Soft Robotics: Robotic Fish

by

Jonathan Kelvin, Kai Gruener, Stephen Otto, David Gonzalez

California State Polytechnic University, Pomona

Mechanical Engineering Department

Fall 2020 - Spring 2021

Faculty Advisor: Prof. Carlos Castro

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

The objective of the project was to design and build a soft robotic fish capable of swimming in shallow water, either autonomously or by remote control. The tail was actuated by alternating pressure chambers pushing each other cyclically. This design was chosen to best mimic the tail motion of a Tuna fish. SolidWorks simulations were utilized to find the magnitude of deflection for three different tail configurations. Once the deflection magnitudes were established, control volume analysis was conducted to calculate the relationship between the deflection, actuation frequency, and thrust. Due to limitations with the closure of campus, a final design was not able to be assembled. Simulations were heavily relied upon to determine the drag forces and maximum velocity of the fish utilizing the pneumatic actuation tail design. SolidWorks was also utilized to perform CFD Analysis to find the drag forces acting on the robot, which were used to find the maximum velocity of the robot. Controls were also designed in C# to control the depth of the robot and to control the forward motion of the fish. These controls are to interface on an arduino with all control hardware in a final build.

Acknowledgement

We would like to give special thanks to our advisor Dr. Carlos Castro for his continuous support and feedback throughout the duration of this project.

1. Objective

The objective of this project is to recreate a fish’s movement utilizing soft robotic design. The general shape and motion of the fish should be loosely based on the motion of a living fish. The purpose of employing soft robotics is to prevent damage or injury to anything that interacts with the robot. In the case of an underwater application, this is important to prevent damage to coral or other fragile wildlife. The primary usage of the robot is to either work in tandem with a diver or to work independently for shallow water reconnaissance.

2. Literature Review

Thunniform swimming is defined as a swimming motion that uses only the rear half of the body to propel itself through the water [1]. This style of swimming is commonly utilized by tuna fish. Once the style of propulsion was established the problem of actuation still lay ahead. In a Tuna Fish the muscles contract with electrical signals coming from the nervous system of the fish. There are methods of actuation with similar physical function, where an electrical current can be pushed through a material and results in the material contracting. However, at the time that method of creating actuation was not phisable. The group decided to employ a simpler pneumatic system for actuation. The pneumatic system consists of small air sacs that run along the tail that are pressurized with air. The sacks then push against the one adjacent to it and thus creates a displacement to either side. Researchers have found success in designing soft robot fish that utilize pneumatic chambers with an on board fluid supply [2]. The group decided on designing the robot based on a Tuna Fish which exhibits thunniform swimming (figure 1).

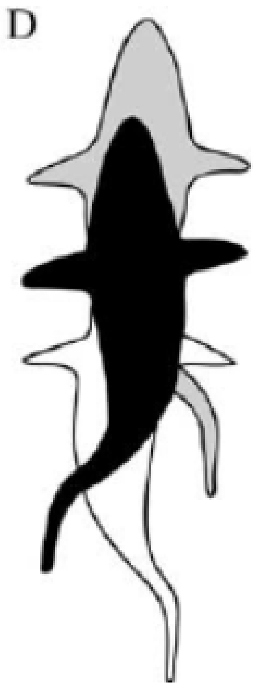


Figure 1: Thunniform Swimming [3]

3. Design

The design of the fish follows principles of biomimicry and soft robotics, however the robot is not a true soft robot. Simple soft robots are composed of entirely soft parts, that is, material that will deform more than the objects that the robot may come in contact with [4]. Our design utilizes principles of a hybrid soft robotic design, in which hard components work together with small components throughout the total system. All components of the robot that are externally contactable are soft, with the exception of the face plate, which is hard in order to protect the components inside.

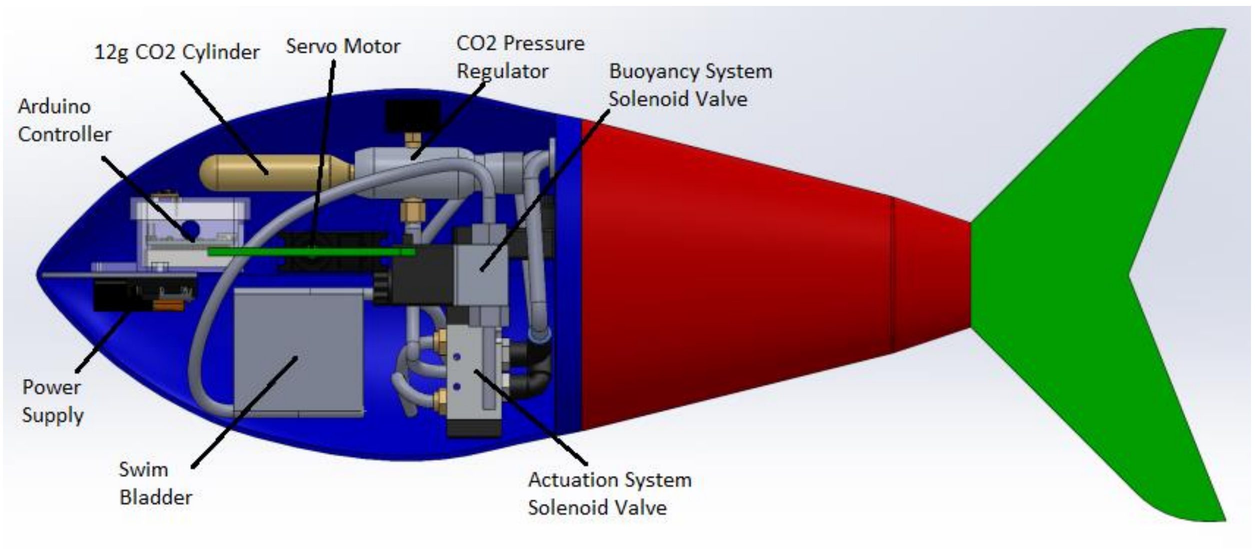


Figure 2: Fish internals 3D models

The robot design is separated into 3 primary systems; the locomotion, buoyancy, and arduino control systems (figure 2). Both the buoyancy and locomotion systems are supplied with carbon dioxide at 60 psi from the fluid supply. The fluid supply consisted of a 12 gram compressed CO2 canister. The canister was attached to a small pressure regulator normally used for carbonating beverages. After exiting the regulator, the pressurized CO2 is distributed to both the locomotion and buoyancy systems by a T-adapter, followed by solenoid valves at both systems. The solenoid valves are controlled by the arduino system located at the head of the fish.

The robot is designed with a rigid head and flexible tail. The head houses the electronics used to control the fish. The chassis is to be made out of silicon to maintain a soft body. For the tail, three different systems of actuation were simulated in SolidWorks to determine the most optimal design. The difference between the three is how the shape of the actuator deforms to cause deflection of the tail fin to produce thrust.

