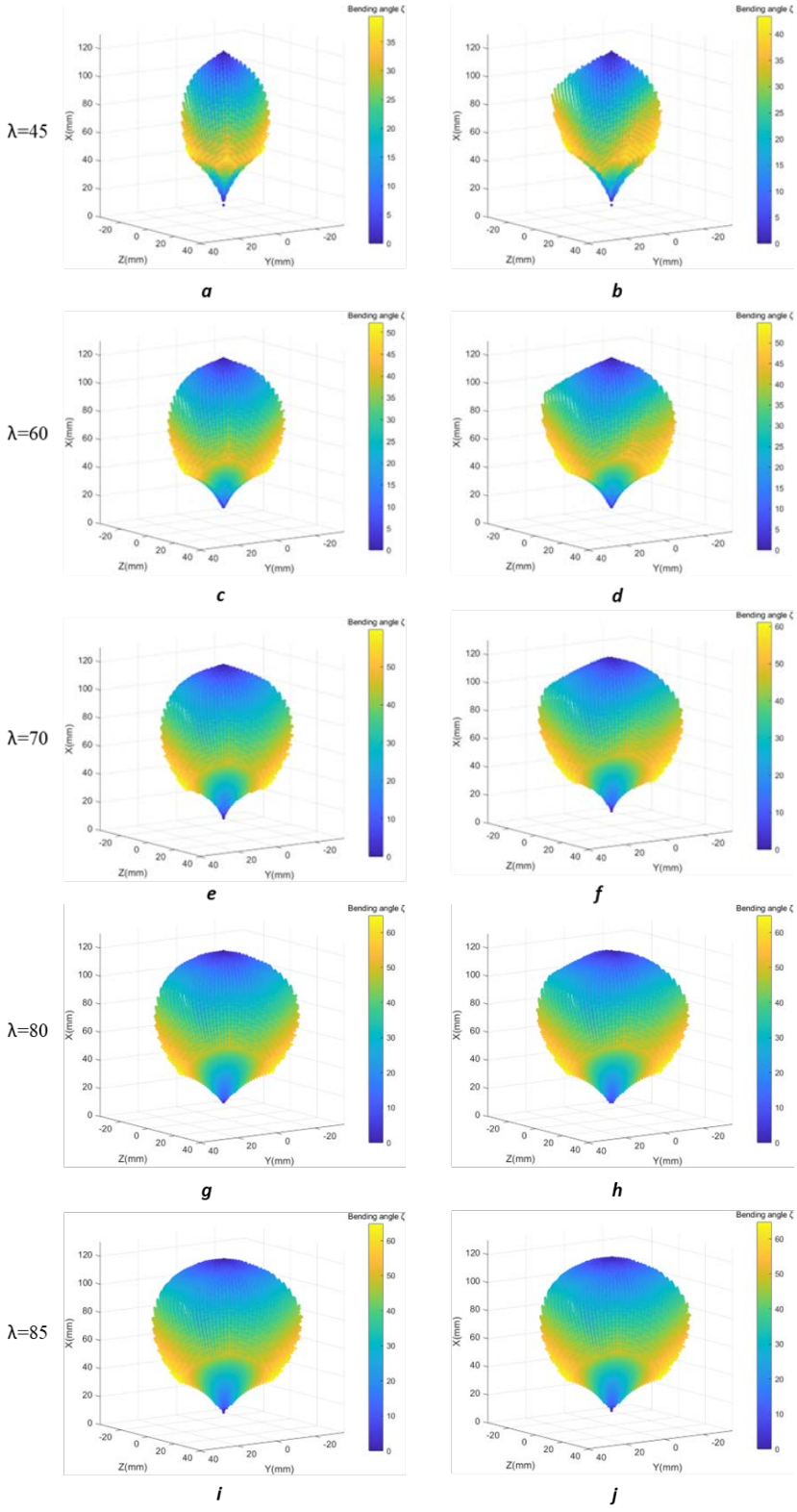


Figure 3. 5: Isometric views of Workspace of Origami Robots at different design angles: RTT (a, c, e, g, i) and Serial RTT (b, d, f, h, j)

The workspace of the origami robots was obtained from MATLAB by using GA algorithm to solve closed loop equation given in kinematic part. Every solution cannot be in the domain due to joint restriction which is caused by material flexibility. Therefore, there is a decision loop in the solution algorithm to decide whether the results are in the solution domain or not. All of them take considerable time to solve. It is especially caused by GA during the analytical calculation in MATLAB. To overcome this waste of time and decrease it, a different algorithm than GA was decided to be composed when closed loop equations are solved.

The new algorithm has a principle Divide and Conquer method which is given in [40], so it is called that name. The working methodology of it can be defined basically dividing the solution range and finding solution which has minimum error at that range. After that, the solution range is updated around the minimum error solution. If error is lower than desired, the iteration will be stopped. In that way, solution will not be sought unnecessary area in the range and solution time will be reduced. In Table 3.2, time gain with Divide and Conquer method than GA was shown for each origami robot design.

Table 3. 2: Comparison between GA and Divide and Conquer Methods

Properties	***Waterbomb***	***RTT***	***Serial RTT***
$\boldsymbol{\lambda}$	$45°$	$45°$	$45°$
Sol. Step Angle	$10°$	$10°$	$10°$
$\boldsymbol{\delta_2}$ & $\boldsymbol{\delta_6}$	$[0°, 180°]$	$[0°, 180°]$	$[0°, 180°]$
$\boldsymbol{\delta_3}$ & $\boldsymbol{\delta_5}$	$[0°, 180°]$	$[0°, 180°]$	$[0°, 180°]$
$\boldsymbol{\delta_1}$ & $\boldsymbol{\delta_4}$	$[10°, 170°]$	$[10°, 170°]$	$[10°, 170°]$
Volume	$111157\ \text{mm}^3$	$111157\ \text{mm}^3$	$132249\ \text{mm}^3$
# of Reached Points	293	293	350
Bending angle, ξ	$58.4619°$	$58.4619°$	$58.4619°$
Sol. time with GA	$33.2286\ s$	$35.6669\ s$	$51.5663\ s$
Sol. time with Divide and Conquer	$8.6373\ s$	$7.422\ s$	$3.6475\ s$
Time Consuming	$\boldsymbol{3.84\ times}$	$\boldsymbol{4.81\ times}$	$\boldsymbol{14.14\ times}$

After generating MATLAB codes to analyze the workspace of the robots, it is essential to validate the correctness of them, as presented in Appendix A. To achieve this, a model of the robot is constructed in SIMULINK, as shown in Figure 3.6. The robot models are transferred to SIMULINK using the SimMechanics tool in SolidWorks. Additionally, SIMULINK software enables the application of feedback control to the robot system, allowing for simulation. By comparing the results obtained from the MATLAB codes and the SIMULINK simulations, we can determine whether the codes are accurate and reliable. This validation process ensures confidence in the accuracy of the workspace analysis conducted in the study. Details of the block are shown in Appendix C.

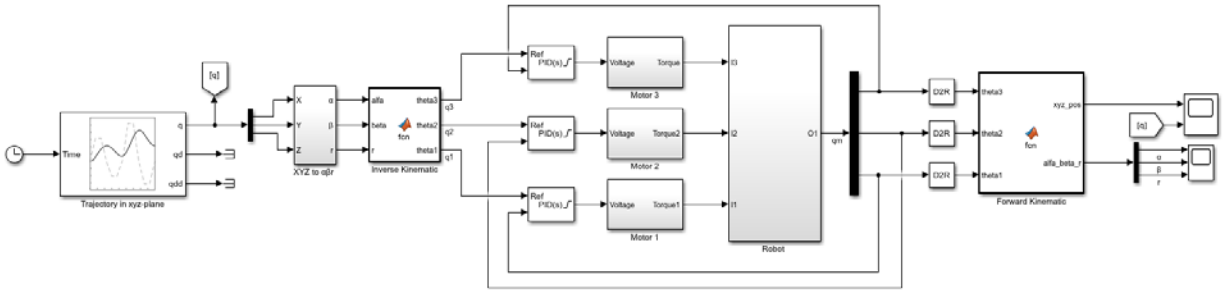


Figure 3. 6: Block diagram of the Origami Robots in SIMULINK

4 FINITE ELEMENT ANALYSIS (FEA)

In this section of our project, we conducted a finite element analysis (FEA) using the static structural module in the ANSYS program to analyze the behavior of the origami robots. The primary goal of this analysis was to gain insights into how the robots would respond under large deflection conditions before proceeding to the manufacturing phase. By employing FEA, we aimed to gather valuable information that would aid us in selecting suitable materials and determining optimal parameters for the robots.

Through the FEA, we not only obtained deformation and stress data but also crucial information such as the locations (x, y, z) and orientation angles (α, β) of the robots. These results will be compared with the kinematic formulation implemented in MATLAB, allowing us to validate our simulations and ensure the accuracy of our models.

- The analysis setup details can be found in Appendix B.

Our analysis began with careful material selection. In order to achieve significant deflections, we opted for Thermoplastic Polyurethane (TPU) due to its exceptional flexibility. Additionally, we considered the utilization of multi-material 3D printing. This approach allows us to customize the rigidity of the links using Polylactic Acid (PLA) while ensuring the joints remain flexible with TPU. To investigate the impact of these two states, we initially applied them to a revolute joint. Subsequently, we extended the evaluation to the entire robot system.

The determination of joint locations will be based on identifying areas where the thickness is reduced. It is crucial to adjust parameters accordingly to avoid any potential impact between bodies or links. To establish the relationship between the total link thickness (T) and the joint width (JW), we will follow the following formulation, considering the joint thickness (JT) determined by the specific 3D printing machine utilized:

$$JW \geq 0.5\pi T \tag{4.1}$$

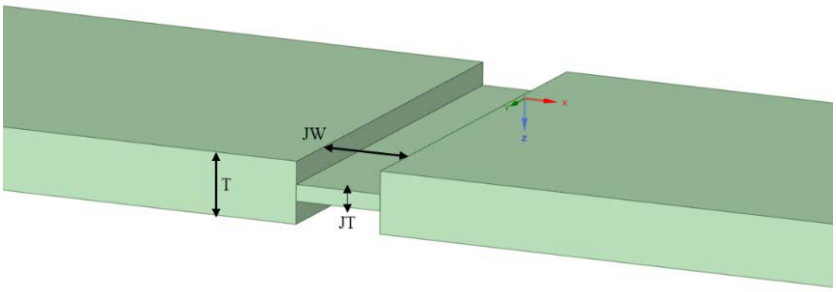


Figure 4. 1: Parameters of Revolute joints

The effect of using PLA for links and TPU for joints and TPU for the entire structure is illustrated in the Figure 4.2.

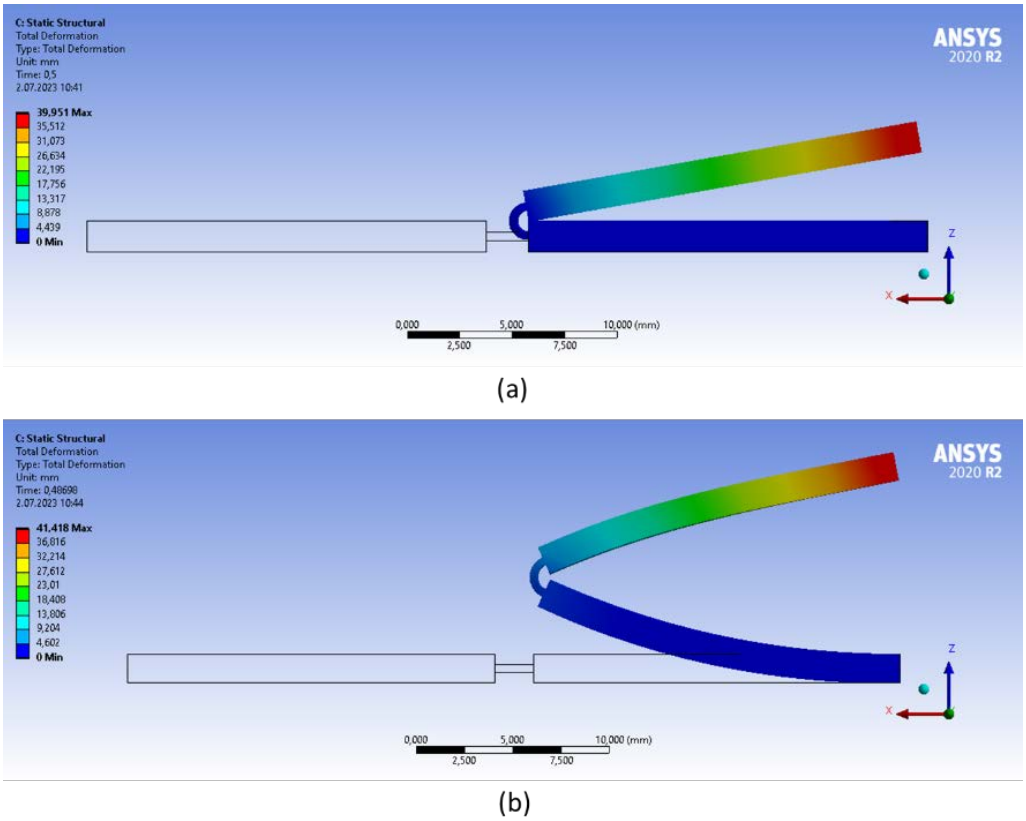


Figure 4. 2: Static structural Behavior of the body: TPU for joint and PLA for links (a) & TPU for entire body (b)

The same procedure is applied to the waterbomb origami robot and deformations are observed as shown in Figure 4.3.

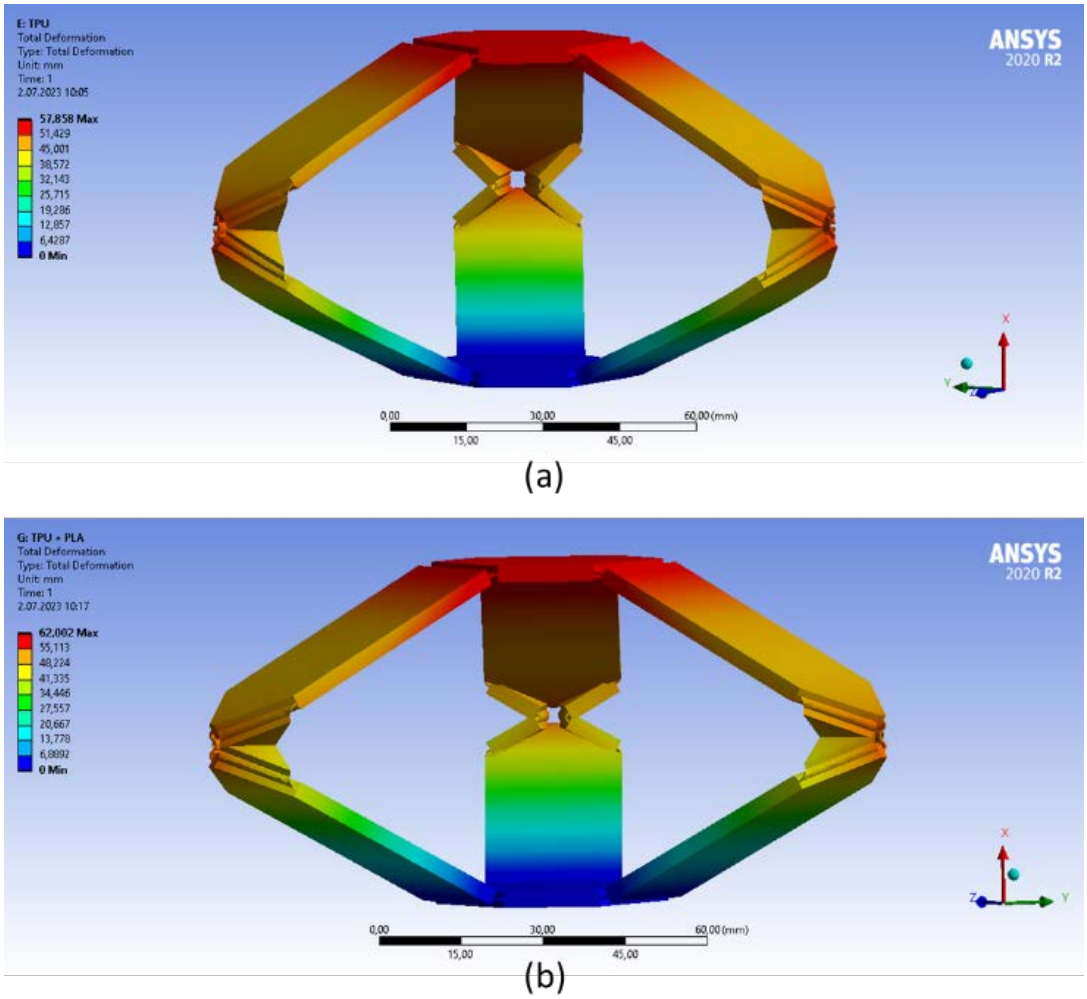
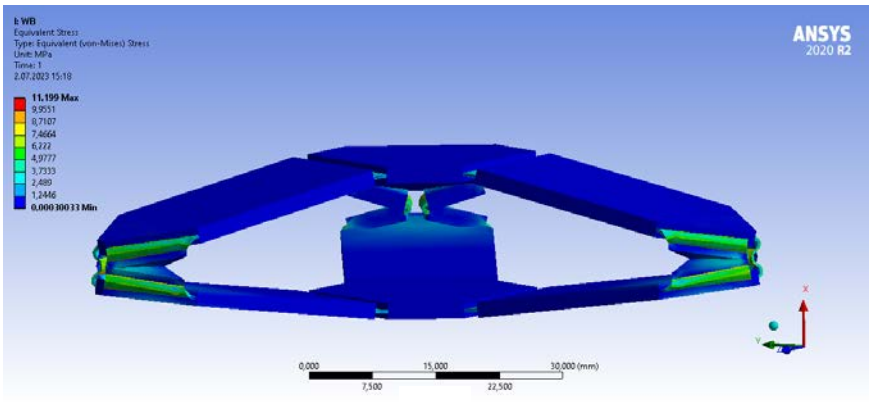
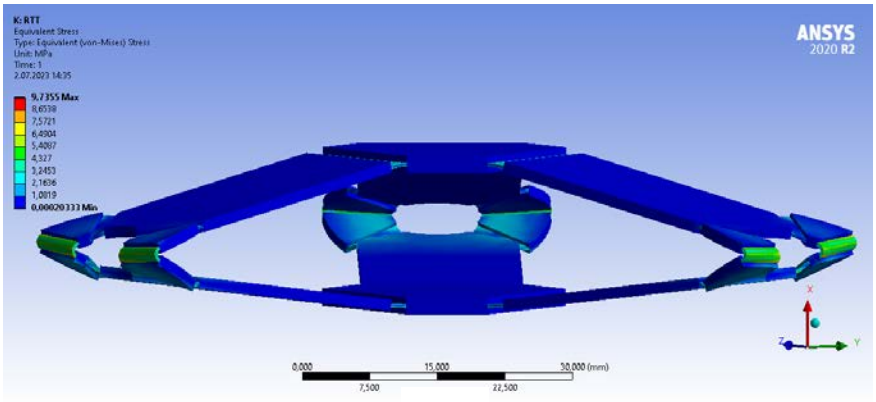


Figure 4. 3: Static structural Behavior of the Waterbomb Origami Robot: TPU for entire body (a) & TPU for joint and PLA for links (b)

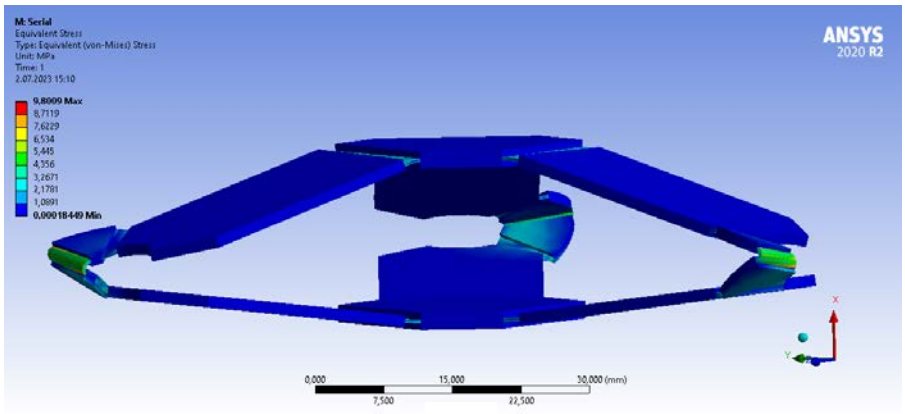
To investigate the geometrical differences and their impact on stress distribution, identical input angles and materials (TPU for joints and PLA for bodies) were applied to three origami robots: the Waterbomb, RTT, and Serial RTT. The resulting stress distribution for each robot is illustrated in Figure 4.4.



