DESIGN OF A MICRO-MIRROR COMPLIANT MECHANISM*

Winter 2026 MAE C294A

Liliana Figueroa Perez, Ryan Fukunaga, Ethan Freed, Nathan Ge, Avi Gerber

***Abstract*— This paper explores flexure-based micro-mirror design through a review of pre-existing approaches and a collaborative try at micro-mirror topology synthesis. More specifically, our flexure-based micro-mirror design can translate (i.e. piston) up and down to be used to modulate the phase of light reflected off from its surface for a variety of high-impact precision applications (e.g., adaptive optics). Such mirrors are often driven to operate at their resonant frequencies to minimize the system's required actuation energy and to simplify the mirror's control such that no sensors or closed-loop control circuitry is necessary to achieve the desired speed. Therefore, our goal is to use topology synthesis as the design basis for a flexure-based compliant mechanism that acts as a precise, purely translational pistoning mirror stage operating at high speeds.**

This paper introduces micromirror systems and its technical background, our micro-mirror design objectives and the associated design approach (FACT), analytical validation of modal frequencies, and concludes with our design's intended fabrication approach.

I. BACKGROUND

A. Introduction

At the core of modern optical systems exist tiny movable mirrors; micromirrors coupled together on chips to form Micromirror Arrays (MMA). Micromirrors are Micro-Electro-Mechanical System (MEMS), or mechanisms that may include springs and levers with electronics that operate on the microscopic level [1]. The scale and precision at which these systems operate creates its own unique set of engineering challenges especially in fabrication. Fundamentally, each small mirror moves precisely in order to manipulate light waves in useful ways, such as steering, changing its direction, or adjusting its phase. The precise light manipulation capabilities of this class of MEMS have proven to be extremely valuable to a wide range of technological fields, such as high-speed internet, telescopes/astronomy, and the medical field with advanced biological imaging.

A notable modern use case is Light Detection and Ranging (LiDAR) for self-driving cars. Self-driving cars, such as Waymo, use LiDAR sensing systems as their primary "eyes" [2]. LiDAR sends rapid laser pulses and captures the time of reflection off surrounding objects and its return path (time of flight). The time of flight of dispersed laser beams uniformly distributed and sampled across a spherical space generates a point cloud that represents the external surfaces of objects in the local vicinity. This helps create a detailed and precise 3D mapping of the surrounding environment and its existing features, including vehicles and people, enabling collision avoidance and precise navigation. By using MEMS mirrors to quickly and accurately steer the laser beam, the system can scan and interpret an entire field at speeds required for navigation control.

B. Technical Breakdown

On a technical level, the vast majority of micromirrors are compliant mechanisms. They consist of a rigid stage, or the mirror, connected to a fixed ground by flexures. Rather than rigid mechanisms that transfer or redirect motion and energy through rigid interlocking structures, compliant mechanisms use elastic deformation to create motion through the flexures. At the micro-scale, compliant mechanisms are preferred over traditional rigid-body joints as they eliminate friction and wear, thereby mitigating non-linear effects such as hysteresis. These parasitic phenomena are significantly more pronounced at smaller scales, where surface forces dominate over inertial forces. Consequently, the inherent precision of compliant architectures is essential for meeting the stringent motion requirements of micro-scale applications. As long as desired compliant movement of the mechanism falls within the elastic regime of the material, smooth, precise, repeatable motion is achieved even at high speeds. The micromirror's function must guarantee movement permittance and subsequent unwanted movement rejection, high bandwidth tracking, minimizing unintended behaviors, and a reliable mode of actuation.

The flexures are designed to be stiff in some directions and compliant in others. Compliant modes of movement define the mechanism's degrees of freedom (DOF). Our micromirror is designed to move with a single specific translational DOF, piston motion (move purely up and down, translating perfectly orthogonal to its reflective surface). This movement changes the distance that light travels before it is reflected, or optical path length, the phase of the light wave. DOF permittance can be procedurally tuned using the FACT method, outlined in the topology portion of the paper.

As mentioned before, micromirrors must achieve precise movements at high speeds. Micromirrors are often applied in situations that require reference signal tracking and disturbances rejection at incredible speeds. We need a high-bandwidth system to follow fast-changing inputs. At the core physical principles, this means a system that has high mechanical resonant frequency ($f = \sqrt{\frac{k}{m}}$) to enable high speed/ high acceleration tracking while avoiding resonance amplification at low frequency movements. Lowering dynamic mass (m) and increasing damping (k) are crucial in achieving this effect.

One of the most prominent challenges is parasitic tilt, which is unwanted rotational piston movement. This often

occurs when there are uncontrolled degrees of freedom within the flexures if they are not stiff enough. In order to minimize this, designers use complex geometries such as double S-shaped unimorph actuators to maintain proper alignment.

To drive compliant mechanisms, actuation methods must not counteract the design strengths of compliant mechanisms, which is to enable smooth, precise, repeatable motion. This is why non-contact actuation methods such as electrostatic plates, comb drives, or thermal beams are used to drive MEMS micromirrors. When a voltage is applied to the system, the actuators create a force that either pushes or pulls the mirror, which then causes the flexures to bend and move the mirror to its intended position.