3a. Design 1

Design 1 consists of a single air chamber on one side of the tail (figure 3). The tail is to be then wrapped in cording to prevent radial deformation of the tail. This is done to create bending deformation when pressurized CO2 enters one side of the tail. See figure 7 for model deflection results.

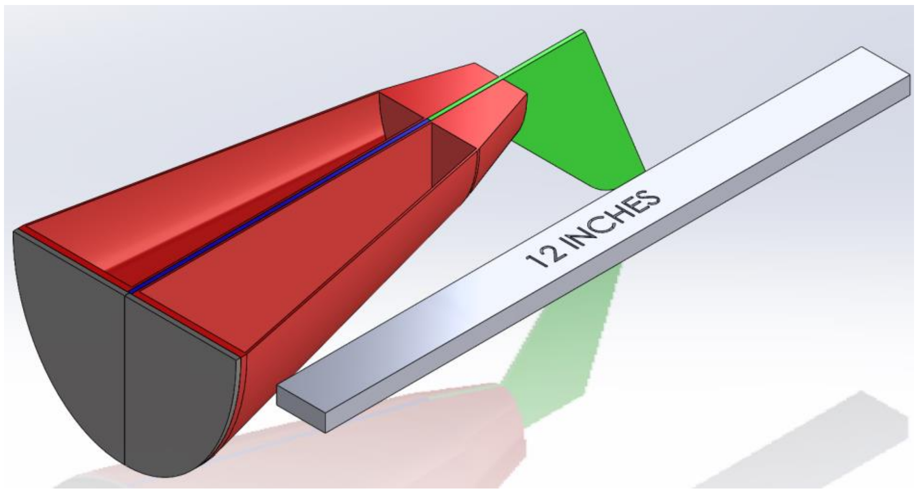


Figure 3: Tail Design 1

3b. Design 2

Design 2 consists of multiple chambers along each side of the tail (figure 4). When one side is filled with CO2 gas the, the chambers inflate and and push up against the adjacent chamber. This produces a bending deformation. This effect can be observed in figure 8 in the Actuation Simulation section.

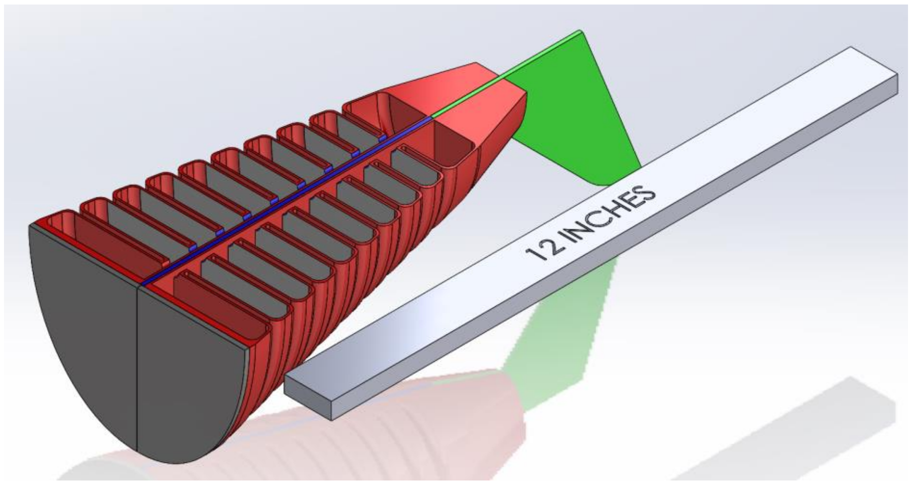


Figure 4: Tail Design 2

3c. Design 3

Design 3 is similar to a combination of Design 1 and 2. Each section consists of a partitioned cavity that the CO2 can occupy (figure 5). Once the cavities are filled the partitioned volumes expand to create the bending deflection. See figure 9 for model deflection results.

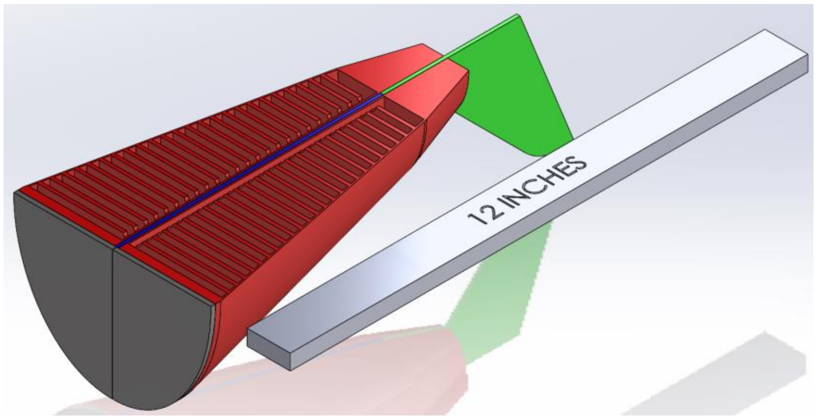


Figure 5: Tail Design 3

4. Simulation Procedure and Results

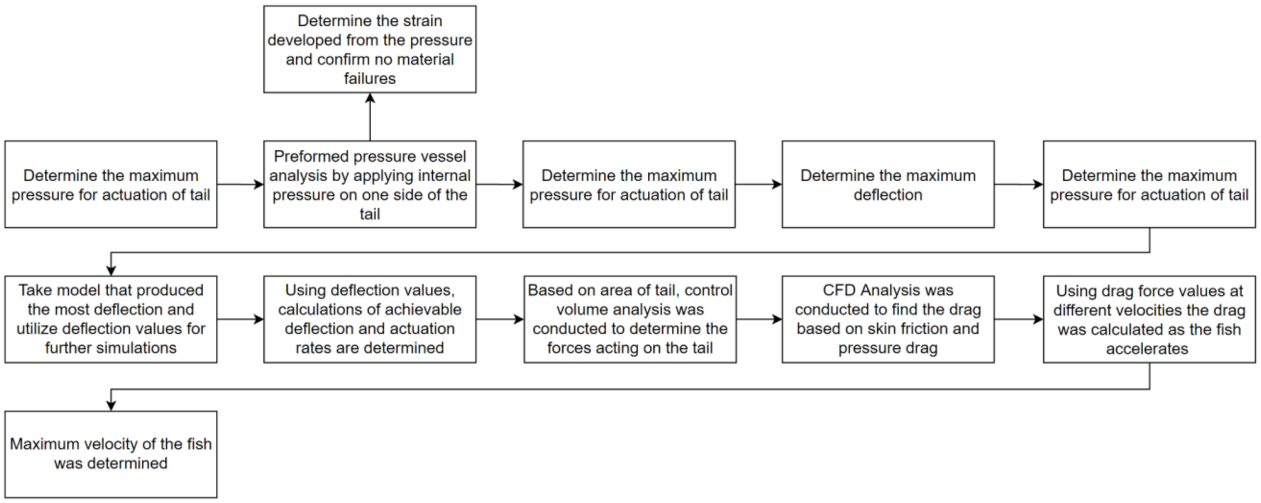

Determine the strain developed from the pressure and confirm no material failures
Determine the maximum pressure for actuation of tail
Preformed pressure vessel analysis by applying internal pressure on one side of the tail
Determine the maximum pressure for actuation of tail
Determine the maximum deflection
Determine the maximum pressure for actuation of tail
Take model that produced the most deflection and utilize deflection values for further simulations
Using deflection values, calculations of achievable deflection and actuation rates are determined
Based on area of tail, control volume analysis was conducted to determine the forces acting on the tail
CFD Analysis was conducted to find the drag based on skin friction and pressure drag
Using drag force values at different velocities the drag was calculated as the fish accelerates
Maximum velocity of the fish was determined

Figure 6: Simulation Procedure Flow Chart

Figure 6 demonstrates the simulation procedures for determination of the maximum velocity of the fish.