(a)



(b)



(c)

Figure 4. 4: Equivalent Stress distribution of Waterbomb, RTT and Serial RTT respectively

Table 4. 1: Comparison of stress occurring under the same conditions

Waterbomb	RTT	Serial RTT
11.199 MPa	9.7355 MPa	9.8009 MPa

5 MECHANICAL DESIGN

To fully leverage the advantages offered by the origami method, we aimed to fabricate the robots in a single piece using 3D printing technology, rather than employing complex composite techniques. This decision was motivated by the desire to streamline the manufacturing process and minimize potential complications. To kickstart this approach, we initiated the production by designing a waterbomb pattern in SolidWorks, utilizing one of the most widely used materials in 3D printing: PLA (Polylactic Acid). PLA is widely chosen for its ease of use and accessibility.

However, the results did not meet our expectations due to the limitations of PLA, specifically its high strength and low flexibility. These properties hindered the desired range of rotations required for the joints on the pattern. Consequently, the PLA material experienced issues with cracking, as depicted in Figure 5.1.

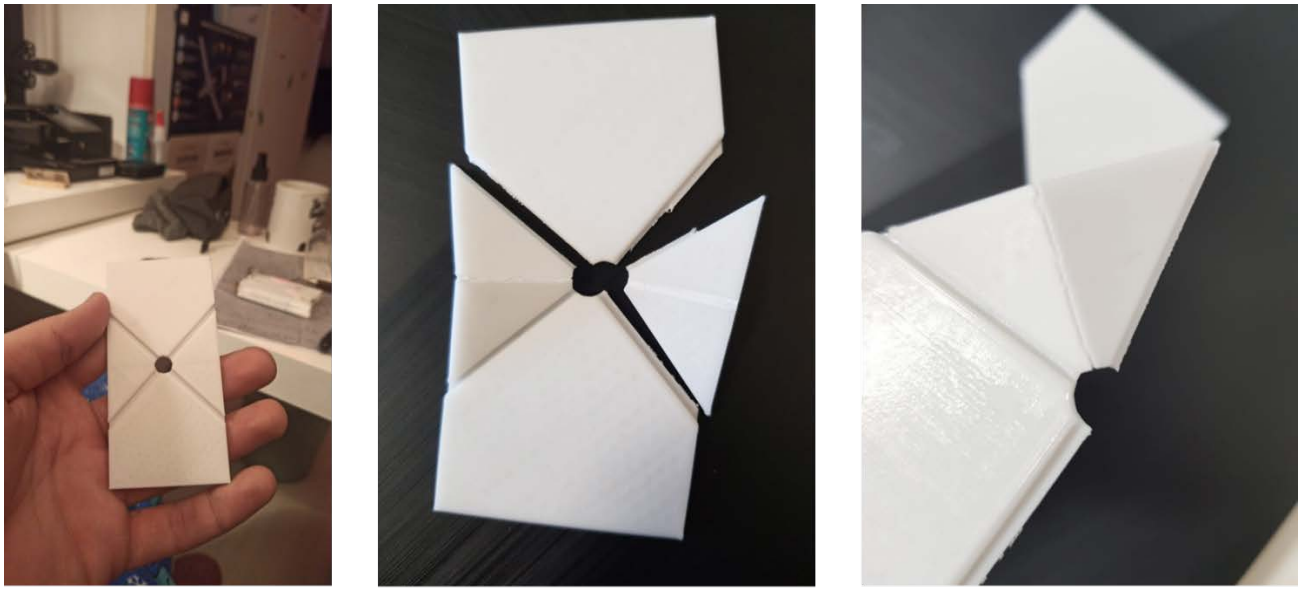


Figure 5. 1: Waterbomb pattern: before failure (a), after failure (b, c)

Considering the challenges encountered with PLA, we turned our attention to exploring more flexible materials within the industry. This search led us to TPU (Thermoplastic Polyurethane), a material known for its high flexibility and suitable behavior for accommodating large deflection motions. Subsequently, we produced a waterbomb origami robot using TPU, as depicted in Figure 5.2.

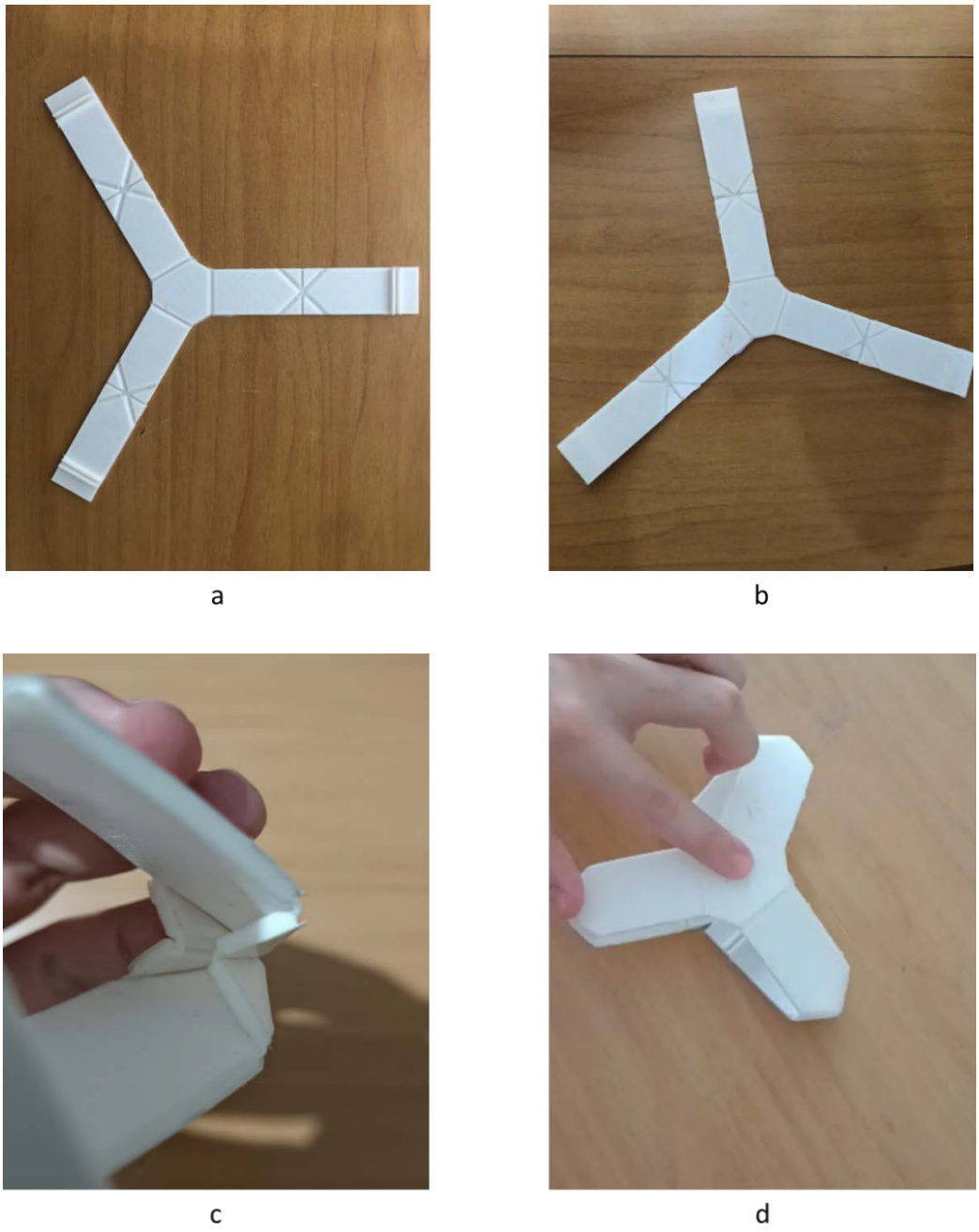


Figure 5. 2: Bottom view (a), top view (b), waterbomb pattern (c) and closed state (d)

While TPU's flexibility proved to be ideal for our purposes, we encountered a challenge during the rotations of the robot's links. Contrary to the principles of rigid origami, we observed deformations occurring on the links.

6 RESULTS

The primary objective of this project involves comprehensive kinematic and structural analyses to compare and evaluate the performance of different origami robot designs.

Among the three designs, the waterbomb origami robot exhibits a comparatively smaller workspace and bending angle limitation, primarily due to its optimum workspace occurring at $\lambda = 45°$. As the design angle increases, the presence of connections between its parts restricts its motion. On the other hand, both the RTT and serial RTT robots have the potential to expand their workspaces and bending angles as λ is increased. Notably, the two robots differ in terms of their workspaces due to the absence of a closed loop in the serial RTT robot.

The impact of removing one chain from the closed loop of the RTT robot, as well as the comparison between the workspaces of the RTT and serial RTT robots at different λ angles, is illustrated in Figure 6.1.

In the figure, the blue points represent the additional points achievable by the serial RTT robot compared to the RTT robot. It is evident that the RTT robot is unable to reach these points at the same λ angle as the serial RTT robot. This highlights the superiority of the serial RTT robot in terms of workspace considerations. However, it is important to note that the FEA results indicate that the RTT robot exhibits slightly higher stress resistance compared to the serial RTT robot.

This comprehensive analysis demonstrates the trade-offs between workspace capabilities and stress resistance when comparing the RTT and serial RTT robots. The results provide valuable insights for selecting the most suitable design based on specific requirements and constraints.

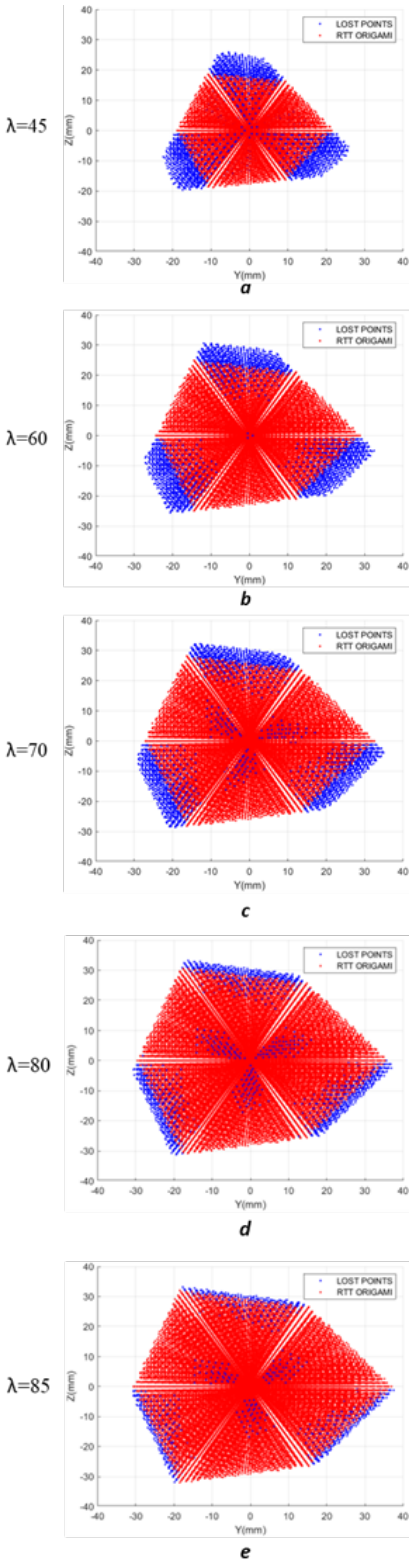


Figure 6. 1: Workplace comparison of RTT and Serial RTT: RTT (red) and difference (blue)

The λ angle and rotation angle γ significantly influence the workspace volume, number of reached points in space, and bending angle of the robots. These relationships can be observed by examining a single robot design, such as the serial RTT robot in Figure 6.2, 6.3, and 6.4.

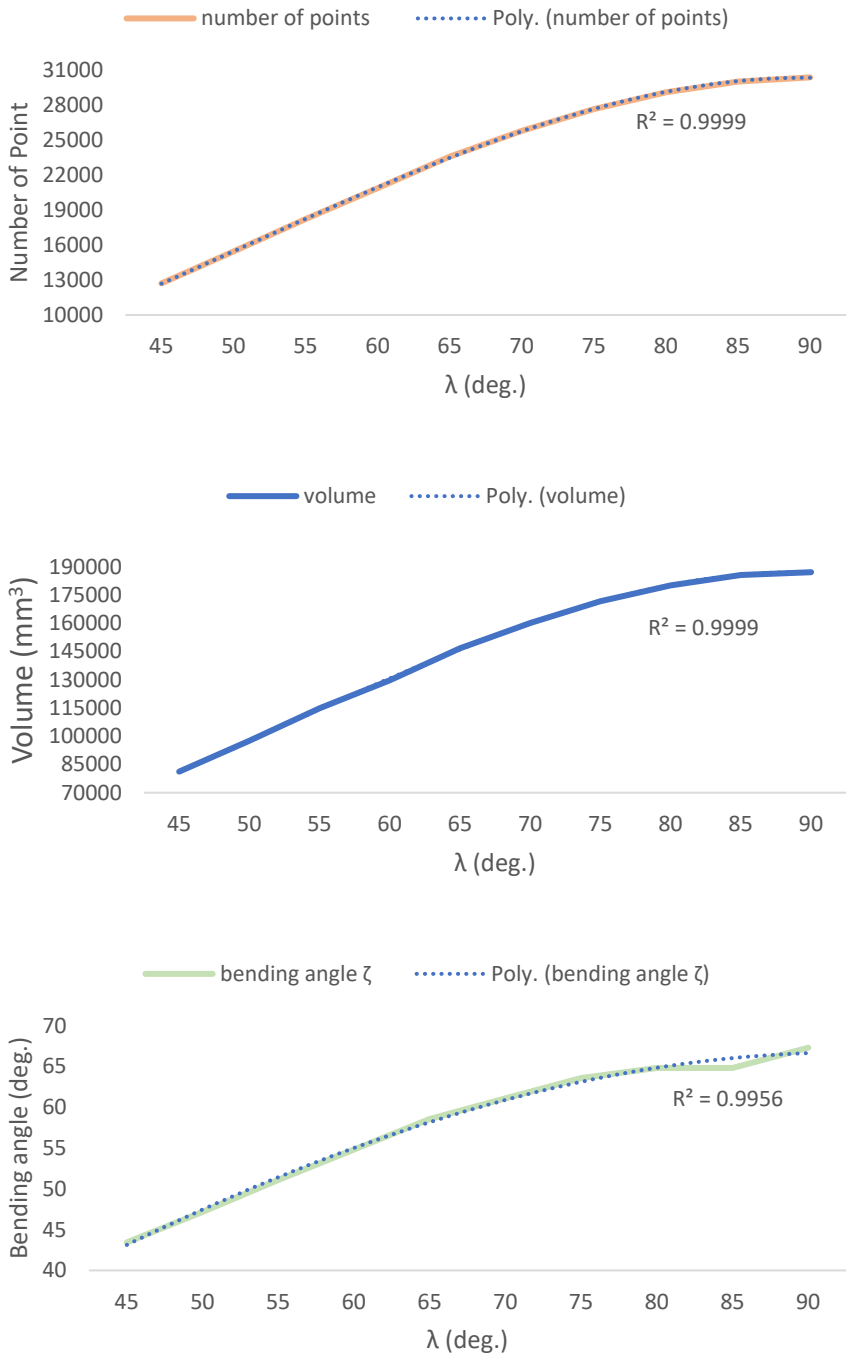


Figure 6. 2: Workspace analyses with respect to bending angle (a), volume (b), and number of reached points in space (c) at different design angles and rotation angles

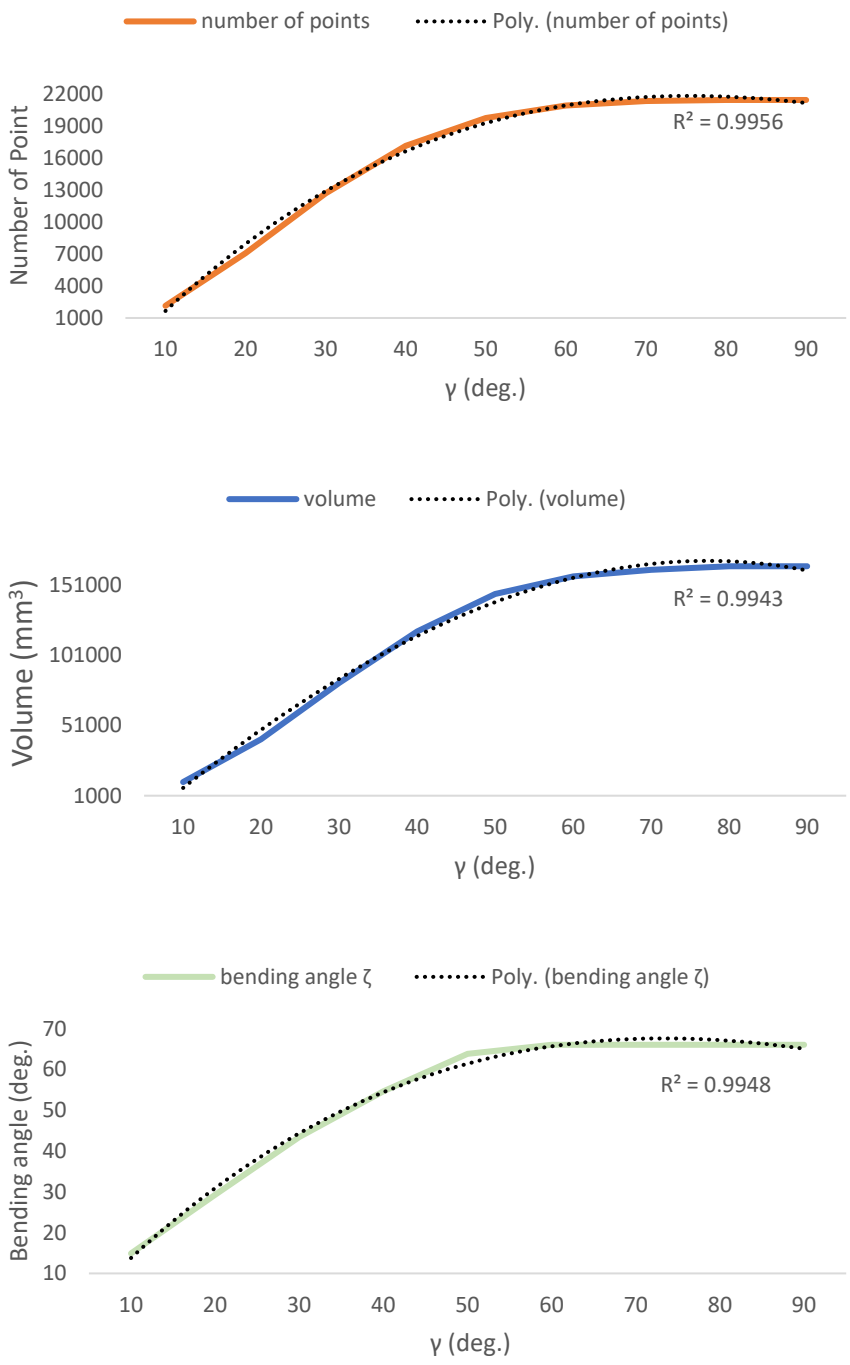


Figure 6. 3: Workspace analyses with respect to bending angle (a), volume (b), and number of reached points in space (c) at 45 deg design angle and different rotation angles