C. Case study: The Fraunhofer Mirror

A relevant existing design that focuses on achieving a single piston DOF using resonant actuation is the Fraunhofer IPMS Translatory Mirror (fig. 1), also known as the "Translatory MEMS Mirror with Extraordinary Large Stroke (Pump Mode)" [3]. This device was created to provide fast modulation of optical path length for high-precision optical metrology systems.

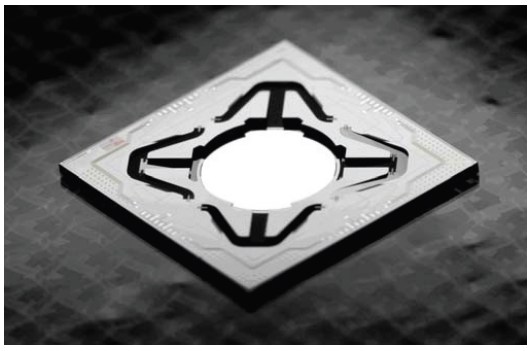

Fig. 1: Fraunhofer IPMS Translatory Mirror

The Fraunhofer mirror is designed to move up and down very quickly, changing the distance light travels before returning. This creates a controlled phase shift in the reflected light. It is essentially a small high-speed optical ruler that can make extremely precise measurements.

Axially symmetric pantograph suspension flexures (which are categorically a collection of wire flexures and blade flexures) work together to guide the mirror and maintain its orientation as it moves up and down. The flexures are compliant in the piston direction but are also stiff in all other directions in order to ensure there are no unwanted rotational movements.

This symmetric design helps achieve large vertical displacements while still being able to maintain stable and predictable motion. Additionally, the mirror is designed to undergo high vibrations and shock insensitivity. This makes it a great candidate for both laboratory and industrial applications such as medical imaging and LiDAR.

Instead of having the mirror at static positions, the Fraunhofer design applies a harmonic driving signal tuned specifically to the natural frequency of the mirror's piston mode. When the driving signal matches the natural frequency, the amplitude of the mirror's motion is significantly amplified. Essentially, the mirror leverages its own natural vibration to create more motion. Subsequently a high mechanical advantage with minimal power usage not possible using standard actuators is achieved. However, resonant designs come with its own set of problems, such as a restriction to sinusoidal patterns and an inability to struggle to hold static positions. They are better suited for continuous repetitive motion.

II. TOPOLOGY

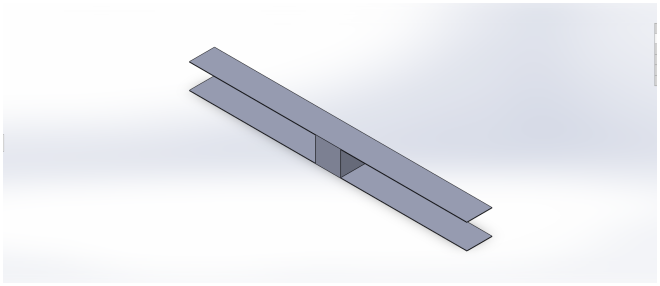

Fig. 2: Micro-mirror Design

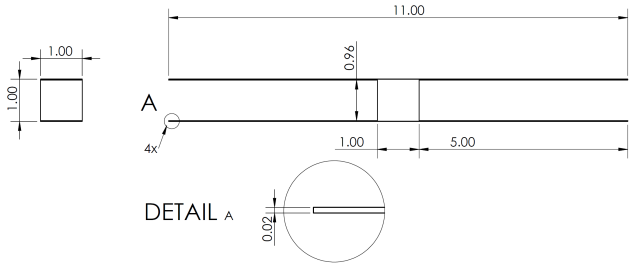

Fig. 3: Micro-mirror Engineering Drawing

The determination of flexure position, orientation, and type selection was governed by a hierarchy of design requirements centered on the moving stage.

A. Design Objectives

The primary objective of the system is to permit the mirror to achieve a single translational *pistoning* motion. Secondary objectives, which ensure real-world performance and structural integrity, are detailed in Table **??**.

B. Topology Synthesis via FACT

The topology synthesis pipeline follows the *Freedom and Constraint Topology* (FACT) approach. Guidelines for the mode of constraint and freedom (DOFs) were sourced from the FACT library, which categorizes compliant mechanisms into specific groupings. The desired movement pattern aligns with the **1 DOF Type 3** freedom space. The associated

Objective	Reasoning
Maximize f_n (1st mode)	Allows the mirror to resonate at the highest possible operating speed.
Frequency Separation	All parasitic natural frequencies (rad/sec) must be at least one order of magnitude ($10\times$) larger than the first natural frequency to prevent unwanted DOFs.
Maximize Modal Gap	Prevents cross-excitation of unwanted modes during high-speed operation, ensuring dynamic stability.
Minimize Parasitic Error	Ensures light is not redirected by unintended tipping or tilting motions during large-range translation.
Manufacturing Feasibility	Increases likelihood of design adoption and production at scale.