4a. Actuation Simulation

Once the working pressure of 60 psi for the pneumatic actuation was determined, pressure vessel analysis was performed on the actuators through SolidWorks. The mechanical properties of TPU used for the simulation are as follows in Appendix 9a [5]. However, TPU exhibits properties similar to rubber. Thus, the material has nonlinear stress-strain characteristics and stiffness may change due to factors such as temperature, strain rate, and manufacturing methods. Therefore some sort of linearity with the mechanical properties of the material was required to be assumed.

Each tail was simulated as a pressure vessel, with 60 psi applied to the internal air sacks on one side and 15.6 psi to the outside (to simulate atmospheric pressure and up to 3 feet underwater). Inflating the air sacks on one side causes the tail to deflect towards the deflated side. Below are figures 7, 8, and 9 showing deflection results of the simulations using the different designs of tail actuators.

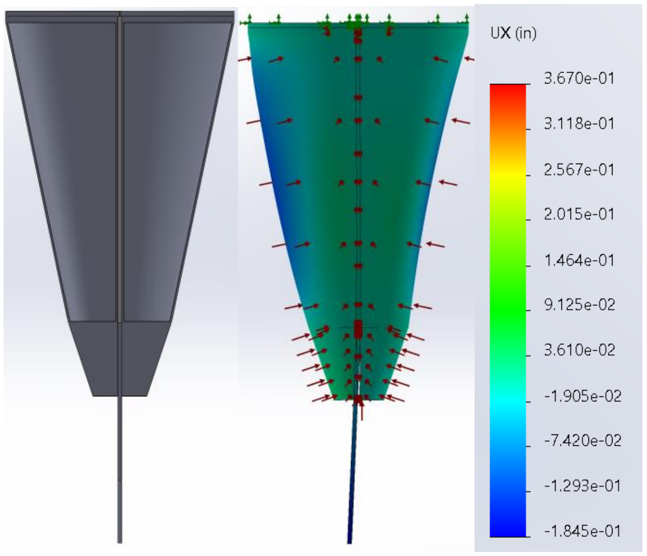


Figure 7: Design 1 with Cross Section (Left), Deflection Model (Center), Scale (Right)

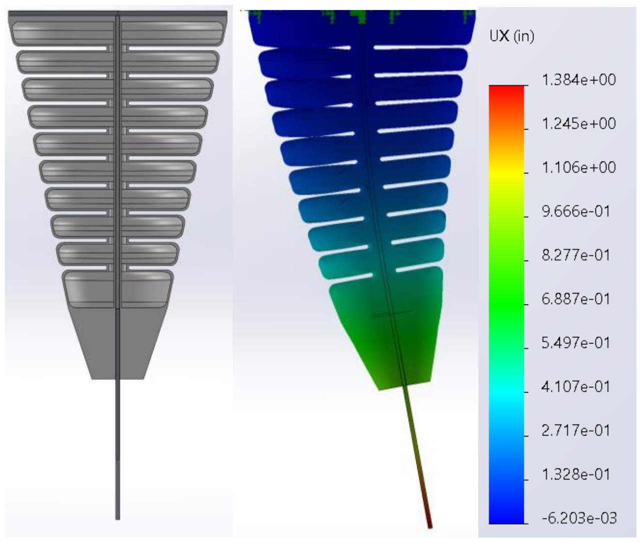


Figure 8: Design 2 with Cross Section (Left), Deflection Model (Center), Scale (Right)

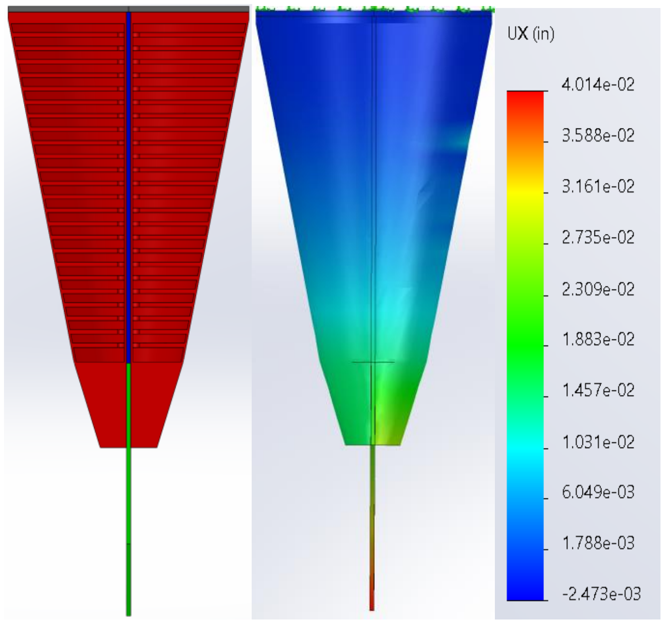


Figure 9: Design 3 with Cross Section (Left), Deflection Model (Center), Scale (Right)

Each design did produce some sort of actuation. However, each tail design varied on the magnitude of the actuation the tail was able to produce with the given conditions. Table 1 lists the magnitude of actuation (measured from the tip of the tail to the neutral axis of the tail).

Table 1: Results of Pressure Vessel Actuation Test

Design Number	Description	Tail displacement (in)
1	Tail with one chamber each side with cable wrap.	0.1
2	Tail with 10 internal chambers with 9 outer channels.	1.4
3	Tail with 26 internal chambers.	0.04

4b. Calculations of Forces Produced by Tail

The robot is designed to exhibit characteristics similar to a Tuna fish. This being the case the style of swimming is to mimic Thunniform swimming [3]. Since the tail is the only portion that actuates, the assumption that the kinematics of the tail has one degree of freedom was made in the calculations. This also means that the tail is a rigid body along its length as shown in figure 10. The parameter that was chosen to be the one degree of freedom is the angle the tail makes with the undeflected neutral axis of the tail.