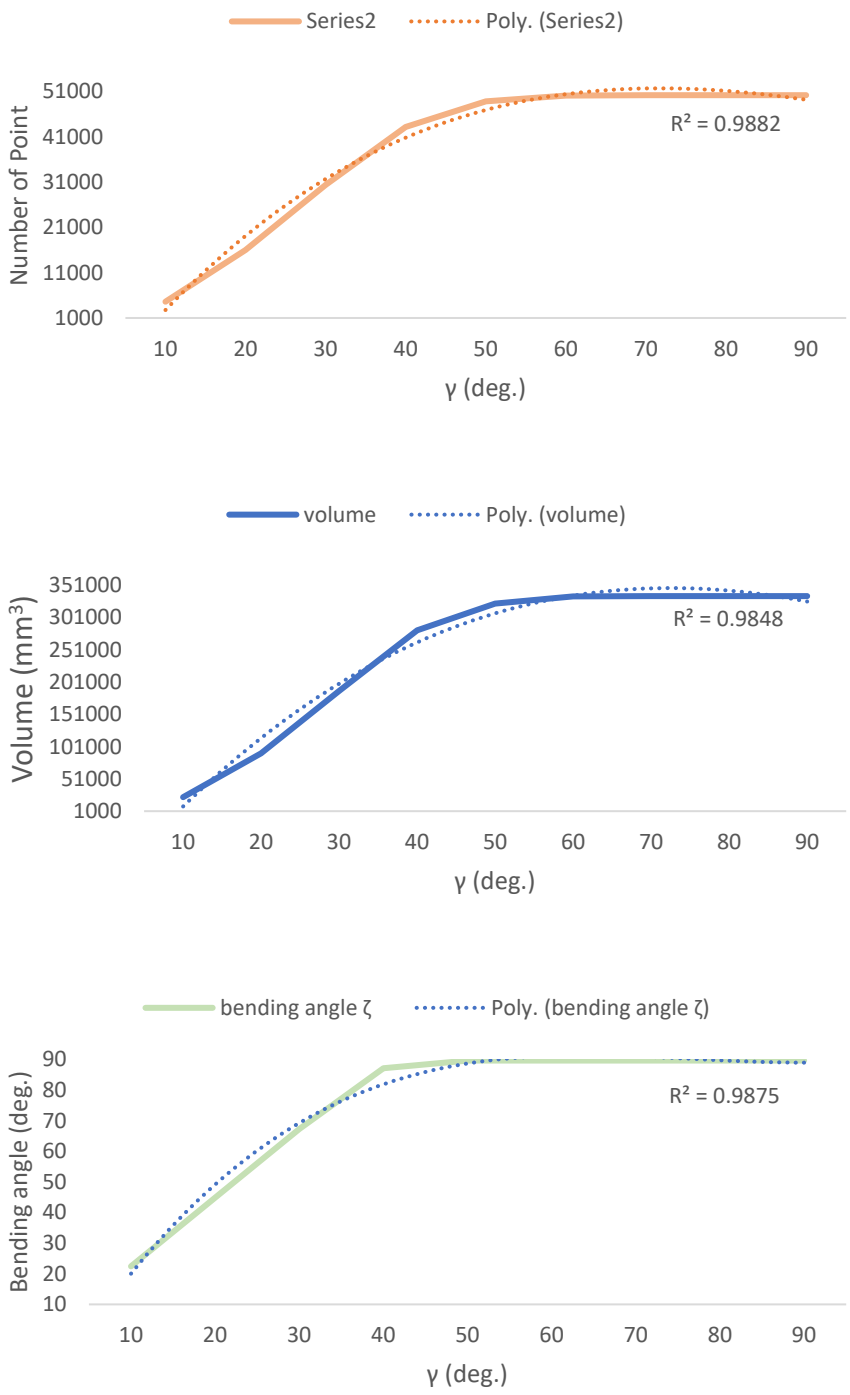


Figure 6. 4: Workspace analyses with respect bending angle ζ (a), volume (b), and number of reached points in space (c) at 90 deg design angle and different rotation angles

Additionally, the position (x, y, z) and orientation (α, β) of the Waterbomb origami robot were compared using both MATLAB and ANSYS for three specific input angles ($\theta = 145°, 120°, 125°$). The analysis was initially conducted with TPU implemented for the entire structure, followed by employing PLA for links and platforms while utilizing TPU for the joints. The corresponding results are presented as follows:

Table 6. 1: Position of top platform with analytical results in MATLAB and FEA results in ANSYS at condition of using only TPU for the waterbomb origami robot

TPU	*MATLAB*	*ANSYS*	*Difference*
$x\ (mm)$	89.50	90.41	1.02%
$y\ (mm)$	−15.90	−14.94	6.07%
$z\ (mm)$	3.34	3.22	3.73%
$\alpha\ (deg)$	−20.18	−18.76	7.05%
$\beta\ (deg)$	−4.15	−4.19	1.06%

Table 6. 2: Position of top platform with analytical results in MATLAB and FEA results in ANSYS at condition of using TPU in joints and PLA in the rests for the waterbomb origami robot

TPU + PLA	*MATLAB*	*ANSYS*	*Difference*
$x\ (mm)$	89.50	88.69	0.90%
$y\ (mm)$	−15.90	−15.43	2.97%
$z\ (mm)$	3.34	3.24	3.13%
$\alpha\ (deg)$	−20.18	−19.80	1.90%
$\beta\ (deg)$	−4.15	−4.22	1.78%

The significance of incorporating rigid origami elements becomes apparent when considering that the utilization of only flexible TPU material leads to excessive deformations on the links, resulting in increased error in the analysis. Moreover, the remarkably close agreement between the obtained results and the outcomes derived from our custom MATLAB code developed with a genetic algorithm provides robust evidence for the validity and correctness of our implemented model, reinforcing its reliability for further investigations and design optimizations.

7 CONCLUSIONS & FUTURE RECOMMENDATIONS

This study investigates the design and analysis of 3 DOF RSR parallel robots inspired by origami principles. The objective is to explore the kinematic behavior of origami robots by integrating origami patterns as joint constraints derived from Genetic Algorithm (GA) into a conventional RSR wrist mechanism. The waterbomb mechanism serves as the basis for this investigation. In response to the growing demand for enhanced workspace and bending angle capabilities, the Resch triangular tessellation (RTT) and serial RTT robots are identified as potential solutions.

The findings highlight the significant influence of the design angle λ and rotation angle γ on the workspaces of the robots. Comparative analysis reveals that the serial RTT robot exhibits the largest workspace under similar conditions, with the disparity in workspaces between the RTT and serial RTT robots decreasing as λ increases. These results emphasize the importance of considering design and rotation angles in optimizing the workspace of parallel robots, with the serial RTT configuration demonstrating promising workspace expansion capabilities.

In addition to the kinematic analysis, a finite element analysis (FEA) is conducted to examine the structural behavior of the robots. The FEA reveals slightly higher stress levels on the joints of the open-loop chain structure of the Serial RTT robot compared to the RTT robot. However, it is noteworthy that the waterbomb mechanism experiences significantly higher stresses on the joints under the same conditions. Furthermore, the positions of the end effectors are validated through a comparison between FEA results and MATLAB simulations. This validation process ensures the accuracy and reliability of overall analysis.

The FEA highlights the importance of appropriate material selection for origami robots. Initial attempts using PLA prove inadequate due to its limited flexibility, whereas the utilization of TPU exceeds expectations by providing the necessary flexibility for the waterbomb mechanism to achieve full rotation within the designated limits. However, the high flexibility of TPU leads to undesired deformations on the links during rotations, deviating from the principles of rigid origami and hindering the attainment of desired locations.

The congruence between the results obtained from both fabrication and FEA validates the conceptual correctness of the approach. To further enhance the design, a proposed solution involves employing multi-material 3D printing technology. This approach incorporates PLA for the rigid components and TPU exclusively for the joints. By integrating different materials, it becomes possible to reduce the overall thickness of the structure and further minimize the size of the origami robot, thereby improving its performance and addressing the deformation issues encountered in the previous design.

Future advancements in the field may involve including control systems, exploring alternative pattern options, or employing advanced manufacturing techniques. This could include developing new origami patterns to optimize workspace and bending angle capabilities or utilizing advanced fabrication methods such as nano-scale 3D printing or self-folding materials.

8 REFERENCES

[1] T. Tachi, "FREEFORM ORIGAMI TESSELLATIONS BY GENERALIZING RESCH'S PATTERNS," *37th Mechanisms and Robotics Conference.*, vol. 6B, 2013.

[2] H. Buri and Y. Weinand, "ORIGAMI – Folded Plate Structures, Architecture," *10th World Conference on Timber Engineering, Miyazaki, Japan*, 2008.

[3] A. Thrall and C. Quaglia, "Accordion shelters: a historical review of origami-like deployable shelters developed by the US military,," *Engineering Structures*, vol. 59, p. 686–692, 2014.

[4] S. Felton, M. Tolley, E. Demaine, D. Rus and R. Wood, "A Method for Building Self-Folding Machines," *Science*, vol. 345, no. 6197, pp. 644 - 646, 2014.

[5] J. Mu, C. Hou, H. Wang, Y. Li, Q. Zhang and M. Zhu, "Origami-Inspired Active Graphene-Based Paper for Programmable Instant Self-Folding Walking Devices," *Science Advances*, vol. 1, no. 10, 2015.

[6] A. Azam, K. E. Laflin, M. Jamal, R. Fernandes and D. Gracias H., "Self-folding micropatterned polymeric containers," *Biomedical microdevices*, vol. 13, no. 1, p. 51–58, 2011.

[7] K. Kuribayashi, K. Tsuchiya, Z. You, D. Tomus, M. Umemoto, T. Ito and M. Sasaki, "Self-Deployable Origami Stent Grafts as a Biomedical Application of Ni-Rich Tini Shape Memory Alloy Foil," *Materials Science and Engineering*, vol. 419, no. 1-2, p. 131–137, 2006.

[8] E. J. Tremblay, R. A. Stack, R. L. Morrison and J. E. Ford, "Ultrathin cameras using annular folded optics," *Applied optics*, vol. 46, no. 4, p. 463–471, 2007.

[9] J. S. Cybulski, J. Clements and M. Prakash, "Foldscope: origami-based paper microscope," *PLoS One*, vol. 9, no. 6, 2014.

[10] A. Heller, "A giant leap for space telescopes," *Science & Technology Review*, vol. 27, no. 1,

p. 12–18, 2003.

[11] A. Lebée, "From Folds to Structures, a Review," *International Journal of Space Structures*, vol. 30, no. 2, p. 55–74, 2015.

[12] Y. Chen, J. Yan and J. Feng, "Geometric and Kinematic Analyses and Novel Characteristics of Origami-Inspired Structures," *Symmetry,* vol. 11, no. 9, 2019.

[13] B. H. Hanna, J. M. Lund, R. J. Lang, S. P. Magleby and L. L. Howell, "Waterbomb base: a symmetric single-vertex bistable origami mechanism," *Smart Materials and Structures,* vol. 23, no. 9, 2014.

[14] L. Fei and S. Debnath, "Origami Theory and its Applications: A Literature Review," *International Journal of Social, Business, Psychological, Human Science and Engineering,* vol. 7, no. 1, pp. 113-117, 2013.

[15] L. M. Fonseca and M. A. Savi, "On the symmetries of the origami waterbomb pattern: kinematics and mechanical investigations," *Meccanica,* vol. 56, no. 10, pp. 2575-2598, 2021.

[16] G. W. Hunt and I. Ario, "Twist buckling and the foldable cylinder: an exercise in origami," *International Journal of Non-Linear Mechanics,* vol. 40, no. 6, pp. 833-843, 2005.

[17] E. Davis, E. D. Demaine, M. L. Demaine and J. Ramseyer, "RECONSTRUCTING DAVID HUFFMAN'S ORIGAMI TESSELLATIONS," *Journal of Mechanical Design,* vol. 135, no. 11, 2013.

[18] R. Resch, "Self-supporting structural unit having a series of repetitious geometrical modules". U.S Patent US3407558A, 29 10 1968.

[19] D. Rus and M. T. Tolley, "Design, fabrication and control of origami robots," *Nature Reviews Materials,* vol. 3, no. 6, pp. 101-112, 2018.

[20] Dae-Young Lee, G.-P. Jung, M.-K. Sin, S.-H. Ahn and K.-J. Cho, "Deformable Wheel Robot Based on Origami Structure," *IEEE International Conference on Robotics and Automation ,*

pp. 5612-5617, 2013.

[21] S. Miyashita, S. Guitron, M. Ludersdorfer, C. R. Sung and D. Rus, "An Untethered Miniature Origami Robot that Self-folds, Walks, Swims, and Degrades," *IEEE International Conference on Robotics and Automation*, pp. 1490-1496, 2015.

[22] S. Miyashita, S. Guitron, K. Yoshida, S. Li, D. D. Damian and D. Rus, "Ingestible, controllable, and degradable origami robot for patching stomach wounds," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 909-916, 2016.

[23] T. G. Nelson, A. Avila, L. L. Howell, J. L. Herder and D. F. Machekposhti, "Origami-inspired sacrificial joints for folding compliant mechanisms," *Mechanism and Machine Theory,* vol. 140, pp. 194-210, 2019.

[24] V. Megaro, J. Zehnder, M. Bächer, S. Coros, M. Gross and B. Thomaszewski, "A Computational Design Tool for Compliant Mechanisms," *ACM Transactions on Graphics,* vol. 36, no. 82, pp. 1-12, 2017.

[25] S. Linß, S. Henning and L. Zentner, "Modeling and design of flexure hinge-based compliant mechanisms," *Kinematics/Mizrahi, Joseph. - [Erscheinungsort nicht ermittelbar]: InTechOpen*, pp. 1-24, 2019.

[26] K. Zhang, C. Qiu and J. S. Dai, "An extensible continuum robot with integrated origami parallel modules," *Journal of Mechanisms and Robotics,* vol. 8, no. 3, 2016.

[27] S. L. Canfield, C. F. Reinholtz, R. J. Salerno and A. J. Ganino, "Spatial, parallel-architecture robotic carpal wrist". U.S. Patent 5,699,695, 23 12 1997.

[28] R. Di Gregorio, "Kinematics of the 3-RSR wrist," *IEEE transactions on robotics,* vol. 20, no. 4, pp. 750-753, 2004.

[29] C. Zhang, Y. Wan, D. Zhang and Q. Ma, "A new mathematical method to study the singularity of 3-RSR multimode mobile parallel mechanism," *Mathematical Problems in Engineering*, pp. 1-11, 2019.

[30] M. Bazman, N. Yilmaz and U. Tumerdem, "Dexterous and back-drivable parallel robotic forceps wrist for robotic surgery," *IEEE 15th International Workshop on Advanced Motion Control (AMC)* , pp. 153-159, 2018.

[31] M. Salerno, K. Zhang, A. Menciassi and J. S. Dai, "A novel 4-DOF origami grasper with an SMA-actuation system for minimally invasive surgery," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 484-498, 2016.

[32] B. Aboulissane, L. El Bakkali and J. El Bahaoui, "Workspace analysis and optimization of the parallel robots based on computer-aided design approach," *Facta Universitatis, Series: Mechanical Engineering*, vol. 18, no. 1, pp. 79-89, 2020.

[33] B. Aboulissane, D. El Haiek, L. El Bakkali and J. El Bahaoui, "On the workspace optimization of parallel robots based on CAD approach," *Procedia Manufacturing*, vol. 32, pp. 1085-1092, 2019.

[34] A. Oarcea, F. Popister, S. D. Stan and V. Cobilean, "Comparative study of CAD optimization features for the workspace of 3DOF Parallel Robot," *9th International Conference on Modern Power Systems (MPS)*, pp. 1-6, 2021.

[35] S. D. Stan, F. Popişter, A. Oarcea and P. Ciudin, "Comparative Study Using CAD Optimization Tools for the Workspace of a 6DOF Parallel Kinematics Machine," *Applied Sciences*, vol. 12, no. 18, 2022.

[36] M. A. Hosseini, H. R. M. Daniali and H. D. Taghirad, "Dexterous workspace optimization of a tricept parallel manipulator," *Advanced Robotics*, vol. 25, no. 13-14, pp. 1697-1712, 2011.

[37] S. Mintchev, M. Salerno, A. Cherpillod, S. Scaduto and J. Paik, "A portable three-degrees-of-freedom force feedback origami robot for human–robot interactions," *Nature Machine Intelligence*, vol. 1, no. 12, pp. 584-593, 2019.

[38] M. Salerno, J. Paik and S. Mintchev, "Ori-pixel, a multi-dofs origami pixel for modular reconfigurable surfaces," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6988-6995, 2020.

[39] Z. Zhakypov, M. Falahi, M. Shah and J. Paik, "The design and control of the multi-modal locomotion origami robot, Tribot," *IEEE/RSJ international conference on intelligent robots and systems (IROS)* , pp. 4349-4355, 2015.

[40] Y. MEI, M. N. OMIDVAR, X. LI and X. YAO, "A Competitive Divide-and-Conquer Algorithm for Unconstrained Large-Scale Black-Box Optimization," *ACM Transactions on Methematical Software*, vol. 42, no. 2, pp. 1-24, 2016.