TABLE I: Design Objectives and Engineering Rationale

freedom and constrain spaces are shown in figs. 4 and 5 respectively. Flexures were positioned in accordance with the associated constraint space of this grouping. Serial and hybrid systems were disregarded in favor of a simpler parallel system to adhere to the "Keep It Simple, Stupid" (KISS) principle.

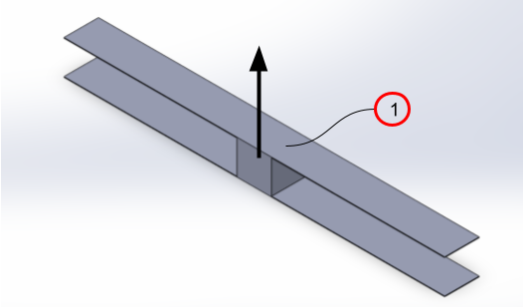

Fig. 4: Freedom space: One Translation

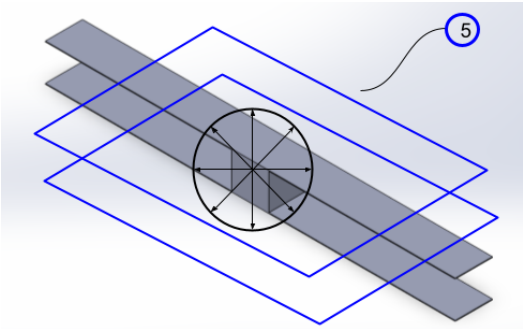

Fig. 5: Constraint Space: Black Moment Sphere, 2 Blue Planes

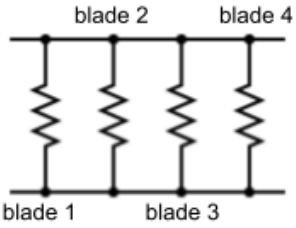

Fig. 6: Parallel stiffness diagram

C. Constraint Analysis

The use of multiple blade flexures results in an overconstrained system. Each blade flexure provides an order of constraint of 3. For a system with 4 blades:

$$C_{total} = 4 \times 3 = 12$$

Given that the target constraint space for this DOF is 5, the degree of overconstraint (O) is calculated as:

$$O = C_{total} - DOF_{constraint} = 12 - 5 = 7$$

The system is overconstrained by 7 degrees of freedom. This trade-off is accepted to maximize off-axis stiffness and maintain the required symmetry for high-speed precision.

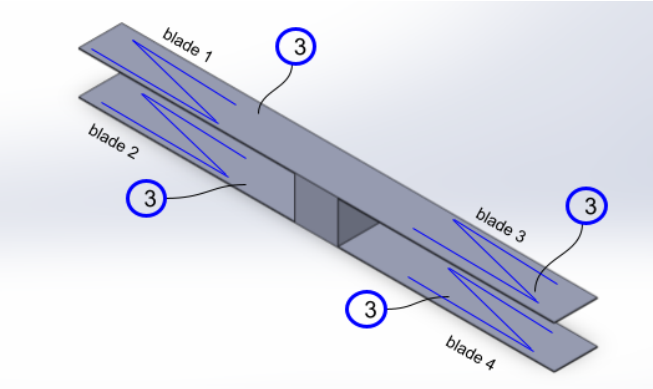

Fig. 7: Combined Order of Constraint

D. System Architecture and Redundancy

The resulting compliant system consists of two pairs of blade flexures placed on opposing sides of a stage block. This symmetric, redundant structure provides several advantages:

- **Buckling Mitigation:** Redundant structures preserve directional compliance while significantly improving load capacity.
- **Dynamic Bandwidth:** Stiffness is increased without a proportional increase in mass. As the natural frequency is defined by $\omega_n = \sqrt{k/m}$, this results in a higher dynamic bandwidth.
- **Error Attenuation:** Axial symmetry attenuates parasitic error and thermal instability. The redundancy corrects the natural arcing path of a single blade into true linear translation.
- **Thermal Robustness:** Symmetrical expansion rates ensure the stage remains on the desired path despite temperature fluctuations.

III. MODAL ANALYSIS

A. Calculations

The dynamic behavior of the proposed micro-mirror system was analyzed using a MATLAB model. The stiffness matrix of the flexure system was generated using the provided `EulerStiffnessMatrix` function, which uses twist-wrench vector analysis to generate the system stiffness matrix for each blade flexure based on its geometry, orientation, and material properties. The mirror stage mass matrix was constructed directly from the stage dimensions, density, and corresponding mass moments of inertia.

The stage mass matrix was assembled using a coordinate transformation of the form

$$M = N\,\Delta\,I\,N^{-1}$$

where N is the transformation matrix, Δ represents the mass distribution, and I is the inertia matrix.