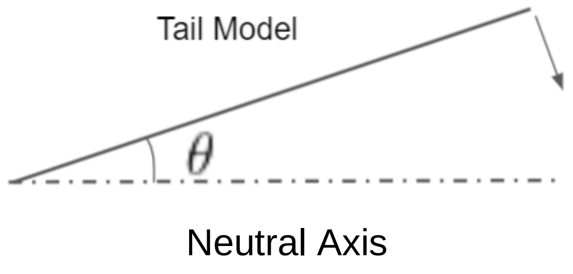


Figure 10: Basic Model of Actuation

While using the maximum deflection found in the SolidWorks simulation, equation 1 is the result.

$$U_{\mathrm{p}} = U_{\mathrm{a}} \cdot f \cdot sin(f \cdot t) \tag{1}$$

Where U_p is the deflection position of the end of the tail, U_a is the maximum deflection from the simulations (Table 2), f is the frequency of actuation, and t is the time variable. In these calculations the frequency was set by the group. Using simple trigonometry, the angle that the tail creates relative to the neutral axis is shown in equation 2.

$$\theta_\mathrm{a} = arcsin(U_\mathrm{a}/L_\mathrm{tail}) \quad (2)$$

Where$U_\mathrm{a}$is the Deflection at the end of the tail, L_tail is the length of the tail, andθ_a is the desired amplitude of the angle. Applying harmonics then yields equation 3.

$$\theta_\mathrm{p} = \theta_\mathrm{a} \cdot f \cdot sin(f \cdot t) \quad (3)$$

Where θ_p is the angular position at a given time, f is the frequency of a cycle of actuation, and t is the time variant. The frequency in this case is the same frequency used to derive the actuation magnitude as shown in equation 4.

$$V_\mathrm{tail} = U_\mathrm{a} \cdot f \cdot cos(f \cdot t) \quad (4)$$

The resulting velocity, angular position, and angular velocity are shown in figure 11 and figure 12. The maximum angular deflection was 7.96 degrees, the maximum angular velocity was 12.5 deg/s, and the maximum velocity of the tail was 2.16 in/s.

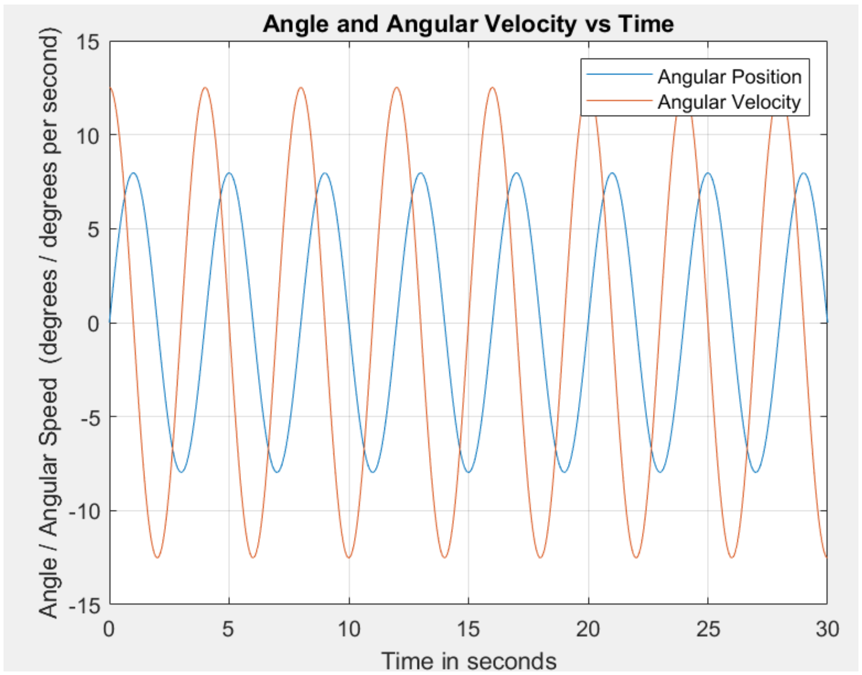


Figure 11: Angle and Angular Velocity vs Time

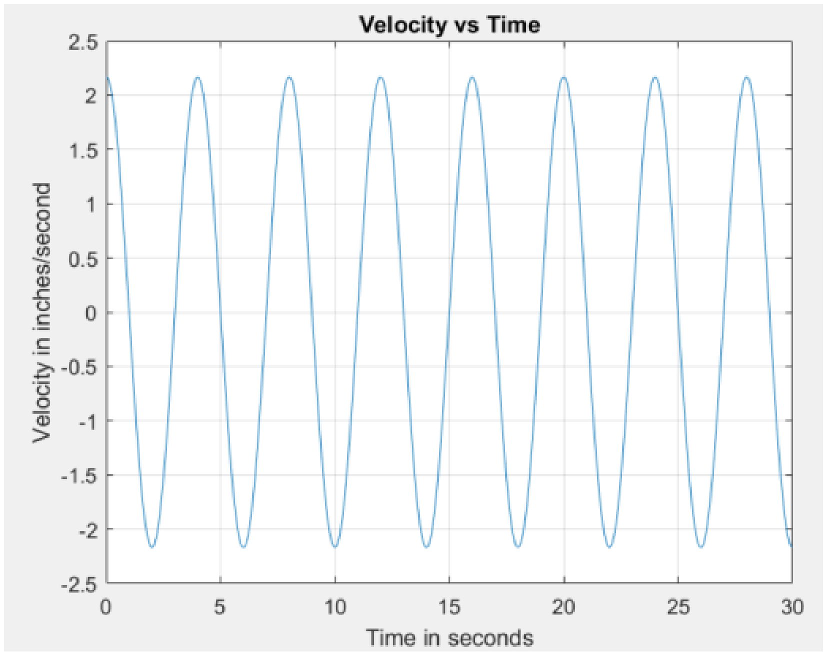


Figure 12: Velocity vs Time

4c. Calculation of Forces Produced by Actuation

In order to perform the Control Volume Analysis, an area was to be created that the fin would occupy when actuated back and forth. In this case the fin resembled a fin of an actual fish and thus in order to perform the calculations a simplified rectangular envelope was utilized as seen in figure 13.