APPENDIX A: MATLAB Codes

Obtain Workspace for Origami Robots Using '*Genetic Algorithm*' Method

Main Codes

For Waterbomb Origami Robot:

```
clear; clc; tic

%% 'Waterbomb' Origami Robot Workspace Analysis
% Optimization method --> 'Genetic Algorithm'
% Fitness Function --> 'Closed Loop'

% Define the options for the genetic algorithm
options = optimoptions('ga', 'Display', 'off', ...
    'MaxGenerations', 30, ...
    'PopulationSize', 25, ...
    'CrossoverFraction', 0.45, ...
    'EliteCount', 4, ...
    'CreationFcn', 'gacreationsobol', ...
    'CrossoverFcn', 'crossoverscattered', ...
    'FunctionTolerance', 1e-6);

%% Input Parameters
xyz0 = [0 0 0]'; % origin of base
da = deg2rad(45); % design angle of waterbomb mechanism
a = 35; % edge of equilateral triangle which is obtained from center points "F" of
links.
D = 60; % length of one link
w = 20; % wide of link
t = w/(2*cos(da)); % hypotenuse of triangle (of waterbomb).
% "t" is required for finding the locations of necessary points

lb14 = deg2rad(10); % lower bound of angle 1-4 of waterbomb
ub14 = deg2rad(170); % upper bound of angles 1-4 of waterbomb
lb23 = deg2rad(10); % lower bound of angle 2-3 of waterbomb
ub23 = deg2rad(170); % upper bound of angles 2-3 of waterbomb

lb = deg2rad(90); % lower bound of theta
ub = deg2rad(180); % upper bound of theta
db = deg2rad(15); % step angle

%% Main Code
F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

iter = length(lb:db:ub)^3; % number of points
carpalX = zeros(1,iter);
carpalY = zeros(1,iter);
```

```
carpalZ = zeros(1,iter);
carpalBending = zeros(1,iter);

i = 0; j = 0;

for theta1 = lb:db:ub
    for theta2 = lb:db:ub
        for theta3 = lb:db:ub
            i = i+1;
            %% Carpal Wrist Forward Kinematics
            % vertexs of midplane "E"
            E1 = [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
            E2 = [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
            E3 = [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];

            % the unit vector(u) directed from origin of base(B) to origin of top(P)
            vec = cross(E2-E1,E3-E1);
            u = vec/norm(vec);
            P = 2*dot(E1-xyz0,u);
            nB = [1;0;0]; % unit vector perpendicular to B
            r = P./(2*dot(nB,u)); % thrust distance
            nP = ((u*P)/r)-nB; % unit vector perpendicular to P

            r0 = dot(E1-xyz0,u)/u(1); % r0 = r

            beta = -asin(2*u(1)*u(3)); % rotation around y-axis
            alpha = asin((2*u(1)*u(2))/cos(beta)); % rotation around z-axis
            ksi = acos(dot(nP,nB)/(norm(nP)*norm(nB))); % bending angle of the top
platform

            % Defining translation matrix [P] according to the ksi <= 90deg or ksi >
90deg
            if ksi <= pi/2
                PM = [(1+cos(alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            else
                PM = [(1+cos(pi-alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            end

            % End-Effector locations of carpal wrist
            carpalXYZ = r0*PM; % end-effector locations of carpal wrist
            carpalX(i) = carpalXYZ(1); carpalY(i) = carpalXYZ(2); carpalZ(i) =
carpalXYZ(3);
            carpalBending = rad2deg(ksi);

            %% Origami Converting
            %% Leg 1
            % (1: mid) (11: right) (12: left)
            % thrust distances
            r1 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)]'-xyz0,u)/u(1);
            r11 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)-w/4]'-xyz0,u)/u(1);
            r12 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)+w/4]'-xyz0,u)/u(1);
```

```
            % base points
            F1 = xyz0+[0 -a/sqrt(3) 0]';
            F11 = xyz0+[0 -a/sqrt(3) w/4]';
            F12 = xyz0+[0 -a/sqrt(3) -w/4]';

            % top points
            P1 = F1 + r1*PM;
            P11 = F11 + r11*PM;
            P12 = F12 + r12*PM;

            % points on upper edge of waterbomb triangle
            ulink2 = (E1-P1)/(norm(E1-P1));
            U1 = P1+ulink2*(D-t*sin(da));
            U11 = P11+ulink2*(D-0.5*t*sin(da));
            U12 = P12+ulink2*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink1 = (E1-F1)/(norm(E1-F1));
            B1 = F1+ulink1*(D-t*sin(da));
            B11 = F11+ulink1*(D-0.5*t*sin(da));
            B12 = F12+ulink1*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v11 = U11-B11;
            v12 = U12-B12;

            %% Leg 2
            % thrust distances
            r2 = dot([E2(1) E2(2)-a/(2*sqrt(3)) E2(3)+a/2]'-xyz0,u)/u(1);
            r21 = dot([E2(1) E2(2)-a/(2*sqrt(3))+(w/4)*sind(60)
E2(3)+a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r22 = dot([E2(1) E2(2)-a/(2*sqrt(3))-(w/4)*sind(60) E2(3)+a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F2 = xyz0+[0 a/(2*sqrt(3)) -a/2]';
            F21 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) -a/2-(w/4)*cosd(60)]';
            F22 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) -a/2+(w/4)*cosd(60)]';

            % top points
            P2 = F2 + r2*PM;
            P21 = F21 + r21*PM;
            P22 = F22 + r22*PM;

            % points on upper edge of waterbomb triangle
            ulink22 = (E2-P2)/(norm(E2-P2));
            U2 = P2+ulink22*(D-t*sin(da));
            U21 = P21+ulink22*(D-0.5*t*sin(da));
            U22 = P22+ulink22*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink21 = (E2-F2)/(norm(E2-F2));
            B2 = F2+ulink21*(D-t*sin(da));
            B21 = F21+ulink21*(D-0.5*t*sin(da));
            B22 = F22+ulink21*(D-0.5*t*sin(da));
```

```
            % vector control (always points U must be >= points B in the x-axis)
            v21 = U21-B21;
            v22 = U22-B22;

            %% Leg 3
            % thrust distances
            r3 = dot([E3(1) E3(2)-a/(2*sqrt(3)) E3(3)-a/2]'-xyz0,u)/u(1);
            r31 = dot([E3(1) E3(2)-a/(2*sqrt(3))-(w/4)*sind(60) E3(3)-
a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r32 = dot([E3(1) E3(2)-a/(2*sqrt(3))+(w/4)*sind(60) E3(3)-a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F3 = xyz0+[0 a/(2*sqrt(3)) a/2]';
            F31 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) a/2-(w/4)*cosd(60)]';
            F32 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) a/2+(w/4)*cosd(60)]';

            % top points
            P3 = F3 + r3*PM;
            P31 = F31 + r31*PM;
            P32 = F32 + r32*PM;

            % points on upper edge of waterbomb triangle
            ulink32 = (E3-P3)/(norm(E3-P3));
            U3 = P3+ulink32*(D-t*sin(da));
            U31 = P31+ulink32*(D-0.5*t*sin(da));
            U32 = P32+ulink32*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink31 = (E3-F3)/(norm(E3-F3));
            B3 = F3+ulink31*(D-t*sin(da));
            B31 = F31+ulink31*(D-0.5*t*sin(da));
            B32 = F32+ulink31*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v31 = U31-B31;
            v32 = U32-B32;
            vx = [v11(1) v12(1) v21(1) v22(1) v31(1) v32(1)]; % collision check
vector
            %% Adding Origami Constraints
            % Measuring the angle 1-4 [rad]
            ang1 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v11)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v12)^2)/(2*(0.5*t*sin(da))^2))];
            ang2 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v21)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v22)^2)/(2*(0.5*t*sin(da))^2))];
            ang3 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v31)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v32)^2)/(2*(0.5*t*sin(da))^2))];
            % Determining whether solution ever exists
```

```
            if all(vx(:) >= 0) && all(isreal([ang1(:),ang2(:),ang3(:)]))
                % Using the MATLAB function `ga` to solve the angle 2-3 [rad]
                % Optimization-limit intervals for each link
                err = deg2rad(0);
                galb = [0, 0]+err;
                gaub = [pi, pi]-err;
                x1 = ga(@(x) fitnessfcn_v2(x,da,ang1), 2, [], [], [], [], galb, gaub,
[], options);
                x2 = ga(@(x) fitnessfcn_v2(x,da,ang2), 2, [], [], [], [], galb, gaub,
[], options);
                x3 = ga(@(x) fitnessfcn_v2(x,da,ang3), 2, [], [], [], [], galb, gaub,
[], options);

                % Validation
                valid_ang = all(lb14 <= [ang1(:), ang2(:), ang3(:)] & [ang1(:),
ang2(:), ang3(:)] <= ub14, 'all');
                valid_x = all(lb23 <= [x1(:), x2(:), x3(:)] & [x1(:), x2(:), x3(:)]
<= ub23, 'all');
                if valid_ang && valid_x
                    j = j+1;
                    % End-Effector locations of origami robot
                    oriXYZ = xyz0+r0*PM;
                    oriX(j) = oriXYZ(1); oriY(j) = oriXYZ(2); oriZ(j) = oriXYZ(3);
                    oriBending(j) = rad2deg(ksi);
                    oriThT(j,:) = rad2deg([theta1 theta2 theta3]);
                end
            end

        end
    end
end

%% Plottings
figure(1)
plot3(carpalY,carpalZ,carpalX,'.r','LineWidth',0.05)
hold on
plot3(oriY,oriZ,oriX,'.b','LineWidth',0.05)
view(-90,0)
legend('CARPAL','ORIGAMI',Location='southeast')
grid
axis('padded')

figure(2)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
view(-90,0)
grid on
axis('padded')
colorbar

%% Volume Calculation
xo = oriY'; yo = oriZ'; zo = oriX';

dataori = [xo,yo,zo]; % Combine the data into a matrix
ko = convhulln(dataori); % Compute the convex hull of the data
centroido = mean(dataori(ko(:),:)); % Calculate the center of mass of the convex hull
```

```
% Calculate the volume of the convex hull
volumeo = 0;
for i=1:size(ko,1)
    volumeo = volumeo + abs(det([dataori(ko(i,1),:)-centroido;...
        dataori(ko(i,2),:)-centroido;...
        dataori(ko(i,3),:)-centroido]))/6;
end

%% Printting
disp(['Elapsed time: ', num2str(toc), ' s']);
disp(['The max bending angle: ', num2str(max(oriBending)), ' deg']);
disp(['The volume of the origami workspace: ', num2str(round(volumeo,0)), ' mm^3']);
disp(['# of points = Carpal: ',num2str(iter),' Origami: ',num2str(j)]);

%% Solidworks Exportation
points = [zo xo yo];
k = boundary(points, 1); % Adjust the second parameter as needed for desired detail
borderPoints = points(k, :);
% figure()
% plot3(borderPoints(:,2), borderPoints(:,3), borderPoints(:,1),
'.r','MarkerSize',10);
% view(-90,0)
% grid
```

For RTT Origami Robot:

```
clear; clc; tic

%% 'RTT' Origami Robot Workspace Analysis
% Optimization method --> 'Genetic Algorithm'
% Fitness Function --> 'Closed Loop'

% Define the options for the genetic algorithm
options = optimoptions('ga', 'Display', 'off', ...
    'MaxGenerations', 30, ...
    'PopulationSize', 25, ...
    'CrossoverFraction', 0.45, ...
    'EliteCount', 4, ...
    'CreationFcn', 'gacreationsobol', ...
    'CrossoverFcn', 'crossoverscattered', ...
    'FunctionTolerance', 1e-6);

%% Input Parameters
xyz0 = [0 0 0]'; % origin of base
da = deg2rad(45); % design angle of waterbomb mechanism
a = 35; % edge of equilateral triangle which is obtained from center points "F" of
links.
D = 60; % length of one link
w = 20; % wide of link
t = w/(2*cos(da)); % hypotenuse of triangle (of waterbomb).
% "t" is required for finding the locations of necessary points

lb14 = deg2rad(10); % lower bound of angle 1-4 of waterbomb
ub14 = deg2rad(170); % upper bound of angles 1-4 of waterbomb
jointlimit = deg2rad(50); % +- joint rotation of angle 2 for waterbomb

lb = deg2rad(90); % lower bound of theta
ub = deg2rad(180); % upper bound of theta
db = deg2rad(10); % step angle

%% Main Code
F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

iter = length(lb:db:ub)^3; % number of points
carpalX = zeros(1,iter);
carpalY = zeros(1,iter);
carpalZ = zeros(1,iter);
carpalBending = zeros(1,iter);

i = 0; j = 0;

for theta1 = lb:db:ub
    for theta2 = lb:db:ub
        for theta3 = lb:db:ub
            i = i+1;
```

```
            %% Carpal Wrist Forward Kinematics
            % vertexs of midplane "E"
            E1 = [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
            E2 = [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
            E3 = [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];

            % the unit vector(u) directed from origin of base(B) to origin of top(P)
            vec = cross(E2-E1,E3-E1);
            u = vec/norm(vec);
            P = 2*dot(E1-xyz0,u);
            nB = [1;0;0]; % unit vector perpendicular to B
            r = P./(2*dot(nB,u)); % thrust distance
            nP = ((u*P)/r)-nB; % unit vector perpendicular to P

            r0 = dot(E1-xyz0,u)/u(1); % r0 = r

            beta = -asin(2*u(1)*u(3)); % rotation around y-axis
            alpha = asin((2*u(1)*u(2))/cos(beta)); % rotation around z-axis
            ksi = acos(dot(nP,nB)/(norm(nP)*norm(nB))); % bending angle of the top
platform

            % Defining translation matrix [P] according to the ksi <= 90deg or ksi >
90deg
            if ksi <= pi/2
                PM = [(1+cos(alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            else
                PM = [(1+cos(pi-alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            end

            % End-Effector locations of carpal wrist
            carpalXYZ = r0*PM; % end-effector locations of carpal wrist
            carpalX(i) = carpalXYZ(1); carpalY(i) = carpalXYZ(2); carpalZ(i) =
carpalXYZ(3);
            carpalBending = rad2deg(ksi);

            %% Origami Converting
            %% Leg 1
            % (1: mid) (11: right) (12: left)
            % thrust distances
            r1 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)]'-xyz0,u)/u(1);
            r11 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)-w/4]'-xyz0,u)/u(1);
            r12 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)+w/4]'-xyz0,u)/u(1);

            % base points
            F1 = xyz0+[0 -a/sqrt(3) 0]';
            F11 = xyz0+[0 -a/sqrt(3) w/4]';
            F12 = xyz0+[0 -a/sqrt(3) -w/4]';

            % top points
            P1 = F1 + r1*PM;
            P11 = F11 + r11*PM;
            P12 = F12 + r12*PM;
```

```
            % points on upper edge of waterbomb triangle
            ulink2 = (E1-P1)/(norm(E1-P1));
            U1 = P1+ulink2*(D-t*sin(da));
            U11 = P11+ulink2*(D-0.5*t*sin(da));
            U12 = P12+ulink2*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink1 = (E1-F1)/(norm(E1-F1));
            B1 = F1+ulink1*(D-t*sin(da));
            B11 = F11+ulink1*(D-0.5*t*sin(da));
            B12 = F12+ulink1*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v11 = U11-B11;
            v12 = U12-B12;

            %% Leg 2
            % thrust distances
            r2 = dot([E2(1) E2(2)-a/(2*sqrt(3)) E2(3)+a/2]'-xyz0,u)/u(1);
            r21 = dot([E2(1) E2(2)-a/(2*sqrt(3))+(w/4)*sind(60)
E2(3)+a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r22 = dot([E2(1) E2(2)-a/(2*sqrt(3))-(w/4)*sind(60) E2(3)+a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F2 = xyz0+[0 a/(2*sqrt(3)) -a/2]';
            F21 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) -a/2-(w/4)*cosd(60)]';
            F22 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) -a/2+(w/4)*cosd(60)]';

            % top points
            P2 = F2 + r2*PM;
            P21 = F21 + r21*PM;
            P22 = F22 + r22*PM;

            % points on upper edge of waterbomb triangle
            ulink22 = (E2-P2)/(norm(E2-P2));
            U2 = P2+ulink22*(D-t*sin(da));
            U21 = P21+ulink22*(D-0.5*t*sin(da));
            U22 = P22+ulink22*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink21 = (E2-F2)/(norm(E2-F2));
            B2 = F2+ulink21*(D-t*sin(da));
            B21 = F21+ulink21*(D-0.5*t*sin(da));
            B22 = F22+ulink21*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v21 = U21-B21;
            v22 = U22-B22;

            %% Leg 3
            % thrust distances
            r3 = dot([E3(1) E3(2)-a/(2*sqrt(3)) E3(3)-a/2]'-xyz0,u)/u(1);
```

```
            r31 = dot([E3(1) E3(2)-a/(2*sqrt(3))-(w/4)*sind(60) E3(3)-
a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r32 = dot([E3(1) E3(2)-a/(2*sqrt(3))+(w/4)*sind(60) E3(3)-a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F3 = xyz0+[0 a/(2*sqrt(3)) a/2]';
            F31 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) a/2-(w/4)*cosd(60)]';
            F32 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) a/2+(w/4)*cosd(60)]';

            % top points
            P3 = F3 + r3*PM;
            P31 = F31 + r31*PM;
            P32 = F32 + r32*PM;

            % points on upper edge of waterbomb triangle
            ulink32 = (E3-P3)/(norm(E3-P3));
            U3 = P3+ulink32*(D-t*sin(da));
            U31 = P31+ulink32*(D-0.5*t*sin(da));
            U32 = P32+ulink32*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink31 = (E3-F3)/(norm(E3-F3));
            B3 = F3+ulink31*(D-t*sin(da));
            B31 = F31+ulink31*(D-0.5*t*sin(da));
            B32 = F32+ulink31*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v31 = U31-B31;
            v32 = U32-B32;
            vx = [v11(1) v12(1) v21(1) v22(1) v31(1) v32(1)]; % collision check
vector

            %% Adding Origami Constraints
            % Measuring the angle 1-4 [rad]
            ang1 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v11)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v12)^2)/(2*(0.5*t*sin(da))^2))];
            ang2 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v21)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v22)^2)/(2*(0.5*t*sin(da))^2))];
            ang3 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v31)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v32)^2)/(2*(0.5*t*sin(da))^2))];
            % Determining whether solution ever exists
            if all(vx(:) >= 0) && all(isreal([ang1(:),ang2(:),ang3(:)]))
                % Using the MATLAB function `ga` to solve the angle 2-3 [rad]
                % Optimization-limit intervals for each link
                if ang1(1) ~= ang1(2)
                    if ang1(1) < ang1(2)
                        galb1 = [pi/2, pi];
                        gaub1 = [pi, 3*pi/2];
```

```
                    else
                        galb1 = [pi, pi/2];
                        gaub1 = [3*pi/2, pi];
                    end
                else
                    galb1 = [pi, pi];
                    gaub1 = [pi, pi];
                end
                if ang2(1) ~= ang2(2)
                    if ang2(1) < ang2(2)
                        galb2 = [pi/2, pi];
                        gaub2 = [pi, 3*pi/2];
                    else
                        galb2 = [pi, pi/2];
                        gaub2 = [3*pi/2, pi];
                    end
                else
                    galb2 = [pi, pi];
                    gaub2 = [pi, pi];
                end
                if ang3(1) ~= ang3(2)
                    if ang3(1) < ang3(2)
                        galb3 = [pi/2, pi];
                        gaub3 = [pi, 3*pi/2];
                    else
                        galb3 = [pi, pi/2];
                        gaub3 = [3*pi/2, pi];
                    end
                else
                    galb3 = [pi, pi];
                    gaub3 = [pi, pi];
                end
                x1 = ga(@(x) fitnessfcn_v2(x,da,ang1), 2, [], [], [], [], galb1,
gaub1, [], options);
                x2 = ga(@(x) fitnessfcn_v2(x,da,ang2), 2, [], [], [], [], galb2,
gaub2, [], options);
                x3 = ga(@(x) fitnessfcn_v2(x,da,ang3), 2, [], [], [], [], galb3,
gaub3, [], options);

                % Validation
                valid_ang14 = all(lb14 <= [ang1(:), ang2(:), ang3(:)] & [ang1(:),
ang2(:), ang3(:)] <= ub14, 'all');
                valid_x23 = all(pi - jointlimit <= [x1(:), x2(:), x3(:)] & [x1(:),
x2(:), x3(:)] <= pi + jointlimit, 'all');

                if valid_ang14 && valid_x23
                    j = j+1;
                    % End-Effector locations of origami robot
                    oriXYZ = xyz0+r0*PM;
                    oriX(j) = oriXYZ(1); oriY(j) = oriXYZ(2); oriZ(j) = oriXYZ(3);
                    oriBending(j) = rad2deg(ksi);
                    oriThT(j,:) = rad2deg([theta1 theta2 theta3]);
                end
            end
```

```
        end
    end
end

%% Plottings
figure(1)
plot3(carpalY,carpalZ,carpalX,'.r','LineWidth',0.05)
hold on
plot3(oriY,oriZ,oriX,'.b','LineWidth',0.05)
view(-90,0)
legend('CARPAL','ORIGAMI',Location='southeast')
grid
axis('padded')

figure(2)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
view(-90,0)
grid on
axis('padded')
colorbar

%% Volume Calculation
xo = oriY'; yo = oriZ'; zo = oriX';

dataori = [xo,yo,zo]; % Combine the data into a matrix
ko = convhulln(dataori); % Compute the convex hull of the data
centroido = mean(dataori(ko(:),:)); % Calculate the center of mass of the convex hull
% Calculate the volume of the convex hull
volumeo = 0;
for i=1:size(ko,1)
    volumeo = volumeo + abs(det([dataori(ko(i,1),:)-centroido;...
        dataori(ko(i,2),:)-centroido;...
        dataori(ko(i,3),:)-centroido]))/6;
end

%% Printting
disp(['Elapsed time: ', num2str(toc), ' s']);
disp(['The max bending angle: ', num2str(max(oriBending)), ' deg']);
disp(['The volume of the origami workspace: ', num2str(round(volumeo,0)), ' mm^3']);
disp(['# of points = Carpal: ',num2str(iter),' Origami: ',num2str(j)]);

%% Solidworks Exportation
points = [zo xo yo];
k = boundary(points, 1); % Adjust the second parameter as needed for desired detail
borderPoints = points(k, :);
% figure()
% plot3(borderPoints(:,2), borderPoints(:,3), borderPoints(:,1),
'.r','MarkerSize',10);
% view(-90,0)
% grid
```