With both matrices defined, the natural frequencies and mode shapes were found by solving for the system eigenvalues using

$$M^{-1}K\,x = \lambda x$$

where $\lambda = \omega^2$. Taking the square root of the eigenvalues gives the angular natural frequencies of the system, which is then divided by 2π to convert from units of rad/s to Hz. This allowed the primary piston mode to be identified and the separation between the first and higher-order modes to be evaluated, ensuring that unwanted modes remain sufficiently suppressed relative to the desired motion. The first 2 calculated resonant frequencies and corresponding mode shapes are

$$\omega_1 = 559.3 \text{ Hz}, \quad \mathbf{x}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1\,\text{m} \\ 0 \end{bmatrix}$$

$$\omega_2 = 27965.0 \text{ Hz}, \quad \mathbf{x}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1\,\text{m} \\ 0 \\ 0 \end{bmatrix}$$

B. FEA Verification

The first two resonant frequencies obtained from the MATLAB calculations are 559.3 Hz and 27,965.0 Hz, while the FEA simulation yields corresponding values of 543.5 Hz and 6008.6 Hz. In both cases, the second natural frequency is more than 10 times greater than the first, with the FEA simulation yielding a 11x increase, and the Matlab simulation as large as 50x. This large separation indicates that the system is dominated by a single mode of motion, confirming that the mirror primarily operates with one degree of freedom and is unlikely to experience significant motion in unintended directions.

Additionally, the first mode shape eigenvector from the analytical calculation corresponds to an upward translation in the positive y-direction, which is normal to the blade flexure surfaces. This behavior is consistent with the intended design and is visually confirmed in Figure 8.

The first resonant frequency has a percent difference of approximately 2.8%, while the second has a percent difference of approximately 78.5%. The second frequency is less critical since the system is designed to operate with a single degree of freedom. The large discrepancy in the second resonant frequency may be due to the mass of the blade flexures not being accounted for in the MATLAB model. Additionally, it is worth noting that in the SolidWorks simulation for the second mode, the blade flexures appear to pass through one another (Figure 9), indicating a non-physical deformation behavior in that mode.

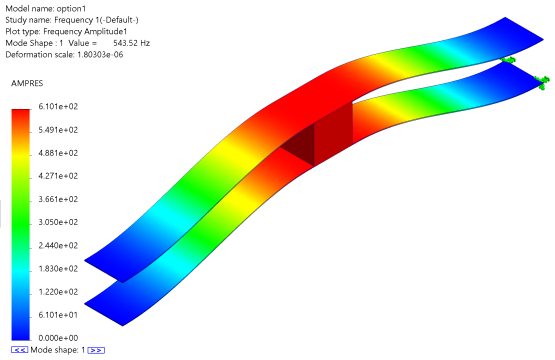

Fig. 8: Mode Shape 1

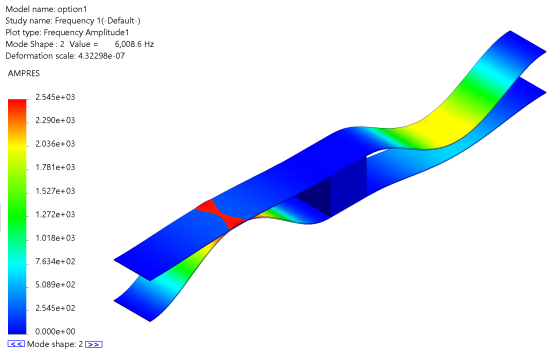

Fig. 9: Mode Shape 2

IV. FABRICATION APPROACH

A. Procedure

To achieve the microscale precision required for high-frequency mirror translation, a multi-step surface and bulk micromachining process is employed.

1) Substrate Preparation: A standard Silicon-on-Insulator (SOI) wafer is prepared with a specific device layer thickness corresponding to the desired flexure height.
2) Photolithography: A photoresist layer is spin-coated and patterned via UV exposure to define the mirror plate and flexure geometries.

3) Anisotropic Etching: DRIE is utilized to etch through the device layer, ensuring vertical sidewalls for the blade flexures.
4) Secondary Patterning: A second photoresist layer is applied and patterned to define the electrode and piezoelectric regions.
5) Piezoelectric Integration: The active layers are deposited via sputtering or Chemical Solution Deposition (CSD).

The fabricated version of the flexure with its base is shown in fig. 10.

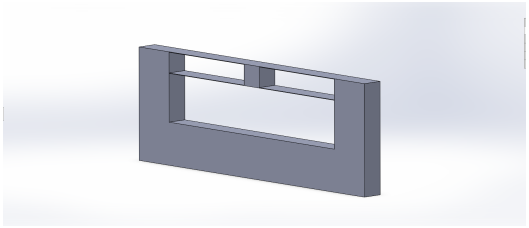

Fig. 10: Form of fabricated flexure

B. Piezoelectric Actuation Stack

The system utilizes non-contact piezoelectric actuation. The material is poled in the d_{31} direction; specifically, an electric field applied across the thickness of the beam (the transverse direction) induces a mechanical strain in the orthogonal longitudinal direction, creating a torsional bending in the flexure blade. Unified bending of all 4 flexures blades in a coordinated direction drives "pistoning" motion.