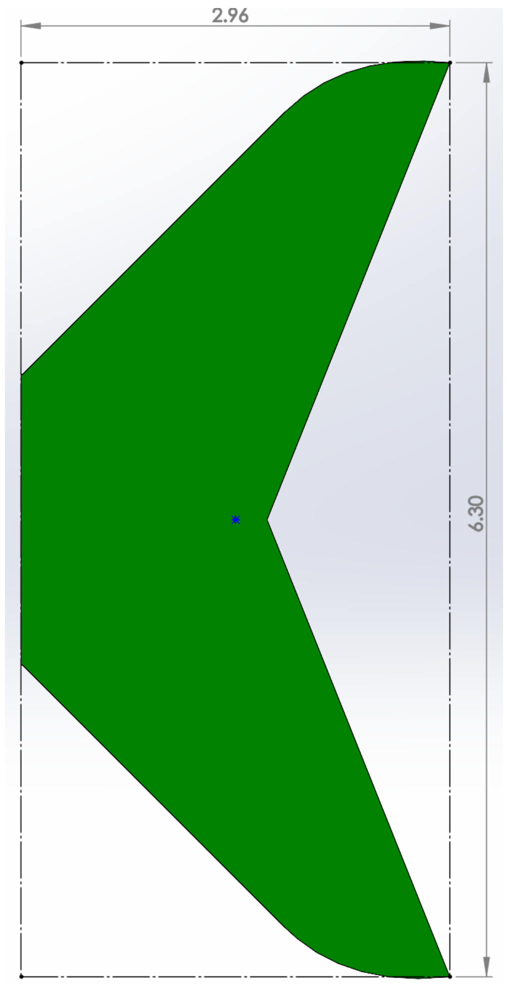


Figure 13: Fin Envelope (Side View) Dimensions in Inches

The envelope that the fin occupies will be the area utilized to calculate the projected areas that make up the control volume as is in figure 14.

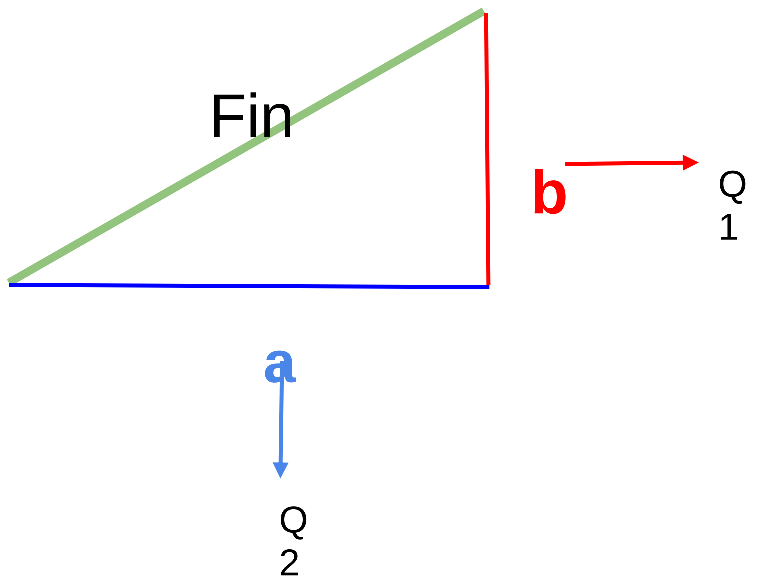


Figure 14: Top View of Projected Areas

Where:

$$a = Length_{Fin} \cdot cos(\theta_{Fin}) \quad (5)$$

and

$$b = Length_{Fin} \cdot sin(\theta_{Fin}) \quad (6)$$

The area used to calculate the control volumes does not represent the projected area of the actual tail. Thus, a correction factor needed to be created for further calculations as shown in equation 7.

$$CF = \frac{Area_{actual}}{Area_{simplified}} \quad (7)$$

For the thrust produced by the actuation a Control Volume Analysis was performed. The tail is able to push water away from the fish in both directions along the neutral axis and perpendicular to it. The Volume also changes as the angle changes with actuation. The change in flow rates axially and perpendicular to the neutral axis are equivalent to the change of the main control volume. This yields the following relationships.

$$Q = Q_1 + Q_2 \tag{8}$$

Where

$$Q = \frac{1}{2} \cdot c^2 \cdot h \cdot cos(2 \cdot \theta_{Tail}) \cdot \omega^{Tail} \cdot \frac{\pi}{180} \tag{9}$$

And

$$Q_2 = Area_{simplified} \cdot V_{tail} \cdot cos(\theta_{Tail}) \tag{10}$$

Which yields

$$Q_1 = Q - Q_2 \tag{11}$$

Once the control volumes are calculated at a given time the following relationship describes the thrust that the tail produces.

$$Thrust = \frac{CF \cdot \rho_{water} \cdot Q_1 \cdot (V_{Tail} \cdot cos(\theta_{Fin}))^2}{V_{Tail}} \tag{12}$$

The thrust value at a given time is as shown in figure 15.

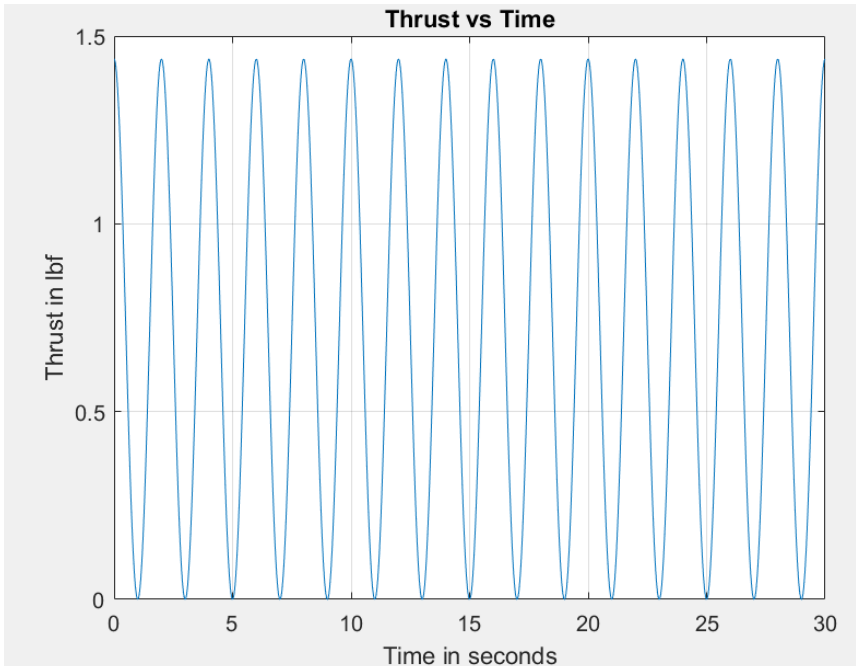


Figure 15 : Thrust vs Time

Under ideal conditions the average thrust produced by the actuation of the tail is 0.7 lbf.

4d. Velocity Calculations

Submerged objects moving through a fluid experience friction between the fluid and the surface of the object (skin friction) and an increase in pressure due to the front facing cross sectional area. These two forces together yield the total drag experienced by the submerged object. Due to the complexity of the fish with varying tail deflections SolidWorks Flow Simulations were utilized to determine the drag that the fish experiences. Images of the SolidWorks models are as shown in figure 16.