For Serial RTT Origami Robot:

```
clear; clc; tic

%% 'Serial RTT' Origami Robot Workspace Analysis
% Optimization method --> 'Genetic Algorithm'
% Fitness Function --> 'Open Loop'

% Define the options for the genetic algorithm
options = optimoptions('ga', 'Display', 'off', ...
    'MaxGenerations', 15, ...
    'PopulationSize', 15, ...
    'FunctionTolerance', 1e-15);

%% Input Parameters
xyz0 = [0 0 0]'; % origin of base
da = deg2rad(80); % design angle of waterbomb mechanism
a = 35; % edge of equilateral triangle which is obtained from center points "F" of
links.
D = 60; % length of one link
w = 20; % wide of link
t = w/(2*cos(da)); % hypotenuse of triangle (of waterbomb).
% "t" is required for finding the locations of necessary points

lb14 = deg2rad(10); % lower bound of angle 1-4 of waterbomb
ub14 = deg2rad(170); % upper bound of angles 1-4 of waterbomb
jointlimit = deg2rad(50); % +- joint rotation of angle 2 for waterbomb

lb = deg2rad(90); % lower bound of theta
ub = deg2rad(180); % upper bound of theta
db = deg2rad(10); % step angle

%% Main Code
F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

iter = length(lb:db:ub)^3; % number of points
carpalX = zeros(1,iter);
carpalY = zeros(1,iter);
carpalZ = zeros(1,iter);
carpalBending = zeros(1,iter);

i = 0; j = 0;

for theta1 = lb:db:ub
    for theta2 = lb:db:ub
        for theta3 = lb:db:ub
            i = i+1;
            %% Carpal Wrist Forward Kinematics
            % vertexs of midplane "E"
            E1 = [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
```

```
            E2 = [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
            E3 = [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];

            % the unit vector(u) directed from origin of base(B) to origin of top(P)
            vec = cross(E2-E1,E3-E1);
            u = vec/norm(vec);
            P = 2*dot(E1-xyz0,u);
            nB = [1;0;0]; % unit vector perpendicular to B
            r = P./(2*dot(nB,u)); % thrust distance
            nP = ((u*P)/r)-nB; % unit vector perpendicular to P

            r0 = dot(E1-xyz0,u)/u(1); % r0 = r

            beta = -asin(2*u(1)*u(3)); % rotation around y-axis
            alpha = asin((2*u(1)*u(2))/cos(beta)); % rotation around z-axis
            ksi = acos(dot(nP,nB)/(norm(nP)*norm(nB))); % bending angle of the top
platform

            % Defining translation matrix [P] according to the ksi <= 90deg or ksi >
90deg
            if ksi <= pi/2
                PM = [(1+cos(alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            else
                PM = [(1+cos(pi-alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            end

            % End-Effector locations of carpal wrist
            carpalXYZ = r0*PM; % end-effector locations of carpal wrist
            carpalX(i) = carpalXYZ(1); carpalY(i) = carpalXYZ(2); carpalZ(i) =
carpalXYZ(3);
            carpalBending = rad2deg(ksi);

            %% Origami Converting
            %% Leg 1
            % (1: mid) (11: right) (12: left)
            % thrust distances
            r1 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)]'-xyz0,u)/u(1);
            r11 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)-w/4]'-xyz0,u)/u(1);
            r12 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)+w/4]'-xyz0,u)/u(1);

            % base points
            F1 = xyz0+[0 -a/sqrt(3) 0]';
            F11 = xyz0+[0 -a/sqrt(3) w/4]';
            F12 = xyz0+[0 -a/sqrt(3) -w/4]';

            % top points
            P1 = F1 + r1*PM;
            P11 = F11 + r11*PM;
            P12 = F12 + r12*PM;

            % points on upper edge of waterbomb triangle
            ulink2 = (E1-P1)/(norm(E1-P1));
            U1 = P1+ulink2*(D-t*sin(da));
```

```
            U11 = P11+ulink2*(D-0.5*t*sin(da));
            U12 = P12+ulink2*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink1 = (E1-F1)/(norm(E1-F1));
            B1 = F1+ulink1*(D-t*sin(da));
            B11 = F11+ulink1*(D-0.5*t*sin(da));
            B12 = F12+ulink1*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v11 = U11-B11;
            v12 = U12-B12;

            %% Leg 2
            % thrust distances
            r2 = dot([E2(1) E2(2)-a/(2*sqrt(3)) E2(3)+a/2]'-xyz0,u)/u(1);
            r21 = dot([E2(1) E2(2)-a/(2*sqrt(3))+(w/4)*sind(60)
E2(3)+a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r22 = dot([E2(1) E2(2)-a/(2*sqrt(3))-(w/4)*sind(60) E2(3)+a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F2 = xyz0+[0 a/(2*sqrt(3)) -a/2]';
            F21 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) -a/2-(w/4)*cosd(60)]';
            F22 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) -a/2+(w/4)*cosd(60)]';

            % top points
            P2 = F2 + r2*PM;
            P21 = F21 + r21*PM;
            P22 = F22 + r22*PM;

            % points on upper edge of waterbomb triangle
            ulink22 = (E2-P2)/(norm(E2-P2));
            U2 = P2+ulink22*(D-t*sin(da));
            U21 = P21+ulink22*(D-0.5*t*sin(da));
            U22 = P22+ulink22*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink21 = (E2-F2)/(norm(E2-F2));
            B2 = F2+ulink21*(D-t*sin(da));
            B21 = F21+ulink21*(D-0.5*t*sin(da));
            B22 = F22+ulink21*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v21 = U21-B21;
            v22 = U22-B22;

            %% Leg 3
            % thrust distances
            r3 = dot([E3(1) E3(2)-a/(2*sqrt(3)) E3(3)-a/2]'-xyz0,u)/u(1);
            r31 = dot([E3(1) E3(2)-a/(2*sqrt(3))-(w/4)*sind(60) E3(3)-
a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r32 = dot([E3(1) E3(2)-a/(2*sqrt(3))+(w/4)*sind(60) E3(3)-a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);
```

```
            % base points
            F3 = xyz0+[0 a/(2*sqrt(3)) a/2]';
            F31 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) a/2-(w/4)*cosd(60)]';
            F32 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) a/2+(w/4)*cosd(60)]';

            % top points
            P3 = F3 + r3*PM;
            P31 = F31 + r31*PM;
            P32 = F32 + r32*PM;

            % points on upper edge of waterbomb triangle
            ulink32 = (E3-P3)/(norm(E3-P3));
            U3 = P3+ulink32*(D-t*sin(da));
            U31 = P31+ulink32*(D-0.5*t*sin(da));
            U32 = P32+ulink32*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink31 = (E3-F3)/(norm(E3-F3));
            B3 = F3+ulink31*(D-t*sin(da));
            B31 = F31+ulink31*(D-0.5*t*sin(da));
            B32 = F32+ulink31*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v31 = U31-B31;
            v32 = U32-B32;
            vx = [v11(1) v12(1) v21(1) v22(1) v31(1) v32(1)]; % collision check
vector

            %% Adding Origami Constraints
            % Measuring the angle 1-4 [rad]
            ang1 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v11)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v12)^2)/(2*(0.5*t*sin(da))^2))];
            ang2 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v21)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v22)^2)/(2*(0.5*t*sin(da))^2))];
            ang3 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v31)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v32)^2)/(2*(0.5*t*sin(da))^2))];
            % Defining fitness function parameters
            tar1 = P1-F1;
            tar2 = P2-F2;
            tar3 = P3-F3;
            rot = deg2rad([0 120 240]);
            % Determining whether solution ever exists
            if all(vx(:) >= 0) && all(isreal([ang1(1),ang2(1),ang3(1)]))
                % Using the MATLAB function `ga` to solve the angle 2-3 [rad]
                galb = pi/2; gaub = 3*pi/2;
                x1 = [ga(@(x) serialOBJ1(x,da,ang1(1),tar1,theta1,D,rot(1)), 1, [],
[], [], [], galb, gaub, [], options)];
                x2 = [ga(@(x) serialOBJ1(x,da,ang2(1),tar2,theta2,D,rot(2)), 1, [],
[], [], [], galb, gaub, [], options)];
```

```
                x3 = [ga(@(x) serialOBJ1(x,da,ang3(1),tar3,theta3,D,rot(3)), 1, [],
[], [], [], galb, gaub, [], options)];

                % Validation
                valid_ang1 = all(lb14 <= [ang1(1), ang2(1), ang3(1)] & [ang1(1),
ang2(1), ang3(1)] <= ub14, 'all');
                valid_x2 = all(pi-jointlimit <= [x1(1), x2(1), x3(1)] & [x1(1),
x2(1), x3(1)] <= pi+jointlimit, 'all');

                if valid_ang1 && valid_x2
                    j = j+1;
                    % End-Effector locations of origami robot
                    oriXYZ = xyz0+r0*PM;
                    oriX(j) = oriXYZ(1); oriY(j) = oriXYZ(2); oriZ(j) = oriXYZ(3);
                    oriBending(j) = rad2deg(ksi);
                    oriThT(j,:) = rad2deg([theta1 theta2 theta3]);
                end
            end

        end
    end
end

%% Plottings
figure(1)
plot3(carpalY,carpalZ,carpalX,'.r','LineWidth',0.05)
hold on
plot3(oriY,oriZ,oriX,'.b','LineWidth',0.05)
view(-90,0)
legend('CARPAL','ORIGAMI',Location='southeast')
grid
axis('padded')

figure(2)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
view(-90,0)
grid on
axis('padded')
colorbar

%% Volume Calculation
xo = oriY'; yo = oriZ'; zo = oriX';

dataori = [xo,yo,zo]; % Combine the data into a matrix
ko = convhulln(dataori); % Compute the convex hull of the data
centroido = mean(dataori(ko(:),:)); % Calculate the center of mass of the convex hull
% Calculate the volume of the convex hull
volumeo = 0;
for i=1:size(ko,1)
    volumeo = volumeo + abs(det([dataori(ko(i,1),:)-centroido;...
        dataori(ko(i,2),:)-centroido;...
        dataori(ko(i,3),:)-centroido]))/6;
end

%% Printting
```

```
disp(['Elapsed time: ', num2str(toc), ' s']);
disp(['The max bending angle: ', num2str(max(oriBending)), ' deg']);
disp(['The volume of the origami workspace: ', num2str(round(volumeo,0)), ' mm^3']);
disp(['# of points = Carpal: ',num2str(iter),' Origami: ',num2str(j)]);

%% Solidworks Exportation
points = [zo xo yo];
k = boundary(points, 1); % Adjust the second parameter as needed for desired detail
borderPoints = points(k, :);
% figure()
% plot3(borderPoints(:,2), borderPoints(:,3), borderPoints(:,1),
'.r','MarkerSize',10);
% view(-90,0)
% grid
```

Objective / Fitness Functions

For Waterbomb and RTT Origami Robots:

```
function y = fitnessfcn(x,da,angle)

ang1 = angle(1);
ang2 = x(1);
ang3 = x(2);
ang4 = angle(2);
ang5 = ang3;
ang6 = ang2;

ang = pi-[ang2 ang3 ang4 ang5 ang6 ang1];

% DH TABLE -- [a,α,d,θ]
dh = [0 pi-da 0 ang(1);...
    0 pi+2*da 0 ang(2);...
    0 pi-da 0 ang(3);...
    0 pi-da 0 ang(4);...
    0 pi+2*da 0 ang(5);...
    0 pi-da 0 ang(6)];
% TRANSFORMATION MATRIX
ab = size(dh);
link = ab(1);
T = eye(4);
t = eye(4);
for i = 1:link
    t(:,:,i) = [cos(dh(i,4)) -sin(dh(i,4)) 0 dh(i,1);...
        sin(dh(i,4))*cos(dh(i,2)) cos(dh(i,4))*cos(dh(i,2)) -sin(dh(i,2)) -
sin(dh(i,2))*dh(i,3);...
        sin(dh(i,4))*sin(dh(i,2)) cos(dh(i,4))*sin(dh(i,2)) cos(dh(i,2))
cos(dh(i,2))*dh(i,3);...
        0 0 0 1];
    T = T*t(:,:,i); % Transformation Matrix
end

TM = T((1:3),(1:3));
target = eye(3);
y = sum(sum((target-TM).^2));

end
```

For Serial RTT Origami Robot:

```
function y = serialOBJ1(x,da,WB1,target,theta,D,rot)

ang = pi-[2*pi-x WB1 -(2*pi-x)];

% DH TABLE -- [a,α,d,θ]
dh = [0 rot 0 pi/2; ...
    0 0 0 pi/2-theta; ...
    0 pi/2 D 0; ...
    0 pi/2-da 0 ang(1); ...
    0 pi+da 0 ang(2); ...
    0 da 0 ang(3); ...
    0 pi/2-da D 0];

% TRANSFORMATION MATRIX
ab = size(dh);
link = ab(1);
T = eye(4);
t = eye(4);
for i = 1:link
    t(:,:,i) = [cos(dh(i,4)) -sin(dh(i,4)) 0 dh(i,1);...
        sin(dh(i,4))*cos(dh(i,2)) cos(dh(i,4))*cos(dh(i,2)) -sin(dh(i,2)) -
sin(dh(i,2))*dh(i,3);...
        sin(dh(i,4))*sin(dh(i,2)) cos(dh(i,4))*sin(dh(i,2)) cos(dh(i,2))
cos(dh(i,2))*dh(i,3);...
        0 0 0 1];
    T = T*t(:,:,i); % Transformation Matrix
end

PM = T((1:3),4);
y = sum(sum((target-PM).^2));

end
```