The thin-film stack is deposited onto the single-crystal silicon device layer in the following order (from top to bottom):

- **Top Electrode:** Platinum (Pt), Gold (Au), or Aluminum (Al)
- **Piezoelectric Film:** Lead Zirconate Titanate (PZT) or Aluminum Nitride (AlN)
- **Bottom Electrode:** Platinum (Pt), Molybdenum (Mo), or Titanium/Platinum (Ti/Pt)
- **Device Layer:** Single-crystal Silicon (Si) — *Forms the beams and mirror plate*
- **Buried Oxide (BOX):** Silicon Dioxide (SiO2)
- **Handle Wafer:** Bulk Silicon (Si)

V. CONCLUSION

To summarize, this report detailed the design, fabrication, and analysis of a high-speed, flexure-based micromirror mechanism optimized for pure translational, or piston, motion. Using the Freedom and Constraint Topology (FACT) method, a symmetric, parallel system of four blade flexures was designed to maximize off-axis stiffness and ensure stable, repeatable motion. This approach addresses key challenges in compliant mechanism design by minimizing parasitic tilt and error attenuation through deliberate overconstraint and axial symmetry. Computational modeling in MATLAB and SolidWorks finite-element analysis confirmed that the system achieves high mechanical resonant frequencies and sufficient modal separation, both essential for precision applications such as LiDAR and adaptive optics. Finally, the outlined surface and bulk micromachining process, integrated with a piezoelectric actuation stack, provides a clear and realistic pathway for fabricating these devices with the necessary microscale precision.

AUTHOR CONTRIBUTIONS

Avi: Topology synthesis, CAD verification, and fabrication process design. **Ryan & Ethan:** MATLAB computational analysis and modeling. **Nathan & Lily:** Synthesis of findings, formatting, and final report writing.

VI. REFERENCES

REFERENCES

[1] Y. Song, R. M. Panas, and J. B. Hopkins, "A review of micromirror arrays," *Precision Engineering*, vol. 51, pp. 729–761, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141635917302210

[2] J. Kim, B.-j. Park, and J. Kim, "Empirical analysis of autonomous vehicle's lidar detection performance degradation for actual road driving in rain and fog," *Sensors*, vol. 23, no. 6, p. 2972, Mar 2023.

[3] T. Sandner and I. Schedwill, "Lamda large aperture mems scanner module for 3d distance measurement," Fraunhofer Institute for Photonic Microsystems (IPMS), Dresden, Germany, Tech. Rep., July 2025, technical Datasheet M3 Lanida EN 2025-07. [Online]. Available: www.ipms.fraunhofer.de