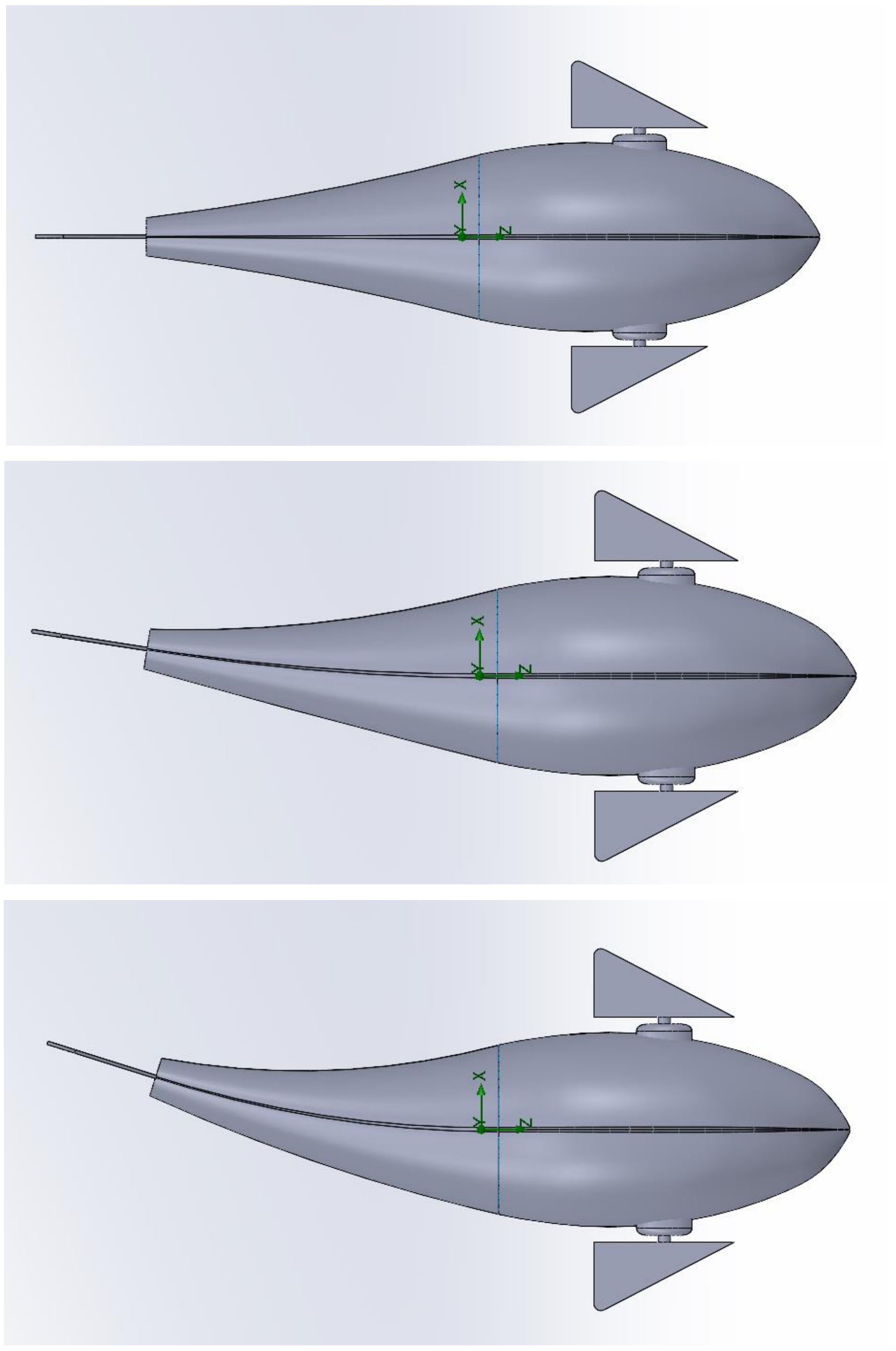


Figure 16: Model with No Deflection (Top), Half Deflection (Middle), and Max Deflection (Bottom)

To perform the SolidWorks Simulations, the initial conditions that were imputed to the system are as shown in Appendix 9b. All drag values are listed in Appendix 9d. The results are shown in figure 17 and 18 below.

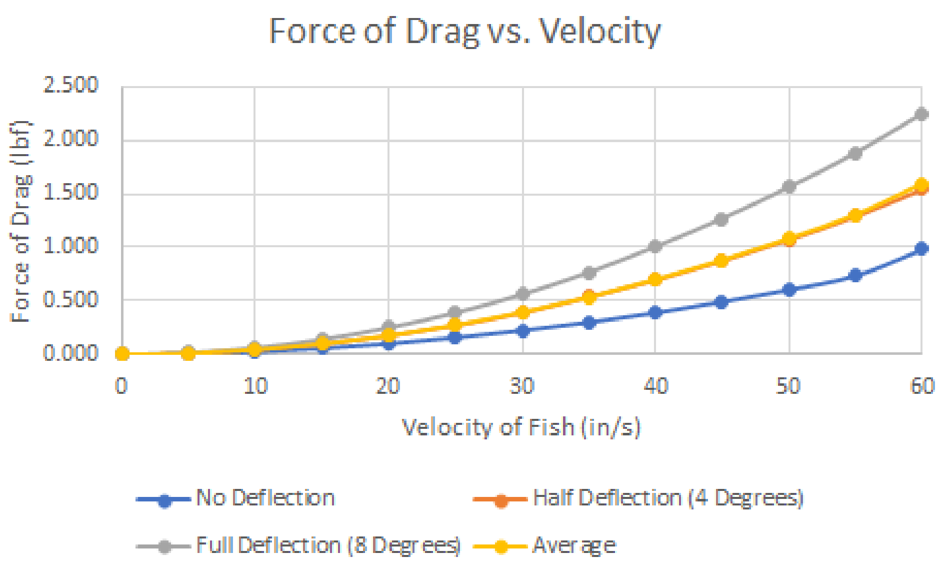


Figure 17: Force of Drag vs. Velocity

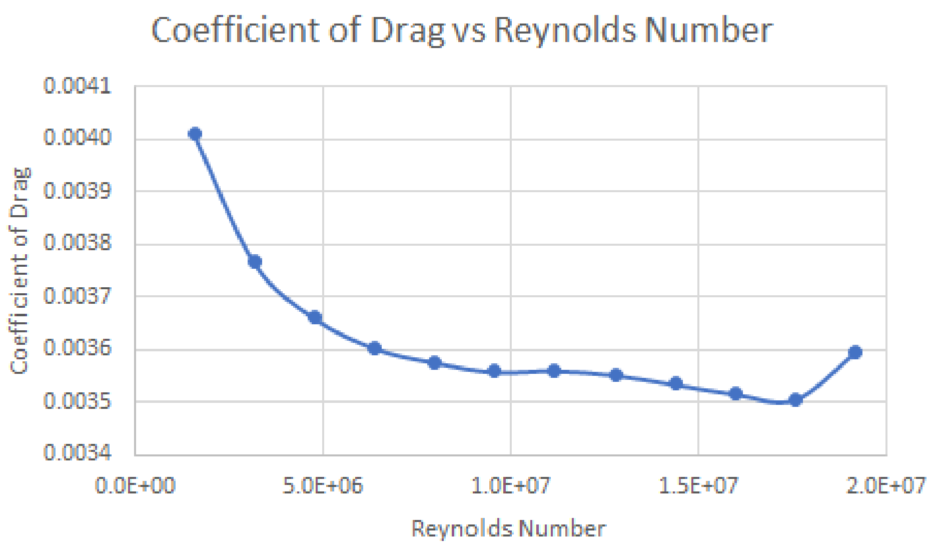


Figure 18: Coefficient of Drag vs. Reynolds Number

In order to find the maximum velocity of the robot, iterative calculations were programmed into MATLAB. The total force that the fish experiences is a relationship between the Average Thrust and the Drag Force.