Obtain Workspace for Origami Robots Using '*Divide and Conquer*' Method

Main Codes

For Waterbomb Origami Robot:

```
clear; clc; tic

%% Waterbomb BASED ORIGAMI ROBOT WORKSPACE with "Divide & Conquer" Optimization
% The workspace of Waterbomb based origami robot can be obtained
% by limiting angle 1-4 and angle 2-3 of waterbomb.

xyz0 = [0 0 0]'; % origin of base
da = deg2rad(45); % design angle of waterbomb mechanism
a = 35; % edge of equilateral triangle which is obtained from center points "F" of
links.
D = 60; % length of one link
w = 20; % wide of link
t = w/(2*cos(da)); % hypotenuse of triangle (of waterbomb).
% "t" is required for finding the locations of necessary points

F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

lb = deg2rad(90); % lower bound of theta
ub = deg2rad(180); % upper bound of theta
db = deg2rad(2.5); % step angle

lb14 = deg2rad(10); % lower bound of angle 1-4 of waterbomb
ub14 = deg2rad(170); % upper bound of angles 1-4 of waterbomb

lb23 = deg2rad(0); % lower bound of angle 2-3 of waterbomb
ub23 = deg2rad(180); % upper bound of angles 2-3 of waterbomb

iter = length(lb:db:ub)^3; % number of points

carpalX = zeros(1,iter);
carpalY = zeros(1,iter);
carpalZ = zeros(1,iter);
carpalBending = zeros(1, iter);

i = 0;
j = 0;

for theta1 = lb:db:ub
    for theta2 = lb:db:ub
        for theta3 = lb:db:ub

            i = i+1;
```

```
            %% Carpal Wrist Forward Kinematics
            % vertexs of midplane "E"
            E1 = [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
            E2 = [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
            E3 = [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];

            % the unit vector(u) directed from origin of base(B) to origin of top(P)
            vec = cross(E2-E1,E3-E1);
            u = vec/norm(vec);
            P = 2*dot(E1-xyz0,u);
            nB = [1;0;0]; % unit vector perpendicular to B
            r = P./(2*dot(nB,u)); % thrust distance
            nP = ((u*P)/r)-nB; % unit vector perpendicular to P

            r0 = dot(E1-xyz0,u)/u(1); % r0 = r

            beta = -asin(2*u(1)*u(3)); % rotation around y-axis
            alpha = asin((2*u(1)*u(2))/cos(beta)); % rotation around z-axis
            ksi = acos(dot(nP,nB)/(norm(nP)*norm(nB))); % bending angle of the top
platform

            % Defining translation matrix [P] according to the ksi <= 90deg
            % or ksi > 90deg
            if ksi <= pi/2
                PM = [(1+cos(alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            else
                PM = [(1+cos(pi-alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            end

            carpalXYZ = r0*PM; % end-effector locations of carpal wrist
            carpalX(i) = carpalXYZ(1); carpalY(i) = carpalXYZ(2); carpalZ(i) =
carpalXYZ(3);
            carpalBending = rad2deg(ksi);

            %% Origami Converting
            %% Leg 1
            % (1: mid) (11: right) (12: left)
            % thrust distances
            r1 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)]'-xyz0,u)/u(1);
            r11 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)-w/4]'-xyz0,u)/u(1);
            r12 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)+w/4]'-xyz0,u)/u(1);

            % base points
            F1 = xyz0+[0 -a/sqrt(3) 0]';
            F11 = xyz0+[0 -a/sqrt(3) w/4]';
            F12 = xyz0+[0 -a/sqrt(3) -w/4]';

            % top points
            P1 = F1 + r1*PM;
            P11 = F11 + r11*PM;
            P12 = F12 + r12*PM;
```

```
            % points on upper edge of waterbomb triangle
            ulink2 = (E1-P1)/(norm(E1-P1));
            U1 = P1+ulink2*(D-t*sin(da));
            U11 = P11+ulink2*(D-0.5*t*sin(da));
            U12 = P12+ulink2*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink1 = (E1-F1)/(norm(E1-F1));
            B1 = F1+ulink1*(D-t*sin(da));
            B11 = F11+ulink1*(D-0.5*t*sin(da));
            B12 = F12+ulink1*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v11 = U11-B11;
            v12 = U12-B12;

            %% Leg 2
            % thrust distances
            r2 = dot([E2(1) E2(2)-a/(2*sqrt(3)) E2(3)+a/2]'-xyz0,u)/u(1);
            r21 = dot([E2(1) E2(2)-a/(2*sqrt(3))+(w/4)*sind(60)
E2(3)+a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r22 = dot([E2(1) E2(2)-a/(2*sqrt(3))-(w/4)*sind(60) E2(3)+a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F2 = xyz0+[0 a/(2*sqrt(3)) -a/2]';
            F21 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) -a/2-(w/4)*cosd(60)]';
            F22 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) -a/2+(w/4)*cosd(60)]';

            % top points
            P2 = F2 + r2*PM;
            P21 = F21 + r21*PM;
            P22 = F22 + r22*PM;

            % points on upper edge of waterbomb triangle
            ulink22 = (E2-P2)/(norm(E2-P2));
            U2 = P2+ulink22*(D-t*sin(da));
            U21 = P21+ulink22*(D-0.5*t*sin(da));
            U22 = P22+ulink22*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink21 = (E2-F2)/(norm(E2-F2));
            B2 = F2+ulink21*(D-t*sin(da));
            B21 = F21+ulink21*(D-0.5*t*sin(da));
            B22 = F22+ulink21*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v21 = U21-B21;
            v22 = U22-B22;

            %% Leg 3
            % thrust distances
            r3 = dot([E3(1) E3(2)-a/(2*sqrt(3)) E3(3)-a/2]'-xyz0,u)/u(1);
            r31 = dot([E3(1) E3(2)-a/(2*sqrt(3))-(w/4)*sind(60) E3(3)-
a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
```

```
            r32 = dot([E3(1) E3(2)-a/(2*sqrt(3))+(w/4)*sind(60) E3(3)-a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F3 = xyz0+[0 a/(2*sqrt(3)) a/2]';
            F31 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) a/2-(w/4)*cosd(60)]';
            F32 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) a/2+(w/4)*cosd(60)]';

            % top points
            P3 = F3 + r3*PM;
            P31 = F31 + r31*PM;
            P32 = F32 + r32*PM;

            % points on upper edge of waterbomb triangle
            ulink32 = (E3-P3)/(norm(E3-P3));
            U3 = P3+ulink32*(D-t*sin(da));
            U31 = P31+ulink32*(D-0.5*t*sin(da));
            U32 = P32+ulink32*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink31 = (E3-F3)/(norm(E3-F3));
            B3 = F3+ulink31*(D-t*sin(da));
            B31 = F31+ulink31*(D-0.5*t*sin(da));
            B32 = F32+ulink31*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v31 = U31-B31;
            v32 = U32-B32;
            vx = [v11(1) v12(1) v21(1) v22(1) v31(1) v32(1)]; % collision check
vector

            %% Adding Origami Constraints

            % Measuring the angle 1-4 [rad]
            ang1 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v11)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v12)^2)/(2*(0.5*t*sin(da))^2))];
            ang2 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v21)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v22)^2)/(2*(0.5*t*sin(da))^2))];
            ang3 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v31)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v32)^2)/(2*(0.5*t*sin(da))^2))];

            % Finding whether solution ever exists
            if all(vx(:) >= 0) && all(isreal([ang1(:),ang2(:),ang3(:)]))
                % Using the `Divide & Conquer` method to solve the angle 2-3 [rad]
                x1 = DnQ_epfl(ang1,da);
                x2 = DnQ_epfl(ang2,da);
                x3 = DnQ_epfl(ang3,da);
                % Constraint
```

```
                valid_ang = all(lb14 <= [ang1(:), ang2(:), ang3(:)] & [ang1(:),
ang2(:), ang3(:)] <= ub14, 'all');
                valid_x = all(lb23 <= [x1(:), x2(:), x3(:)] & [x1(:), x2(:), x3(:)]
<= ub23, 'all');
                if valid_ang && valid_x
                    j = j+1;
                    oriXYZ = r0*PM; % end-effector locations of origami robot
                    oriX(j) = oriXYZ(1); oriY(j) = oriXYZ(2); oriZ(j) = oriXYZ(3);
                    oriBending(j) = ksi*(180/pi);
                    theta(j,:) = rad2deg([theta1 theta2 theta3]);
                end
            end

        end
    end
end

%% Plottings
figure(1)
plot3(carpalY,carpalZ,carpalX,'.r','LineWidth',0.05)
hold on
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
plot3(oriY,oriZ,oriX,'.b','LineWidth',0.05)
view(-90,0)
legend('CARPAL','ORIGAMI',Location='southeast')
grid on


figure(2)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
axis([ -35 40 -35 40 0 130 ])
view(145,10)
grid on
h=colorbar;
h.Title.String = 'Bending angle ζ';


figure(3)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
axis([ -35 40 -35 40 0 130 ])
view(0,90)
grid on
h=colorbar;
h.Title.String = 'Bending angle ζ';
%% Volume Calculation
xo = oriY'; yo = oriZ'; zo = oriX';
```

```
% Combine the data into a matrix
dataori = [xo,yo,zo];

% Compute the convex hull of the data
ko = convhulln(dataori);

% Calculate the center of mass of the convex hull
centroido = mean(dataori(ko(:),:));

% Calculate the volume of the convex hull
volumeo = 0;
for i=1:size(ko,1)
    volumeo = volumeo + abs(det([dataori(ko(i,1),:)-centroido;...
        dataori(ko(i,2),:)-centroido;...
        dataori(ko(i,3),:)-centroido]))/6;
end

%% Printting
disp(['Elapsed time: ', num2str(toc), ' s']);
disp(['The max bending angle: ', num2str(max(oriBending)), ' deg']);
disp(['The volume of the origami workspace: ', num2str(volumeo), ' mm^3']);

%% Solidworks exportation
points = [oriX' oriY' oriZ'];
k = boundary(points, 1); % Adjust the second parameter as needed for desired detail
borderPoints = points(k, :);
% figure()
% plot3(borderPoints(:,2), borderPoints(:,3), borderPoints(:,1),
'.r','MarkerSize',10);
% view(-90,0)
% grid
```

For RTT Origami Robot:

```
clear; clc; tic

%% 'RTT' Origami Robot Workspace Analysis
% Optimization method --> 'Divide and Conquer'
% Fitness Function --> 'Closed Loop'


%% Input Parameters
xyz0 = [0 0 0]'; % origin of base
da = deg2rad(90); % design angle of waterbomb mechanism
a = 35; % edge of equilateral triangle which is obtained from center points "F" of
links.
D = 60; % length of one link
w = 20; % wide of link
t = w/(2*cos(da)); % hypotenuse of triangle (of waterbomb).
% "t" is required for finding the locations of necessary points

lb14 = deg2rad(10); % lower bound of angle 1-4 of waterbomb
ub14 = deg2rad(170); % upper bound of angles 1-4 of waterbomb
jointlimit = deg2rad(30); % +- joint rotation of angle 2 for waterbomb

lb = deg2rad(90); % lower bound of theta
ub = deg2rad(180); % upper bound of theta
db = deg2rad(2.5); % step angle

%% Main Code
F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

iter = length(lb:db:ub)^3; % number of points
carpalX = zeros(1,iter);
carpalY = zeros(1,iter);
carpalZ = zeros(1,iter);
carpalBending = zeros(1,iter);

i = 0; j = 0;

for theta1 = lb:db:ub
    for theta2 = lb:db:ub
        for theta3 = lb:db:ub
            i = i+1;
            %% Carpal Wrist Forward Kinematics
            % vertexs of midplane "E"
            E1 = [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
            E2 = [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
            E3 = [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];
```

```
            % the unit vector(u) directed from origin of base(B) to origin of top(P)
            vec = cross(E2-E1,E3-E1);
            u = vec/norm(vec);
            P = 2*dot(E1-xyz0,u);
            nB = [1;0;0]; % unit vector perpendicular to B
            r = P./(2*dot(nB,u)); % thrust distance
            nP = ((u*P)/r)-nB; % unit vector perpendicular to P

            r0 = dot(E1-xyz0,u)/u(1); % r0 = r

            beta = -asin(2*u(1)*u(3)); % rotation around y-axis
            alpha = asin((2*u(1)*u(2))/cos(beta)); % rotation around z-axis
            ksi = acos(dot(nP,nB)/(norm(nP)*norm(nB))); % bending angle of the top
platform

            % Defining translation matrix [P] according to the ksi <= 90deg or ksi >
90deg
            if ksi <= pi/2
                PM = [(1+cos(alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            else
                PM = [(1+cos(pi-alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            end

            % End-Effector locations of carpal wrist
            carpalXYZ = r0*PM; % end-effector locations of carpal wrist
            carpalX(i) = carpalXYZ(1); carpalY(i) = carpalXYZ(2); carpalZ(i) =
carpalXYZ(3);
            carpalBending = rad2deg(ksi);

            %% Origami Converting
            %% Leg 1
            % (1: mid) (11: right) (12: left)
            % thrust distances
            r1 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)]'-xyz0,u)/u(1);
            r11 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)-w/4]'-xyz0,u)/u(1);
            r12 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)+w/4]'-xyz0,u)/u(1);

            % base points
            F1 = xyz0+[0 -a/sqrt(3) 0]';
            F11 = xyz0+[0 -a/sqrt(3) w/4]';
            F12 = xyz0+[0 -a/sqrt(3) -w/4]';

            % top points
            P1 = F1 + r1*PM;
            P11 = F11 + r11*PM;
            P12 = F12 + r12*PM;

            % points on upper edge of waterbomb triangle
            ulink2 = (E1-P1)/(norm(E1-P1));
            U1 = P1+ulink2*(D-t*sin(da));
            U11 = P11+ulink2*(D-0.5*t*sin(da));
            U12 = P12+ulink2*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink1 = (E1-F1)/(norm(E1-F1));
```

```
            B1 = F1+ulink1*(D-t*sin(da));
            B11 = F11+ulink1*(D-0.5*t*sin(da));
            B12 = F12+ulink1*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v11 = U11-B11;
            v12 = U12-B12;

            %% Leg 2
            % thrust distances
            r2 = dot([E2(1) E2(2)-a/(2*sqrt(3)) E2(3)+a/2]'-xyz0,u)/u(1);
            r21 = dot([E2(1) E2(2)-a/(2*sqrt(3))+(w/4)*sind(60)
E2(3)+a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r22 = dot([E2(1) E2(2)-a/(2*sqrt(3))-(w/4)*sind(60) E2(3)+a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F2 = xyz0+[0 a/(2*sqrt(3)) -a/2]';
            F21 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) -a/2-(w/4)*cosd(60)]';
            F22 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) -a/2+(w/4)*cosd(60)]';

            % top points
            P2 = F2 + r2*PM;
            P21 = F21 + r21*PM;
            P22 = F22 + r22*PM;

            % points on upper edge of waterbomb triangle
            ulink22 = (E2-P2)/(norm(E2-P2));
            U2 = P2+ulink22*(D-t*sin(da));
            U21 = P21+ulink22*(D-0.5*t*sin(da));
            U22 = P22+ulink22*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink21 = (E2-F2)/(norm(E2-F2));
            B2 = F2+ulink21*(D-t*sin(da));
            B21 = F21+ulink21*(D-0.5*t*sin(da));
            B22 = F22+ulink21*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v21 = U21-B21;
            v22 = U22-B22;

            %% Leg 3
            % thrust distances
            r3 = dot([E3(1) E3(2)-a/(2*sqrt(3)) E3(3)-a/2]'-xyz0,u)/u(1);
            r31 = dot([E3(1) E3(2)-a/(2*sqrt(3))-(w/4)*sind(60) E3(3)-
a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r32 = dot([E3(1) E3(2)-a/(2*sqrt(3))+(w/4)*sind(60) E3(3)-a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F3 = xyz0+[0 a/(2*sqrt(3)) a/2]';
            F31 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) a/2-(w/4)*cosd(60)]';
            F32 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) a/2+(w/4)*cosd(60)]';
```

```
            % top points
            P3 = F3 + r3*PM;
            P31 = F31 + r31*PM;
            P32 = F32 + r32*PM;

            % points on upper edge of waterbomb triangle
            ulink32 = (E3-P3)/(norm(E3-P3));
            U3 = P3+ulink32*(D-t*sin(da));
            U31 = P31+ulink32*(D-0.5*t*sin(da));
            U32 = P32+ulink32*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink31 = (E3-F3)/(norm(E3-F3));
            B3 = F3+ulink31*(D-t*sin(da));
            B31 = F31+ulink31*(D-0.5*t*sin(da));
            B32 = F32+ulink31*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v31 = U31-B31;
            v32 = U32-B32;
            vx = [v11(1) v12(1) v21(1) v22(1) v31(1) v32(1)]; % collision check
vector

            %% Adding Origami Constraints
            % Measuring the angle 1-4 [rad]
            ang1 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v11)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v12)^2)/(2*(0.5*t*sin(da))^2))];
            ang2 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v21)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v22)^2)/(2*(0.5*t*sin(da))^2))];
            ang3 = [acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v31)^2)/(2*(0.5*t*sin(da))^2)),...
                acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v32)^2)/(2*(0.5*t*sin(da))^2))];
            % Determining whether solution ever exists
            if all(vx(:) >= 0) && all(isreal([ang1(:),ang2(:),ang3(:)]))
                % Using the MATLAB function `DnC` to solve the angle 2-3 [rad]
                % Optimization limit intervals for each link

                x1 = DnC_Resch(ang1,da,jointlimit);
                x2 = DnC_Resch(ang2,da,jointlimit);
                x3 = DnC_Resch(ang3,da,jointlimit);

                % Validation
                valid_ang14 = all(lb14 <= [ang1(:), ang2(:), ang3(:)] & [ang1(:),
ang2(:), ang3(:)] <= ub14, 'all');
                valid_x23 = all(pi - jointlimit <= [x1(:), x2(:), x3(:)] & [x1(:),
x2(:), x3(:)] <= pi + jointlimit, 'all');

                if valid_ang14 && valid_x23
                    j = j+1;
                    % End-Effector locations of origami robot
```

```
                    oriXYZ = xyz0+r0*PM;
                    oriX(j) = oriXYZ(1); oriY(j) = oriXYZ(2); oriZ(j) = oriXYZ(3);
                    oriBending(j) = rad2deg(ksi);
                    oriThT(j,:) = rad2deg([theta1 theta2 theta3]);
                    tht3(j,:) = rad2deg(x3(:));
                end
            end

        end
    end
end

%% Plottings
figure(1)
plot3(carpalY,carpalZ,carpalX,'.r','LineWidth',0.05)
hold on
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
plot3(oriY,oriZ,oriX,'.b','LineWidth',0.05)
view(-90,0)
legend('CARPAL','ORIGAMI',Location='southeast')
grid on


figure(2)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
axis([ -40 45 -40 45 0 130 ])
view(145,10)
grid on
h=colorbar;
h.Title.String = 'Bending angle ζ';


figure(3)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
axis([ -40 45 -40 45 0 130 ])
view(0,90)
grid on
h=colorbar;
h.Title.String = 'Bending angle ζ';
%% Volume Calculation
xo = oriY'; yo = oriZ'; zo = oriX';

dataori = [xo,yo,zo]; % Combine the data into a matrix
ko = convhulln(dataori); % Compute the convex hull of the data
centroido = mean(dataori(ko(:),:)); % Calculate the center of mass of the convex hull
% Calculate the volume of the convex hull
volumeo = 0;
```

```
for i=1:size(ko,1)
    volumeo = volumeo + abs(det([dataori(ko(i,1),:)-centroido;...
        dataori(ko(i,2),:)-centroido;...
        dataori(ko(i,3),:)-centroido]))/6;
end

%% Printting
disp(['Elapsed time: ', num2str(toc), ' s']);
disp(['The max bending angle: ', num2str(max(oriBending)), ' deg']);
disp(['The volume of the origami workspace: ', num2str(round(volumeo,0)), ' mm^3']);
disp(['# of points = Carpal: ',num2str(iter),' Origami: ',num2str(j)]);

%% Solidworks Exportation
points = [zo xo yo];
k = boundary(points, 1); % Adjust the second parameter as needed for desired detail
borderPoints = points(k, :);
% figure()
% plot3(borderPoints(:,2), borderPoints(:,3), borderPoints(:,1),
'.r','MarkerSize',10);
% view(-90,0)
% grid
```