ACKNOWLEDGMENT

Thanks to Professor Hopkins for guidance and instruction throughout the MAE C294A Compliant Mechanisms course.

```
clear all; clc;
format long g;

% global definitions
function [KK]=EulerStiffnessMatrix(Constraint)

format long;

[row column]=size(Constraint);
stages=Constraint(row,1);
wire=Constraint(row,2);
blade=Constraint(row,3);
ConstNum=row-1;
MainMatrix=zeros(stages*6,stages*6);

NumStageConst=zeros(stages,1);
for i=1:ConstNum
    for j=1:stages
        if(Constraint(i,column-1)==j || Constraint(i,column)==j)
            NumStageConst(j,1)=NumStageConst(j,1)+1;
        end
    end
end

for j=1:stages
    Iden=zeros(6,6*NumStageConst(j,1));
    for k=0:NumStageConst(j,1)-1
        Iden(1:6,(k*6)+1:(k*6)+6)=eye(6);
    end
    WrenchMatrix=zeros(6*NumStageConst(j,1),6*stages);
    count=0;
    for i=1:wire
        if(Constraint(i,column-1)==j)
            count=count+1;
            %Construct S for wire flexure i
            ss=zeros(6,6);
            l=Constraint(i,10);
            d=Constraint(i,11);
            E=Constraint(i,13);
            G=Constraint(i,14);
            I=(pi*(d^4))/64;
            J=(pi*(d^4))/32;
            A=(pi*(d^2))/4;
            ss(1,1)=l/(E*I);
            ss(1,5)=-(l^2)/(2*E*I);
            ss(2,2)=l/(E*I);
            ss(2,4)=(l^2)/(2*E*I);
            ss(3,3)=l/(G*J);
            ss(4,2)=(l^2)/(2*E*I);
            ss(4,4)=(l^3)/(3*E*I);
            ss(5,1)=-(l^2)/(2*E*I);
            ss(5,5)=(l^3)/(3*E*I);
            ss(6,6)=l/(E*A);
            S=inv(ss);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            r=Constraint(i,1:3);
            n3=Constraint(i,4:6);
            n3=n3/sqrt(dot(n3,n3));
            orth=null(n3);
            n2=transpose(orth(1:3,1));
            n2=n2/sqrt(dot(n2,n2));
```

```
            n1=cross(n2,n3);
            Na=zeros(6,6);
            Na(1:3,1)=transpose(n1);
            Na(1:3,2)=transpose(n2);
            Na(1:3,3)=transpose(n3);
            Na(4:6,4)=transpose(n1);
            Na(4:6,5)=transpose(n2);
            Na(4:6,6)=transpose(n3);
            Na(4:6,1)=transpose(cross(r,n1));
            Na(4:6,2)=transpose(cross(r,n2));
            Na(4:6,3)=transpose(cross(r,n3));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            NR=zeros(6,6);
            NR(1:6,1:3)=Na(1:6,4:6);
            NR(1:6,4:6)=Na(1:6,1:3);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            Nb=zeros(6,6);
            Nb(1:3,1)=transpose(n1);
            Nb(1:3,2)=transpose(n2);
            Nb(1:3,3)=transpose(n3);
            Nb(4:6,4)=transpose(n1);
            Nb(4:6,5)=transpose(n2);
            Nb(4:6,6)=transpose(n3);
            Nb(4:6,1)=transpose(cross((r-(l*n3)),n1));
            Nb(4:6,2)=transpose(cross((r-(l*n3)),n2));
            Nb(4:6,3)=transpose(cross((r-(l*n3)),n3));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((j-1)*6)+1:((j-1)*6)+6)=NR*S/(Na);
            if(Constraint(i,column)~=0)
                P=zeros(6,6);
                P(4,2)=-l;
                P(5,1)=l;
                WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((Constraint(i,column)-1)*6)+1:
((Constraint(i,column)-1)*6)+6)=NR*S*(P-eye(6))/(Nb);
            end
        elseif(Constraint(i,column)==j)
            count=count+1;
            %Construct S for wire flexure i
            ss=zeros(6,6);
            l=Constraint(i,10);
            d=Constraint(i,11);
            E=Constraint(i,13);
            G=Constraint(i,14);
            I=(pi*(d^4))/64;
            J=(pi*(d^4))/32;
            A=(pi*(d^2))/4;
            ss(1,1)=l/(E*I);
            ss(1,5)=-(l^2)/(2*E*I);
            ss(2,2)=l/(E*I);
            ss(2,4)=(l^2)/(2*E*I);
            ss(3,3)=l/(G*J);
            ss(4,2)=(l^2)/(2*E*I);
            ss(4,4)=(l^3)/(3*E*I);
            ss(5,1)=-(l^2)/(2*E*I);
            ss(5,5)=(l^3)/(3*E*I);
            ss(6,6)=l/(E*A);
            S=inv(ss);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            n3=-Constraint(i,4:6);
            n3=n3/sqrt(dot(n3,n3));
            r=Constraint(i,1:3)+l*n3;
```

```
122             orth=null(n3);
123             n2=transpose(orth(1:3,1));
124             n2=n2/sqrt(dot(n2,n2));
125             n1=cross(n2,n3);
126             Na=zeros(6,6);
127             Na(1:3,1)=transpose(n1);
128             Na(1:3,2)=transpose(n2);
129             Na(1:3,3)=transpose(n3);
130             Na(4:6,4)=transpose(n1);
131             Na(4:6,5)=transpose(n2);
132             Na(4:6,6)=transpose(n3);
133             Na(4:6,1)=transpose(cross(r,n1));
134             Na(4:6,2)=transpose(cross(r,n2));
135             Na(4:6,3)=transpose(cross(r,n3));
136             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
137             NR=zeros(6,6);
138             NR(1:6,1:3)=Na(1:6,4:6);
139             NR(1:6,4:6)=Na(1:6,1:3);
140             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
141             Nb=zeros(6,6);
142             Nb(1:3,1)=transpose(n1);
143             Nb(1:3,2)=transpose(n2);
144             Nb(1:3,3)=transpose(n3);
145             Nb(4:6,4)=transpose(n1);
146             Nb(4:6,5)=transpose(n2);
147             Nb(4:6,6)=transpose(n3);
148             Nb(4:6,1)=transpose(cross((r-(l*n3)),n1));
149             Nb(4:6,2)=transpose(cross((r-(l*n3)),n2));
150             Nb(4:6,3)=transpose(cross((r-(l*n3)),n3));
151             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
152             WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((j-1)*6)+1:((j-1)*6)+6)=NR*S/(Na);
153             if(Constraint(i,column-1)~=0)
154                 P=zeros(6,6);
155                 P(4,2)=-l;
156                 P(5,1)=l;
157                 WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((Constraint(i,column-1)-1)*6)+1:
    ((Constraint(i,column-1)-1)*6)+6)=NR*S*(P-eye(6))/(Nb);
158             end
159         end
160     end
161     for i=1+wire:blade+wire
162         if(Constraint(i,column-1)==j)
163             count=count+1;
164             %Construct S for blade flexure i
165             ss=zeros(6,6);
166             l=Constraint(i,10);
167             w=Constraint(i,11);
168             t=Constraint(i,12);
169             E=Constraint(i,13);
170             G=Constraint(i,14);