$$Force = Average\,Thrust - Drag \tag{13}$$

Using simple kinematics, the acceleration that the fish experiences is as shown in the following.

$$a = Force/Mass \tag{14}$$

Where a is the acceleration and mass was designated to be 4 lbm. With this the following expression represents the velocity for each iteration.

$$V_i = V_{i-1} + a \cdot t \tag{15}$$

Where V_i is the velocity of the current iteration, V_{i-1} is the velocity of the previous iteration, and $t = 0.5$ seconds is the time interval that the fish is able to accelerate before the start of a new iteration.

After all the iterations the final velocity is 40.8 inches per second or 2.3 MPH. At this velocity, the average drag force acting on the fish was 0.7 lbf.

5. Hardware

The hardware was chosen in order to have the smallest and cheapest materials in order to allow the fish to operate properly.

The main design of the system requires one 5-way 12 volt solenoid valve, and two 2-way 12 volt solenoid valves. The 5 way solenoid valve has tubing that attaches to both sides of the actuation fin. When the valve is off the fin is inflated on the left side and deflated on the right. When it turns on it will deflate the left side and inflate the right side. A CO2 canister will be connected to the valve supplying the air. This CO2 canister will be connected to a pressure regulator which keeps the air at 60 psi. Using Arduino controllers, the valve can be turned off and on.

The buoyancy system requires two valves to be connected to a bag with the same CO2 canister connected to the opening. The bag will act as a ballast tank and fill with water. One valve will open and close letting CO2 in to allow the bag to push out the water. The other valve will allow water to enter and exit.

The Arduino Nano was chosen for compactness and to give the ability to control the other hardware. Everything can be assembled on a 400 pin breadboard. An accelerometer was chosen to measure angles, servos with the PWN (pulse width modulation, a typical way to control servos in RC vehicles) as the angle output were chosen to control the pectoral fins. The solenoid valves were chosen to control the buoyancy system and actuation system. Everything will be powered by a 12V source which will be 12V alkaline batteries. The Arduino, the servos, and the accelerometer can be powered by the 12V source. This can be done by using a voltage regulator allowing it to be 5V. The Solenoid Valves need a 12V source, and in order to control it with the 5V Arduino, transistors will be used which allow control of higher voltage systems. The solenoid valves, when turning off and on, create a spark because of the fast change in current which is not good for the system. In order to combat this snubber, diodes are used to allow current to go one way and no back emf will cause disruption.

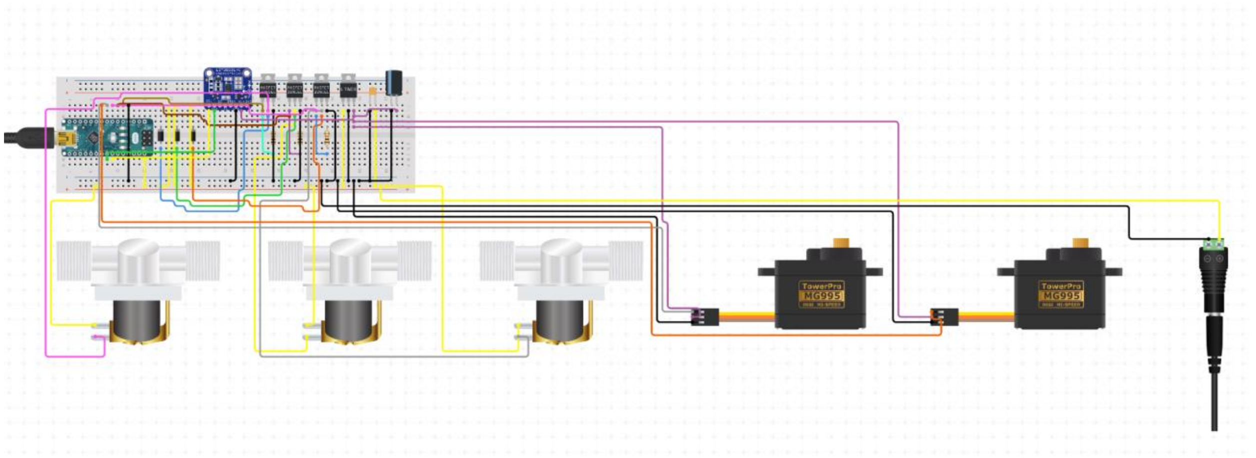


Figure 19: Final Schematic of the Wiring

Figure 19 above shows the final wiring of the Hardware of the system. The valves pictured look different but the wiring is exactly the same.

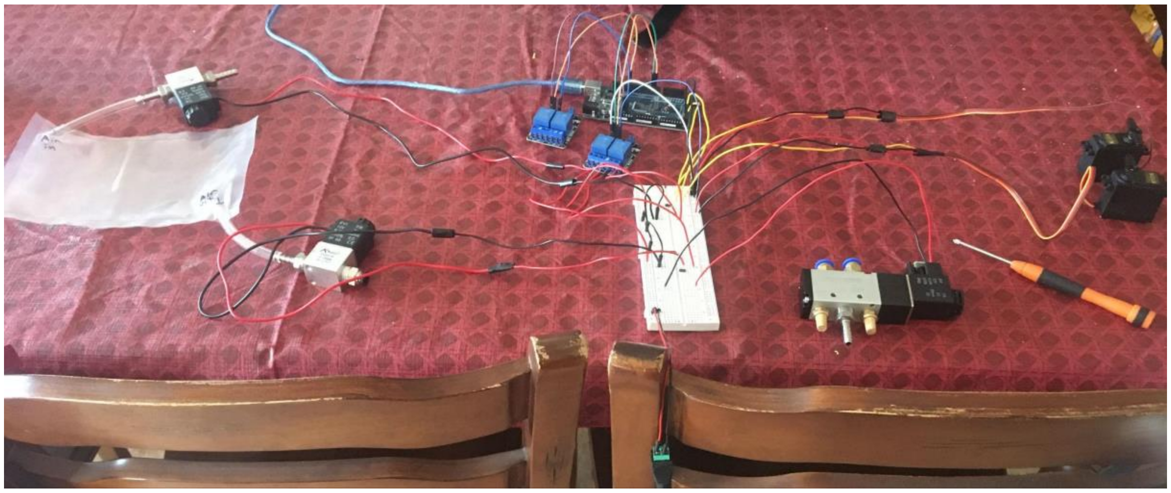


Figure 20: Assembled Buoyancy, Controls, and Tail Actuation subsystems

Due to time constraints, a full chassis could not be assembled. The internal subsystems of the fish were assembled to the point of being ready for installation into a chassis (figure 20).