For Serial RTT Origami Robot:

```
clear; clc; tic

%% 'Serial RTT' Origami Robot Workspace Analysis
% Optimization method --> 'Divide and Conquer'
% Fitness Function --> 'Open Loop'


%% Input Parameters
xyz0 = [0 0 0]'; % origin of base
da = deg2rad(90); % design angle of waterbomb mechanism
a = 35; % edge of equilateral triangle which is obtained from center points "F" of
links.
D = 60; % length of one link
w = 20; % wide of link
t = w/(2*cos(da)); % hypotenuse of triangle (of waterbomb).
% "t" is required for finding the locations of necessary points

lb14 = deg2rad(0); % lower bound of angle 1-4 of waterbomb
ub14 = deg2rad(180); % upper bound of angles 1-4 of waterbomb
jointlimit = deg2rad(90); % +- joint rotation of angle 2 for waterbomb

lb = deg2rad(90); % lower bound of theta
ub = deg2rad(180); % upper bound of theta
db = deg2rad(10); % step angle

%% Main Code
F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

iter = length(lb:db:ub)^3; % number of points
carpalX = zeros(1,iter);
carpalY = zeros(1,iter);
carpalZ = zeros(1,iter);
carpalBending = zeros(1,iter);

i = 0; j = 0;g=1;

for theta1 = lb:db:ub
    for theta2 = lb:db:ub
        for theta3 = lb:db:ub
            i = i+1;
            %% Carpal Wrist Forward Kinematics
            % vertexs of midplane "E"
            E1 = [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
            E2 = [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
            E3 = [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];
```

```
            % the unit vector(u) directed from origin of base(B) to origin of top(P)
            vec = cross(E2-E1,E3-E1);
            u = vec/norm(vec);
            P = 2*dot(E1-xyz0,u);
            nB = [1;0;0]; % unit vector perpendicular to B
            r = P./(2*dot(nB,u)); % thrust distance
            nP = ((u*P)/r)-nB; % unit vector perpendicular to P

            r0 = dot(E1-xyz0,u)/u(1); % r0 = r

            beta = -asin(2*u(1)*u(3)); % rotation around y-axis
            alpha = asin((2*u(1)*u(2))/cos(beta)); % rotation around z-axis
            ksi = acos(dot(nP,nB)/(norm(nP)*norm(nB))); % bending angle of the top
platform

            % Defining translation matrix [P] according to the ksi <= 90deg or ksi >
90deg
            if ksi <= pi/2
                PM = [(1+cos(alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            else
                PM = [(1+cos(pi-alpha)*cos(beta)) sin(alpha)*cos(beta) -sin(beta)]';
            end

            % End-Effector locations of carpal wrist
            carpalXYZ = r0*PM; % end-effector locations of carpal wrist
            carpalX(i) = carpalXYZ(1); carpalY(i) = carpalXYZ(2); carpalZ(i) =
carpalXYZ(3);
            carpalBending = rad2deg(ksi);

            %% Origami Converting
            %% Leg 1
            % (1: mid) (11: right) (12: left)
            % thrust distances
            r1 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)]'-xyz0,u)/u(1);
            r11 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)-w/4]'-xyz0,u)/u(1);
            r12 = dot([E1(1) E1(2)+a/sqrt(3) E1(3)+w/4]'-xyz0,u)/u(1);

            % base points
            F1 = xyz0+[0 -a/sqrt(3) 0]';
            F11 = xyz0+[0 -a/sqrt(3) w/4]';
            F12 = xyz0+[0 -a/sqrt(3) -w/4]';

            % top points
            P1 = F1 + r1*PM;
            P11 = F11 + r11*PM;
            P12 = F12 + r12*PM;

            % points on upper edge of waterbomb triangle
            ulink2 = (E1-P1)/(norm(E1-P1));
            U1 = P1+ulink2*(D-t*sin(da));
            U11 = P11+ulink2*(D-0.5*t*sin(da));
            U12 = P12+ulink2*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink1 = (E1-F1)/(norm(E1-F1));
```

```
            B1 = F1+ulink1*(D-t*sin(da));
            B11 = F11+ulink1*(D-0.5*t*sin(da));
            B12 = F12+ulink1*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v11 = U11-B11;
            v12 = U12-B12;

            %% Leg 2
            % thrust distances
            r2 = dot([E2(1) E2(2)-a/(2*sqrt(3)) E2(3)+a/2]'-xyz0,u)/u(1);
            r21 = dot([E2(1) E2(2)-a/(2*sqrt(3))+(w/4)*sind(60)
E2(3)+a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r22 = dot([E2(1) E2(2)-a/(2*sqrt(3))-(w/4)*sind(60) E2(3)+a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F2 = xyz0+[0 a/(2*sqrt(3)) -a/2]';
            F21 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) -a/2-(w/4)*cosd(60)]';
            F22 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) -a/2+(w/4)*cosd(60)]';

            % top points
            P2 = F2 + r2*PM;
            P21 = F21 + r21*PM;
            P22 = F22 + r22*PM;

            % points on upper edge of waterbomb triangle
            ulink22 = (E2-P2)/(norm(E2-P2));
            U2 = P2+ulink22*(D-t*sin(da));
            U21 = P21+ulink22*(D-0.5*t*sin(da));
            U22 = P22+ulink22*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink21 = (E2-F2)/(norm(E2-F2));
            B2 = F2+ulink21*(D-t*sin(da));
            B21 = F21+ulink21*(D-0.5*t*sin(da));
            B22 = F22+ulink21*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v21 = U21-B21;
            v22 = U22-B22;

            %% Leg 3
            % thrust distances
            r3 = dot([E3(1) E3(2)-a/(2*sqrt(3)) E3(3)-a/2]'-xyz0,u)/u(1);
            r31 = dot([E3(1) E3(2)-a/(2*sqrt(3))-(w/4)*sind(60) E3(3)-
a/2+(w/4)*cosd(60)]'-xyz0,u)/u(1);
            r32 = dot([E3(1) E3(2)-a/(2*sqrt(3))+(w/4)*sind(60) E3(3)-a/2-
(w/4)*cosd(60)]'-xyz0,u)/u(1);

            % base points
            F3 = xyz0+[0 a/(2*sqrt(3)) a/2]';
            F31 = xyz0+[0 a/(2*sqrt(3))+(w/4)*sind(60) a/2-(w/4)*cosd(60)]';
            F32 = xyz0+[0 a/(2*sqrt(3))-(w/4)*sind(60) a/2+(w/4)*cosd(60)]';
```

```
            % top points
            P3 = F3 + r3*PM;
            P31 = F31 + r31*PM;
            P32 = F32 + r32*PM;

            % points on upper edge of waterbomb triangle
            ulink32 = (E3-P3)/(norm(E3-P3));
            U3 = P3+ulink32*(D-t*sin(da));
            U31 = P31+ulink32*(D-0.5*t*sin(da));
            U32 = P32+ulink32*(D-0.5*t*sin(da));

            % points on bottom edge of waterbomb triangle
            ulink31 = (E3-F3)/(norm(E3-F3));
            B3 = F3+ulink31*(D-t*sin(da));
            B31 = F31+ulink31*(D-0.5*t*sin(da));
            B32 = F32+ulink31*(D-0.5*t*sin(da));

            % vector control (always points U must be >= points B in the x-axis)
            v31 = U31-B31;
            v32 = U32-B32;
            vx = [v11(1) v12(1) v21(1) v22(1) v31(1) v32(1)]; % collision check
vector

            %% Adding Origami Constraints
            % Measuring the angle 1 [rad]
            ang1 = acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v11)^2)/(2*(0.5*t*sin(da))^2));
            ang2 = acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v21)^2)/(2*(0.5*t*sin(da))^2));
            ang3 = acos(((0.5*t*sin(da))^2+(0.5*t*sin(da))^2-
norm(v31)^2)/(2*(0.5*t*sin(da))^2));

            wb_in(i,:) = [ang1 ang2 ang3];

            % Defining fitness function parameters
            rot = [0 -120 -240];% Angle must be in degree

            %Production x-rotation matrix to apply dh table for each link.
            tar1 = rotx(rot(1))*(P1-F1);
            tar2 = rotx(rot(2))*(P2-F2);
            tar3 = rotx(rot(3))*(P3-F3);

            % Determining whether solution ever exists
            if all(vx(:) >= 0) && all(isreal([ang1(1),ang2(1),ang3(1)]))

                % Using the MATLAB function `DnC` to solve the angle 2 [rad]
                x1 = DnC_Serial(da,D,theta1,ang1,tar1);
                x2 = DnC_Serial(da,D,theta2,ang2,tar2);
                x3 = DnC_Serial(da,D,theta3,ang3,tar3);

                wb_out(i,:) = [x1 x2 x3];

                % Validation
                valid_ang1 = all(lb14 <= [ang1, ang2, ang3] & [ang1, ang2, ang3] <=
ub14, 'all');
```

```
                valid_x2 = all(-jointlimit <= [x1, x2, x3] & [x1, x2, x3] <=
jointlimit, 'all');

                if valid_ang1 && valid_x2
                    j = j+1;
                    % End-Effector locations of origami robot
                    oriXYZ = xyz0+r0*PM;
                    oriX(j) = oriXYZ(1); oriY(j) = oriXYZ(2); oriZ(j) = oriXYZ(3);
                    oriBending(j) = rad2deg(ksi);
                    oriThT(j,:) = rad2deg([theta1 theta2 theta3]);
                end
            end

        end
    end
end

%% Plottings

figure(1)
plot3(carpalY,carpalZ,carpalX,'.r','LineWidth',0.05)
hold on
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
plot3(oriY,oriZ,oriX,'.b','LineWidth',0.05)
view(-90,0)
legend('CARPAL','ORIGAMI',Location='southeast')
grid on


figure(2)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
axis([ -35 40 -35 40 0 130 ])
view(145,10)
grid on
h=colorbar;
h.Title.String = 'Bending angle ζ';


figure(3)
scatter3(oriY,oriZ,oriX,10,oriBending,'filled')
xlabel('Y(mm)')
ylabel('Z(mm)')
zlabel('X(mm)')
axis([ -35 40 -35 40 0 130 ])
view(0,90)
grid on
h=colorbar;
h.Title.String = 'Bending angle ζ';
%% Volume Calculation
xo = oriY'; yo = oriZ'; zo = oriX';
```

```
dataori = [xo,yo,zo]; % Combine the data into a matrix
ko = convhulln(dataori); % Compute the convex hull of the data
centroido = mean(dataori(ko(:),:)); % Calculate the center of mass of the convex hull
% Calculate the volume of the convex hull
volumeo = 0;
for i=1:size(ko,1)
    volumeo = volumeo + abs(det([dataori(ko(i,1),:)-centroido;...
        dataori(ko(i,2),:)-centroido;...
        dataori(ko(i,3),:)-centroido]))/6;
end

%% Printting
disp(['Elapsed time: ', num2str(toc), ' s']);
disp(['The max bending angle: ', num2str(max(oriBending)), ' deg']);
disp(['The volume of the origami workspace: ', num2str(round(volumeo,0)), ' mm^3']);
disp(['# of points = Carpal: ',num2str(iter),' Origami: ',num2str(j)]);

%% Solidworks Exportation
points = [zo xo yo];
k = boundary(points, 1); % Adjust the second parameter as needed for desired detail
borderPoints = points(k, :);
% figure()
% plot3(borderPoints(:,2), borderPoints(:,3), borderPoints(:,1),
'.r','MarkerSize',10);
% view(-90,0)
% grid
```

Objective / Fitness Functions

For Waterbomb Origami Robot:

```
function [TH2,TH3] = DnQ_epfl(x,da)

tht1 = x(1);
tht4 = x(2);

a1 = pi-da;
a2 = pi+2*da;
a3 = a1;
a4 = a1;
a5 = a2;
a6 = a1;

theta_s2 = 1*pi/180; %Starting angle of theta
theta_e2 = 179*pi/180; %Ending angle of theta
theta_s3 = 1*pi/180; %Starting angle of theta
theta_e3 = 179*pi/180; %Ending angle of theta

step = 10*pi/180; %Angel of step

t3 = pi - tht4;
t6 = pi - tht1;
div = 5; %Division of step for each iteration
E = 7; %Desired error as power (10^-7)
k= 1;

while 1

    theta2 = theta_s2:step:theta_e2;
    theta3 = theta_s3:step:theta_e3;
    p = (length(theta2))^2; %Number of point.

    i = 0;
    x = zeros(p,1);

    for tht2 =theta_s2:step:theta_e2
        for tht3 =theta_s3:step:theta_e3

            i = i + 1;

            t1 = pi - tht2;
            t2 = pi - tht3;


            T_1_2 = [                                                cos(t1)*cos(t2) -
cos(a2)*sin(t1)*sin(t2),                                               - cos(t1)*sin(t2) -
cos(a2)*cos(t2)*sin(t1),                                    sin(a2)*sin(t1), 0;
                cos(a1)*cos(t2)*sin(t1) - sin(a1)*sin(a2)*sin(t2) +
cos(a1)*cos(a2)*cos(t1)*sin(t2), cos(a1)*cos(a2)*cos(t1)*cos(t2) -
cos(a1)*sin(t1)*sin(t2) - sin(a1)*sin(a2)*cos(t2), - cos(a2)*sin(a1) -
cos(a1)*sin(a2)*cos(t1), 0;
```

```
                cos(a1)*sin(a2)*sin(t2) + sin(a1)*cos(t2)*sin(t1) +
cos(a2)*sin(a1)*cos(t1)*sin(t2), cos(a1)*sin(a2)*cos(t2) - sin(a1)*sin(t1)*sin(t2) +
cos(a2)*sin(a1)*cos(t1)*cos(t2),   cos(a1)*cos(a2) - sin(a1)*sin(a2)*cos(t1), 0;
                0,
0,                                          0, 0];


            T_3_4 =[                                                cos(t3)*cos(t2) -
cos(a4)*sin(t3)*sin(t2),                                              - cos(t3)*sin(t2) -
cos(a4)*cos(t2)*sin(t3),                                  sin(a4)*sin(t3), 0;
                cos(a3)*cos(t2)*sin(t3) - sin(a3)*sin(a4)*sin(t2) +
cos(a3)*cos(a4)*cos(t3)*sin(t2), cos(a3)*cos(a4)*cos(t3)*cos(t2) -
cos(a3)*sin(t3)*sin(t2) - sin(a3)*sin(a4)*cos(t2), - cos(a4)*sin(a3) -
cos(a3)*sin(a4)*cos(t3), 0;
                cos(a3)*sin(a4)*sin(t2) + sin(a3)*cos(t2)*sin(t3) +
cos(a4)*sin(a3)*cos(t3)*sin(t2), cos(a3)*sin(a4)*cos(t2) - sin(a3)*sin(t3)*sin(t2) +
cos(a4)*sin(a3)*cos(t3)*cos(t2),   cos(a3)*cos(a4) - sin(a3)*sin(a4)*cos(t3), 0;
                0,
0,                                          0, 0];

            T_5_6 = [                                               cos(t1)*cos(t6) -
cos(a6)*sin(t1)*sin(t6),                                              - cos(t1)*sin(t6) -
cos(a6)*cos(t6)*sin(t1),                                  sin(a6)*sin(t1), 0;
                cos(a5)*cos(t6)*sin(t1) - sin(a5)*sin(a6)*sin(t6) +
cos(a5)*cos(a6)*cos(t1)*sin(t6), cos(a5)*cos(a6)*cos(t1)*cos(t6) -
cos(a5)*sin(t1)*sin(t6) - sin(a5)*sin(a6)*cos(t6), - cos(a6)*sin(a5) -
cos(a5)*sin(a6)*cos(t1), 0;
                cos(a5)*sin(a6)*sin(t6) + sin(a5)*cos(t6)*sin(t1) +
cos(a6)*sin(a5)*cos(t1)*sin(t6), cos(a5)*sin(a6)*cos(t6) - sin(a5)*sin(t1)*sin(t6) +
cos(a6)*sin(a5)*cos(t1)*cos(t6),   cos(a5)*cos(a6) - sin(a5)*sin(a6)*cos(t1), 0;
                0,
0,                                          0, 0];
            T = T_1_2*T_3_4*T_5_6;

            x(i) = sqrt((1-T(1,1))^2  +    T(1,2)^2    +     T(1,3)^2    +...
                T(2,1)^2    +  (1-T(2,2))^2  +     T(2,3)^2    +...
                T(3,1)^2    +    T(3,2)^2    +   (1-T(3,3)))^2 ;

        end
    end

    [a,b] = min(x);
    e = a;

    tt2 = theta2(ceil(b/sqrt(p)));
    if rem(b,sqrt(p)) == 0
        tt3 = theta3(sqrt(p));
    else
        tt3 = theta3(rem(b,sqrt(p)));
    end

    TH2 = tt2;
    TH3 = tt3;

    if a<10^-E
```

```
        break
    end

    if k == 18
        TH2 = 0;
        TH3 = 0;
        break
    end

    range = 3*step;
    step = step/div;
    theta_s2 = tt2-range;
    theta_e2 = tt2+range;
    theta_s3 = tt3-range;
    theta_e3 = tt3+range;

    k = k+1;
end
```

For RTT Origami Robot:

```
function y = DnC_Resch(ang,da,limit)

tht1 = ang(1);
tht4 = ang(2);

a1 = pi-da;
a2 = pi+2*da;
a3 = a1;
a4 = a1;
a5 = a2;
a6 = a1;

if tht1 >= tht4
    theta_s2 = pi; %Starting angle of theta
    theta_e2 = pi+limit; %Ending angle of theta
    theta_s3 = pi-limit; %Starting angle of theta
    theta_e3 = pi; %Ending angle of theta

elseif tht1 < tht4
    theta_s2 = pi-limit; %Starting angle of theta
    theta_e2 = pi; %Ending angle of theta
    theta_s3 = pi; %Starting angle of theta
    theta_e3 = pi+limit; %Ending angle of theta

end
step = 10*pi/180; %Angel of step

t3 = pi - tht4;
t6 = pi - tht1;
div = 5; %Division of step for each iteration
E = 7; %Desired error as power
k= 1;
```

```
while 1

    theta2 = theta_s2:step:theta_e2;
    theta3 = theta_s3:step:theta_e3;
    p = (length(theta2))^2; %Number of point.

    i = 0;
    x = zeros(p,1);

    for tht2 =theta_s2:step:theta_e2
        for tht3 =theta_s3:step:theta_e3

            i = i + 1;

            t1 = pi - tht2;
            t2 = pi - tht3;


            T_1_2 = [                                                cos(t1)*cos(t2) -
cos(a2)*sin(t1)*sin(t2),                                               - cos(t1)*sin(t2) -
cos(a2)*cos(t2)*sin(t1),                                  sin(a2)*sin(t1), 0;
                cos(a1)*cos(t2)*sin(t1) - sin(a1)*sin(a2)*sin(t2) +
cos(a1)*cos(a2)*cos(t1)*sin(t2), cos(a1)*cos(a2)*cos(t1)*cos(t2) -
cos(a1)*sin(t1)*sin(t2) - sin(a1)*sin(a2)*cos(t2), - cos(a2)*sin(a1) -
cos(a1)*sin(a2)*cos(t1), 0;
                cos(a1)*sin(a2)*sin(t2) + sin(a1)*cos(t2)*sin(t1) +
cos(a2)*sin(a1)*cos(t1)*sin(t2), cos(a1)*sin(a2)*cos(t2) - sin(a1)*sin(t1)*sin(t2) +
cos(a2)*sin(a1)*cos(t1)*cos(t2),   cos(a1)*cos(a2) - sin(a1)*sin(a2)*cos(t1), 0;
                0,
0,                                          0, 0];


            T_3_4 =[                                                cos(t3)*cos(t2) -
cos(a4)*sin(t3)*sin(t2),                                               - cos(t3)*sin(t2) -
cos(a4)*cos(t2)*sin(t3),                                  sin(a4)*sin(t3), 0;
                cos(a3)*cos(t2)*sin(t3) - sin(a3)*sin(a4)*sin(t2) +
cos(a3)*cos(a4)*cos(t3)*sin(t2), cos(a3)*cos(a4)*cos(t3)*cos(t2) -
cos(a3)*sin(t3)*sin(t2) - sin(a3)*sin(a4)*cos(t2), - cos(a4)*sin(a3) -
cos(a3)*sin(a4)*cos(t3), 0;
                cos(a3)*sin(a4)*sin(t2) + sin(a3)*cos(t2)*sin(t3) +
cos(a4)*sin(a3)*cos(t3)*sin(t2), cos(a3)*sin(a4)*cos(t2) - sin(a3)*sin(t3)*sin(t2) +
cos(a4)*sin(a3)*cos(t3)*cos(t2),   cos(a3)*cos(a4) - sin(a3)*sin(a4)*cos(t3), 0;
                0,
0,                                          0, 0];

            T_5_6 = [                                                cos(t1)*cos(t6) -
cos(a6)*sin(t1)*sin(t6),                                               - cos(t1)*sin(t6) -
cos(a6)*cos(t6)*sin(t1),                                  sin(a6)*sin(t1), 0;
                cos(a5)*cos(t6)*sin(t1) - sin(a5)*sin(a6)*sin(t6) +
cos(a5)*cos(a6)*cos(t1)*sin(t6), cos(a5)*cos(a6)*cos(t1)*cos(t6) -
cos(a5)*sin(t1)*sin(t6) - sin(a5)*sin(a6)*cos(t6), - cos(a6)*sin(a5) -
cos(a5)*sin(a6)*cos(t1), 0;
```

```
                cos(a5)*sin(a6)*sin(t6) + sin(a5)*cos(t6)*sin(t1) +
cos(a6)*sin(a5)*cos(t1)*sin(t6), cos(a5)*sin(a6)*cos(t6) - sin(a5)*sin(t1)*sin(t6) +
cos(a6)*sin(a5)*cos(t1)*cos(t6),   cos(a5)*cos(a6) - sin(a5)*sin(a6)*cos(t1), 0;
                0,
0,                                          0, 0];
            T = T_1_2*T_3_4*T_5_6;

            x(i) = sqrt((1-T(1,1))^2  +    T(1,2)^2    +     T(1,3)^2    +...
                T(2,1)^2    +  (1-T(2,2))^2  +     T(2,3)^2    +...
                T(3,1)^2    +    T(3,2)^2    +   (1-T(3,3)))^2 ;

        end
    end

    [a,b] = min(x);
    e=a;

    tt2 = theta2(ceil(b/sqrt(p)));
    if rem(b,sqrt(p)) == 0
        tt3 = theta3(sqrt(p));
    else
        tt3 = theta3(rem(b,sqrt(p)));
    end

    y = [tt2,tt3];

    if a<10^-E
        break
    end

    if k == 25
        TH2 = 0;
        TH3 = 0;
        break
    end

    range = 3*step;
    step = step/div;
    theta_s2 = tt2-range;
    theta_e2 = tt2+range;
    theta_s3 = tt3-range;
    theta_e3 = tt3+range;

    k=k+1;
end
```