171             I1=w*(t^3)/12;
172             I2=t*(w^3)/12;
173             Temp=0;
174             if(w>t)
175                 for n=1:2:7
176                     Temp=Temp+(tanh(n*pi*w/(2*t))/(n^5));
177                 end
178                 J=((t^3)*w/3)*(1-((192*t/((pi^5)*w))*Temp));
179             else
180                 for n=1:2:7
181                     Temp=Temp+(tanh(n*pi*t/(2*w))/(n^5));
```

```
                end
                J=((w^3)*t/3)*(1-((192*w/((pi^5)*t))*Temp));
            end
            A=w*t;
            ss(1,1)=l/(E*I1);
            ss(1,5)=-(l^2)/(2*E*I1);
            ss(2,2)=l/(E*I2);
            ss(2,4)=(l^2)/(2*E*I2);
            ss(3,3)=l/(G*J);
            ss(4,2)=(l^2)/(2*E*I2);
            ss(4,4)=(l^3)/(3*E*I2);
            ss(5,1)=-(l^2)/(2*E*I1);
            ss(5,5)=(l^3)/(3*E*I1);
            ss(6,6)=l/(E*A);
            S=inv(ss);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            r=Constraint(i,1:3);
            n3=Constraint(i,4:6);
            n3=n3/sqrt(dot(n3,n3));
            n2=Constraint(i,7:9);
            n2=n2/sqrt(dot(n2,n2));
            n1=cross(n2,n3);
            Na=zeros(6,6);
            Na(1:3,1)=transpose(n1);
            Na(1:3,2)=transpose(n2);
            Na(1:3,3)=transpose(n3);
            Na(4:6,4)=transpose(n1);
            Na(4:6,5)=transpose(n2);
            Na(4:6,6)=transpose(n3);
            Na(4:6,1)=transpose(cross(r,n1));
            Na(4:6,2)=transpose(cross(r,n2));
            Na(4:6,3)=transpose(cross(r,n3));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            NR=zeros(6,6);
            NR(1:6,1:3)=Na(1:6,4:6);
            NR(1:6,4:6)=Na(1:6,1:3);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            Nb=zeros(6,6);
            Nb(1:3,1)=transpose(n1);
            Nb(1:3,2)=transpose(n2);
            Nb(1:3,3)=transpose(n3);
            Nb(4:6,4)=transpose(n1);
            Nb(4:6,5)=transpose(n2);
            Nb(4:6,6)=transpose(n3);
            Nb(4:6,1)=transpose(cross((r-(l*n3)),n1));
            Nb(4:6,2)=transpose(cross((r-(l*n3)),n2));
            Nb(4:6,3)=transpose(cross((r-(l*n3)),n3));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((j-1)*6)+1:((j-1)*6)+6)=NR*S/(Na);
            if(Constraint(i,column)~=0)
                P=zeros(6,6);
                P(4,2)=-l;
                P(5,1)=l;
                WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((Constraint(i,column)-1)*6)+1:
((Constraint(i,column)-1)*6)+6)=NR*S*(P-eye(6))/(Nb);
            end
        elseif(Constraint(i,column)==j)
            count=count+1;
            %Construct S for blade flexure i
            ss=zeros(6,6);
            l=Constraint(i,10);
```

```
            w=Constraint(i,11);
            t=Constraint(i,12);
            E=Constraint(i,13);
            G=Constraint(i,14);
            I1=w*(t^3)/12;
            I2=t*(w^3)/12;
            Temp=0;
            if(w>t)
                for n=1:2:7
                    Temp=Temp+(tanh(n*pi*w/(2*t))/(n^5));
                end
                J=((t^3)*w/3)*(1-((192*t/((pi^5)*w))*Temp));
            else
                for n=1:2:7
                    Temp=Temp+(tanh(n*pi*t/(2*w))/(n^5));
                end
                J=((w^3)*t/3)*(1-((192*w/((pi^5)*t))*Temp));
            end
            A=w*t;
            ss(1,1)=l/(E*I1);
            ss(1,5)=-(l^2)/(2*E*I1);
            ss(2,2)=l/(E*I2);
            ss(2,4)=(l^2)/(2*E*I2);
            ss(3,3)=l/(G*J);
            ss(4,2)=(l^2)/(2*E*I2);
            ss(4,4)=(l^3)/(3*E*I2);
            ss(5,1)=-(l^2)/(2*E*I1);
            ss(5,5)=(l^3)/(3*E*I1);
            ss(6,6)=l/(E*A);
            S=inv(ss);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            n3=-Constraint(i,4:6);
            n3=n3/sqrt(dot(n3,n3));
            r=Constraint(i,1:3)+l*n3;
            n2=Constraint(i,7:9);
            n2=n2/sqrt(dot(n2,n2));
            n1=cross(n2,n3);
            Na=zeros(6,6);
            Na(1:3,1)=transpose(n1);
            Na(1:3,2)=transpose(n2);
            Na(1:3,3)=transpose(n3);
            Na(4:6,4)=transpose(n1);
            Na(4:6,5)=transpose(n2);
            Na(4:6,6)=transpose(n3);
            Na(4:6,1)=transpose(cross(r,n1));
            Na(4:6,2)=transpose(cross(r,n2));
            Na(4:6,3)=transpose(cross(r,n3));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            NR=zeros(6,6);
            NR(1:6,1:3)=Na(1:6,4:6);
            NR(1:6,4:6)=Na(1:6,1:3);
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            Nb=zeros(6,6);
            Nb(1:3,1)=transpose(n1);
            Nb(1:3,2)=transpose(n2);
            Nb(1:3,3)=transpose(n3);
            Nb(4:6,4)=transpose(n1);
            Nb(4:6,5)=transpose(n2);
            Nb(4:6,6)=transpose(n3);
            Nb(4:6,1)=transpose(cross((r-(l*n3)),n1));
            Nb(4:6,2)=transpose(cross((r-(l*n3)),n2));
```

```
            Nb(4:6,3)=transpose(cross((r-(l*n3)),n3));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((j-1)*6)+1:((j-1)*6)+6)=NR*S/(Na);
            if(Constraint(i,column-1)~=0)
                P=zeros(6,6);
                P(4,2)=-l;
                P(5,1)=l;
                WrenchMatrix(((count-1)*6)+1:((count-1)*6)+6,((Constraint(i,column-1)-1)*6)+1:
((Constraint(i,column-1)-1)*6)+6)=NR*S*(P-eye(6))/(Nb);
            end
        end
    end
    MainMatrix(((j-1)*6)+1:((j-1)*6)+6,1:(stages*6))=Iden*WrenchMatrix;
end

KK=MainMatrix;

end

Delta = [
    0 0 0 1 0 0;
    0 0 0 0 1 0;
    0 0 0 0 0 1;
    1 0 0 0 0 0;
    0 1 0 0 0 0;
    0 0 1 0 0 0; ];

z3 = zeros(3,1);

N = @(n1, n2, n3, L) [n1 n2 n3 z3 z3 z3;
    cross(L, n1) cross(L, n2) cross(L, n3) n1 n2 n3;];
Ninv = @(N) inv(N);


IxS = @(b,h,l,p) (p*b*h*l/12)*(b^2 + h^2); %Ix Stage (Mass momement of Inertia)
IyS = @(b,h,l,p) (p*b*h*l/12)*(l^2 + h^2); %Iy Stage (Mass momement of Inertia)
IzS = @(b,h,l,p) (p*b*h*l/12)*(b^2 + l^2); %Iz Stage (Mass momement of Inertia)


E = 112.4e9; %Pa
G = 49e9; %Pa
p = 2330.0; %kg/m^3

% Definitions (in meters)
%Blade flexure 1
L1 = 0.001* [0, (0.5-0.01), -0.5;]  ;
n11 = [1 0 0]  ;
n12 = [0 1 0]  ;
n13 = [0 0 1]  ;

t1 = 0.02*0.001;
w1 = 0.001;
l1 = 0.005;


%Blade flexure 2
L2 = 0.001* [0, -(0.5-0.01), -0.5;]  ;
n21 = [1 0 0]  ;
n22 = [0 1 0]  ;
n23 = [0 0 1]  ;

```

```
t2 = 0.02*0.001;
w2 = 0.001;
l2 = 0.005;

%Blade flexure 3
L3 = 0.001* [0, (0.5-0.01), 0.5;]  ;
n31 = [-1 0 0]  ;
n32 = [0 1 0]  ;
n33 = [0 0 -1]  ;

t3 = 0.02*0.001;
w3 = 0.001;
l3 = 0.005;

%Blade flexure 4
L4 = 0.001* [0, -(0.5-0.01), 0.5;]  ;
n41 = [-1 0 0]  ;
n42 = [0 1 0]  ;
n43 = [0 0 -1]  ;

t4 = 0.02*0.001;
w4 = 0.001;
l4 = 0.005;

%Stage 1
L5 = [0, 0, 0].' ;
n51 = [1 0 0].' ;
n52 = [0 1 0].' ;
n53 = [0 0 1].' ;

b5 = 0.001;
h5 = 0.001;
l5 = 0.001;

Ix5 = IxS(b5,h5,l5,p);
Iy5 = IyS(b5,h5,l5,p);
Iz5 = IzS(b5,h5,l5,p);

I5 = diag([Ix5, Iy5, Iz5, p*b5*h5*l5, p*b5*h5*l5, p*b5*h5*l5]);

N5 = N(n51, n52, n53, L5);
N5inv = inv(N5);

M5 = N5*Delta*I5*N5inv;
M5inv = inv(M5);

% Mode and Frequency Calculation
Constraint = [
    L1 n13 n12 l1 w1 t1 E G 1 0;
    L2 n23 n22 l2 w2 t2 E G 1 0;
    L3 n33 n32 l3 w3 t3 E G 1 0;
    L4 n43 n42 l4 w4 t4 E G 1 0;
    1 0 4 0 0 0 0 0 0 0 0 0 0 0 0];

[K1] = EulerStiffnessMatrix(Constraint);
Minv = M5inv;

[x,lambda] = eig(Minv*K1);

omega = lambda.^(1/2);

```

```
x
omega
K1
M5

om2 = omega/(2*pi)
```