For Serial RTT Origami Robot:

```
function TH_13 = DnC_Serial(da,D,th_in,ang,tar)
%y=waterbom angle 2-6
%tar = Position of the link's top point

th_2 = -(pi - ang);%DH table form
th_13_s = -pi/2; %Starting angle of theta
th_13_e =  pi/2; %Ending angle of theta
step = deg2rad(10); %Angel of step

div = 5; %Division of step for each iteration
E = 7; %Desired error as power
k= 1;
while 1

    theta_13 = th_13_s:step:th_13_e;
    p = length(theta_13); %Number of point.
    i = 0;
    xx = zeros(p,1);

    for th_13 =th_13_s:step:th_13_e

            i = i + 1;
            T = TM_Serial(da,D,th_in,th_13,th_2); %Transfromation matrix of serail
robot
            xx(i) = sqrt((tar(1)-T(1,4))^2 + (tar(2)-T(2,4))^2 + (tar(3)-T(3,4))^2) ;
    end

    [a,b] = min(xx);
    e=a;

    if rem(b,p) == 0
        TH_13 = theta_13(p);
    else
        TH_13 = theta_13(rem(b,p));
    end

    if a<10^-E
        break
    end

    if k == 25
        TH_13 = -pi;
        break
    end

    range = 3*step;
    step = step/div;
    th_13_s = TH_13-range;
    th_13_e = TH_13+range;

    k=k+1;

end
```

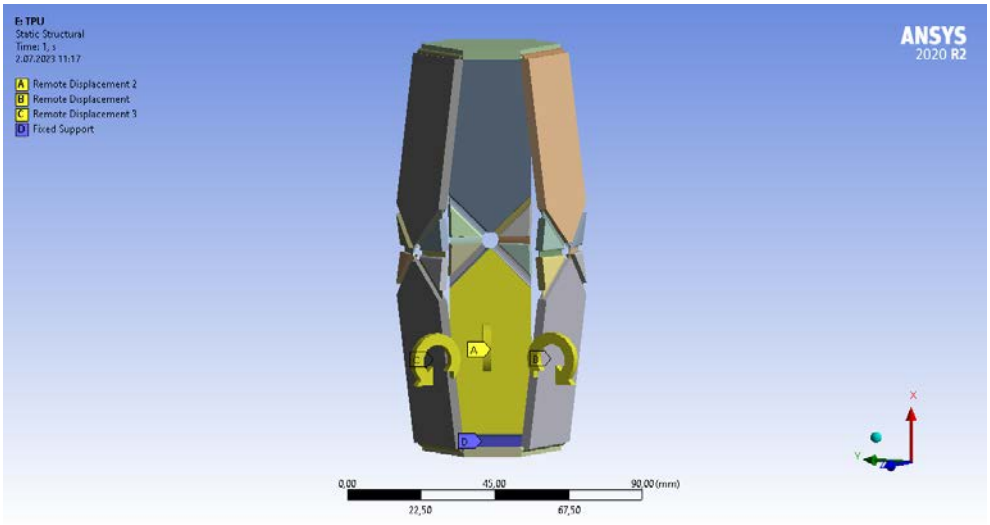
Finding Transformation Matrix for Serial RTT:

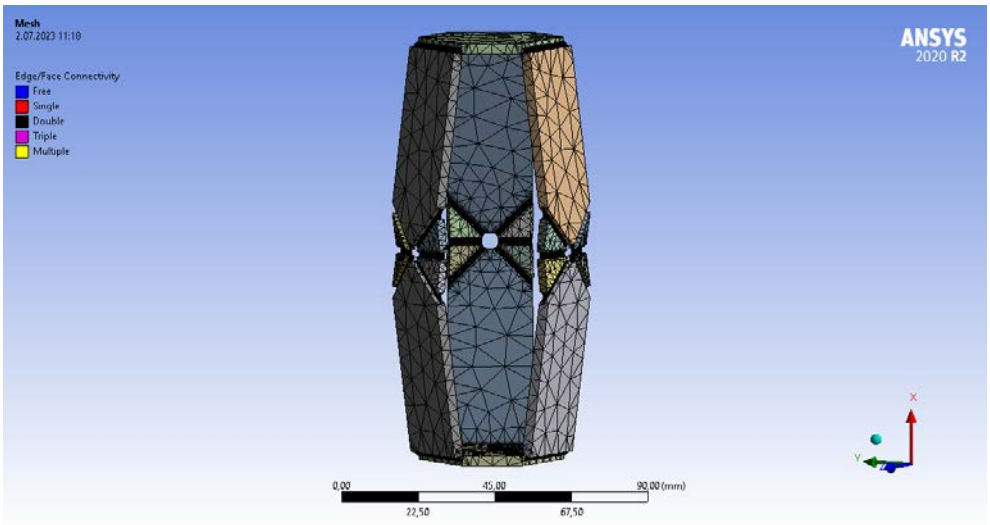
```
function T_total = TM_Serial(da,L,th_in,th_13,th_2)

      %Dh table of links
     %alfa            a       theta         d
th = [  0             0       pi-th_in      0;
       pi/2           0        0            L;
      pi/2-da         0       th_13         0
        da            0       th_2          0;
        da            0       th_13         0;
     -(pi/2+da)       0        0            L];


T_i_1 = eye(4);
for i=1:length(th)


T_i = [cos(th(i,3))                -sin(th(i,3))                0
th(i,2) ;
       cos(th(i,1))*sin(th(i,3))    cos(th(i,3))*cos(th(i,1))   -sin(th(i,1))    -
sin(th(i,1))*th(i,4);
       sin(th(i,3))*sin(th(i,1))    sin(th(i,1))*cos(th(i,3))    cos(th(i,1))
cos(th(i,1))*th(i,4);
       0                            0                            0               1
];

T_total = T_i_1*T_i;
T_i_1 = T_total;
end
```

APPENDIX B: Details of Finite Element Analysis

Finite Element Analysis (FEA) is performed by sequentially defining the geometry, mesh, determining the material properties (linear or nonlinear), and setting up the analysis parameters (linear or nonlinear solving) to accurately simulate and evaluate the structural behavior of a system.

- Material properties are taken from ANSYS engineering materials library.



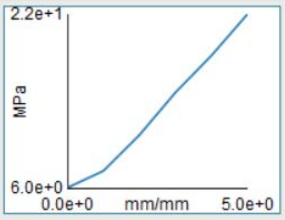


TPU_15

Density	1,178e-06 kg/mm³

Structural

Isotropic Elasticity

Derive from	Young's Modulus and Poisson's Ratio
Young's Modulus	19,090 MPa
Poisson's Ratio	0,38970
Bulk Modulus	28,846 MPa
Shear Modulus	6,8684 MPa
Multilinear Isotropic Hardening	
Tensile Ultimate Strength	21,500 MPa
Tensile Yield Strength	6,0000 MPa

Plastic, PLA

PLA (general purpose), Polylactide / Polylactic acid (General purpose)

Data compiled by the Granta Design team at ANSYS, incorporating various sources including JAHM and MagWeb.
ANSYS Inc. provides no warranty for this data.

Density	1,255e-06 kg/mm³

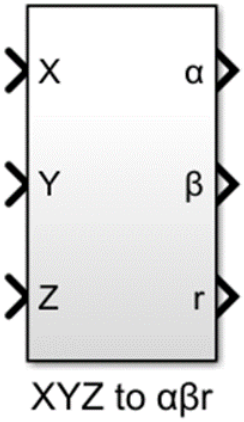
Structural

Isotropic Elasticity

Derive from	Young's Modulus and Poisson's Ratio
Young's Modulus	3447,0 MPa
Poisson's Ratio	0,38990
Bulk Modulus	5218,0 MPa
Shear Modulus	1240,0 MPa
Isotropic Secant Coefficient of Thermal Expansion	0,00013520 1/K
Tensile Ultimate Strength	62,930 MPa
Tensile Yield Strength	52,440 MPa

APPENDIX C: Simulink Control System Diagram

XYZ to αβr Block: It makes a conservation from xyz position of top platform to αβr values of the top platform.

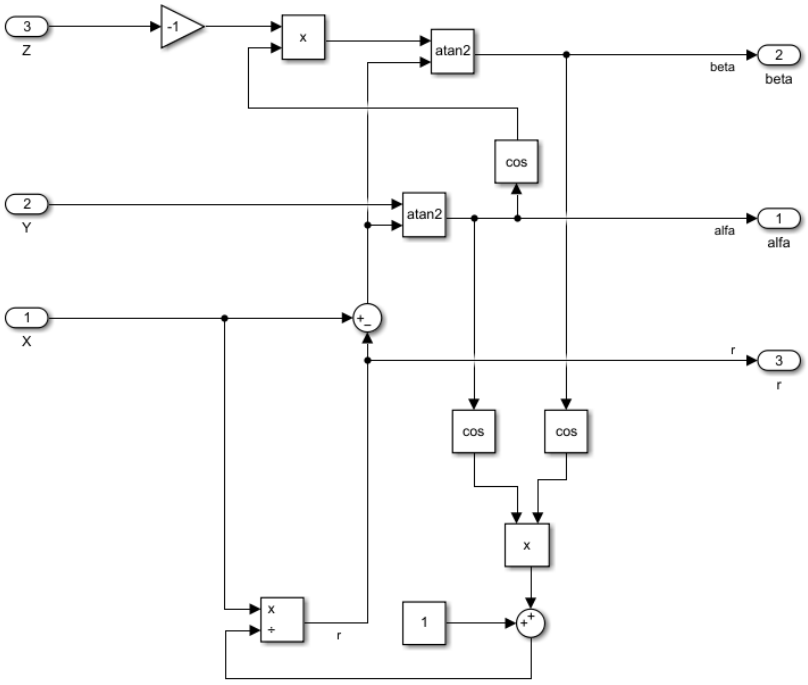


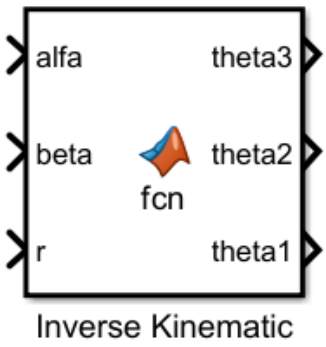
Inputs:

- X in mm
- Y in mm
- Z in mm

Outputs:

- α (alfa) in radian
- β (beta) in radian
- r in mm





Inverse Kinematic Function Block: Values of α, β, and r must be provided as input so that angle of each motor can be achieved as output. All inverse kinematic steps are included on the block.

Inputs:

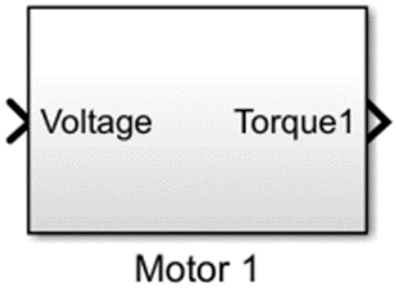
- α (alfa) in radian
- β (beta) in radian
- r in mm

Outputs:

- theta1 (angle of DC Motor 1) in radian
- theta2 (angle of DC Motor 2) in radian
- theta3 (angle of DC Motor 3) in radian

```
function [theta3,theta2,theta1] = fcn(alfa,beta,r)

n_p = [cos(alfa)*cos(beta);  %x direction of top plane
       sin(alfa)*cos(beta);  %y direction of top plane
           -sin(beta)    ];  %z direction of top plane

D = 35; % length of one link
a = 20; % edge of equilateral triangle which is obtained from center points "F" of
links.
xyz0 = [0 0 0]'; % origin of base

phi = [pi ; 5*pi/3 ; pi/3];% These are constant value for symmetric position of link
on the base.

F    =    [xyz0(1)    xyz0(1)    xyz0(1);    xyz0(2)-a/sqrt(3)    xyz0(2)+a/(2*sqrt(3))
xyz0(2)+a/(2*sqrt(3)); xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

V = zeros(3,1);
Y = zeros(3,1);
Z = zeros(3,1);

for i=1:3
    V(i) = r*(-L*cos(phi(i))*n_p(2) - L*sin(i(i))*n_p(3));
    Y(i) = r*L*(n_p(1) + 1);
    Z(i) = r*dot([F(1,i) ; F(2,i) ; F(3,i)] , [(n_p(1) + 1) ; n_p(2) ; n_p(3)]) -
r^2*((n_p(1) + 1)) ;
end
theta = zeros(3,1);
for j = 1:3
```

```
    theta(j) = 2*atan((-Y(j) - sqrt(Y(j)^2 - Z(j)^2 + V(j)^2))/(Z(j) - V(j)));
end

theta1 = rad2deg(theta(1));
theta2 = rad2deg(theta(2));
theta3 = rad2deg(theta(3));
```



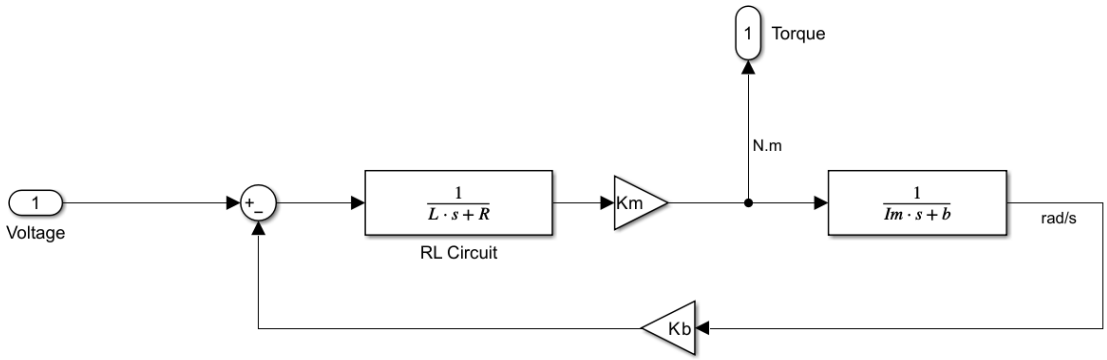
Motor 1 Block: It includes a block model of DC motor which is used to give motion to each leg. Motor 2 and Motor 3 block have the same block body as Motor 1.

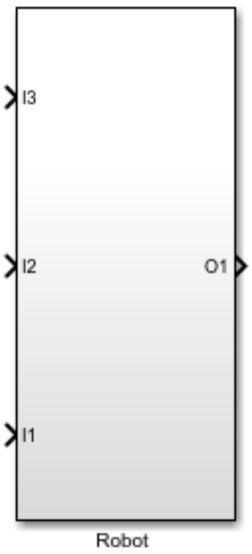
Input:

- Voltage (V)

Output:

- Torque (N.m)





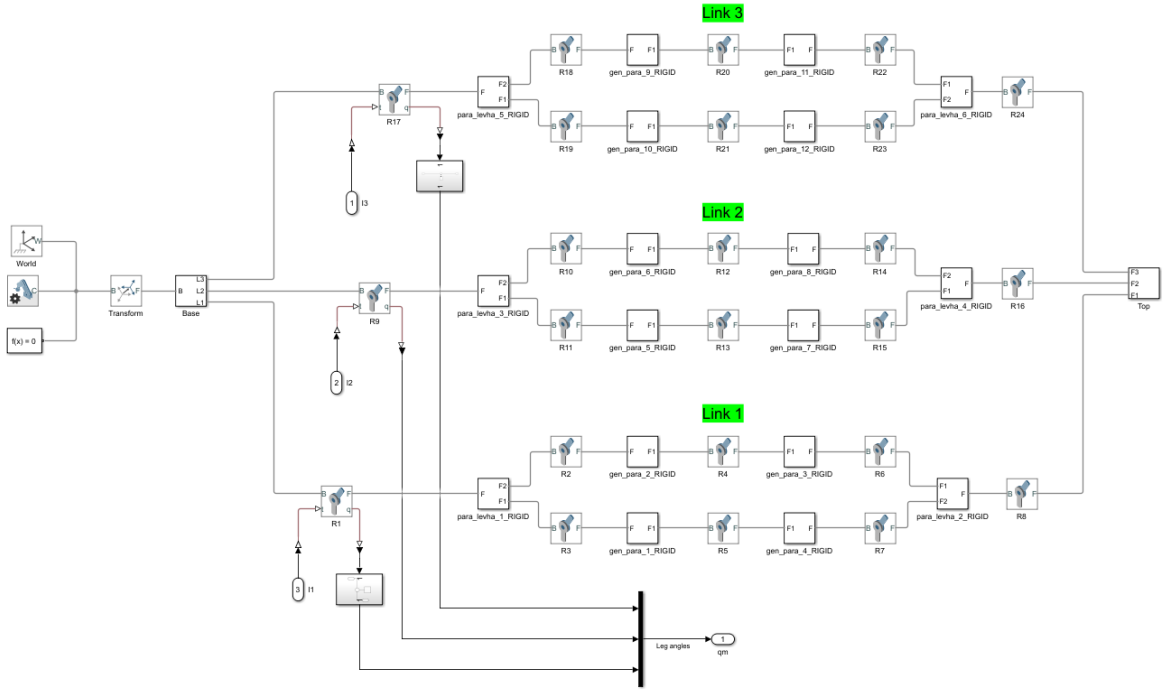
Robot Block: The parallel robots were designed in CAD program. The CAD models were transferred to SIMULINK via SimMechanics tool of CAD program. Sledworks was used as a CAD program.

Inputs:

- I1 in N.m
- I2 in N.m
- I3 in N.m

Output:

- O1 in radian (It includes each link angle separately as output)



Forward Kinematic Block: Measured input angle from each leg is used to obtain top platform position in terms of cartesian (xyz) coordinate and αβr.

Inputs:

- theta1 in radian
- theta2 in radian
- theta3 in radian

Outputs:

- xyz_pos in mm (Cartesian coordinate)
- alfa_beta_r in radian-radian-mm(2 rotation 1 translation)

```
function [xyz_pos, alfa_beta_r] = fcn(theta3,theta2,theta1)

D = 35; % length of one link
a = 20; % edge of equilateral triangle which is obtained from center points "F" of
links.
xyz0 = [0 0 0]'; % origin of base

F = [xyz0(1) xyz0(1) xyz0(1); ...
    xyz0(2)-a/sqrt(3) xyz0(2)+a/(2*sqrt(3)) xyz0(2)+a/(2*sqrt(3)); ...
    xyz0(3) xyz0(3)-0.5*a xyz0(3)+0.5*a];

phi = deg2rad([180;300;60]); % rotation vector (along links)

E1          =          [F(1,1);F(2,1);F(3,1)]+D*[sin(theta1);-cos(theta1)*cos(phi(1));-
cos(theta1)*sin(phi(1))];
E2          =          [F(1,2);F(2,2);F(3,2)]+D*[sin(theta2);-cos(theta2)*cos(phi(2));-
cos(theta2)*sin(phi(2))];
E3          =          [F(1,3);F(2,3);F(3,3)]+D*[sin(theta3);-cos(theta3)*cos(phi(3));-
cos(theta3)*sin(phi(3))];

u = cross((E2 - E1),(E3 - E1))/norm(cross((E2 - E1),(E3 - E1))); %Direction vector of
P
u_x = u(1);
u_y = u(2);
u_z = u(3);

r = dot(E1,u)/u_x;
beta = asin(-2*(u_z*u_x));
alfa = asin(2*(u_y*u_x)/cos(beta));
x = r*(1+cos(alfa)*cos(beta));
y = r*sin(alfa)*cos(beta);%+6.93;
z = -r*sin(beta);
alfa_beta_r = [rad2deg(alfa);rad2deg(beta);r];
xyz_pos = [x,y,z];
```