It was expected that when 3D printing parts there would be some imperfections between individual printed layers. This was the case with the tail actuators that were printed (figure 21). The air sacs were leaking air in between the individual layers. This was considered a failed test and further printing of tail actuators were abandoned.

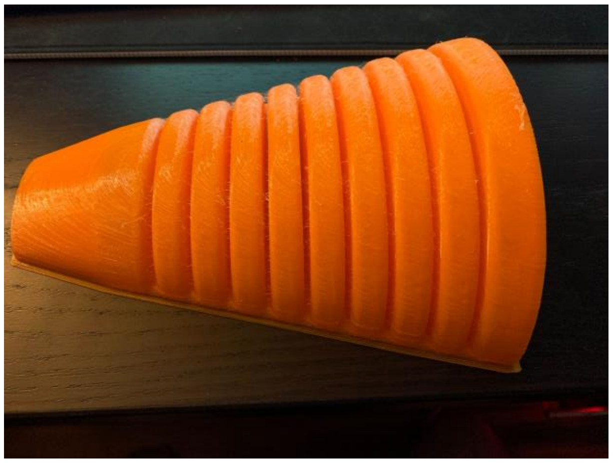


Figure 21: Half of the 3D Printed Tail

6. Software

The project was programmed using Visual Studio Windows Form Application with C# and Arduino IDE. Visual Studio was used in order to have real-time control of the fish. Serial communication between Arduino and Visual Studio allows the two to communicate to each other.

Figure 22 shows the graphical user interface made for the soft robotic fish. It will allow the user to control the buoyancy, actuation, and pectoral fins. It will also relay the angles of the accelerometer in order to see how the fish is oriented.

The code for the Visual Studio uses button click in order to create an action. The text box at the bottom will display the angles spaced apart in order. The serial communication will send a string to the Arduino IDE. It is a four letter word and the first character controls the pectoral fins. The second character controls the buoyancy system. The third and fourth character control the actuation. The third character will tell the actuation to turn on or off and the fourth tells the Arduino whether the system is inflated on the left or right.

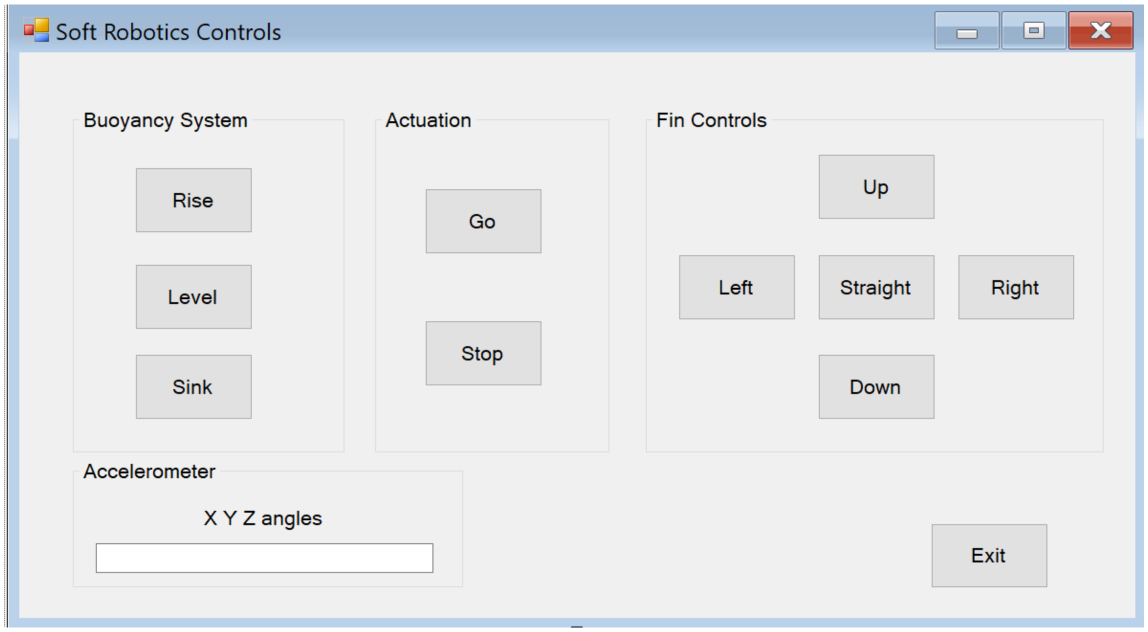


Figure 22: Visual Studio GUI

Arduino IDE was used to communicate to the hardware. Using Arduino the valves can open and close, the servo motors can be controlled, and the accelerometer can be run.

The Arduino IDE receives the communication in a loop from Visual Studio for a quick response on any actions. The loop will have 4 functions. If the string has not changed the Arduino will continue as is. The first function the accelerometer will receive the timer variable and for every one second it will print back the X, Y, and Z angles. The second function is the buoyancy system and when the sink is pressed the second valve will be opened. When rise is pressed both valves will be opened. When level is pressed the valves will be closed. The third function is the Servo system and when one of the buttons is pressed it will relay the servos to spin to a certain angle.

The last function is the Actuation which when go is pressed will tell the valve to turn on and off so it will inflate and deflate the actuation fin. When stop is pressed it will keep one side of the actuation fin inflated. This can be all viewed in Appendix 9f.

7. Conclusion

A soft robotic fish was designed and fluid simulation was performed to determine the effectiveness of soft robotic actuated movement. The robot utilizes pressurized CO2 to pressurize alternating chambers in the TPU tail, which produces a thunniform swimming motion. The same fluid supply is also capable of pressurizing the fish's swim bladder, allowing for increasing buoyancy for retrieval. The fish's motion was simulated using SolidWorks and the drag forces and thrusting forces were determined. An arduino nano was selected to minimize the size of the control system. The control system was utilized to control the buoyancy and tail actuation systems through solenoid valves. The actuator material selected for the actuator simulations was thermoplastic polyurethane or TPU (see Appendix 9a for properties used). When 3D printed, reoccurring imperfections in the trial actuators made them porous and gave way for air leaks rendering the physical model unusable.

8. Future Work

Due to the inability to meet in person, one of the project limitations was completing the construction of the prototype. Different team members were able to develop and test the different subsystems individually. A theoretical model was produced considering the components separately. Performing tests on a physical prototype to allow for considerations for design changes is a very important aspect of the engineering process but it was simply not possible to do comfortably in the midst of the pandemic circumstances. For this reason it is expected that extensive field testing of a complete prototype would be part of the future tasks to expand on this project.

The performance of the tail soft actuators can benefit greatly from a change in material and production process. Since the actuator material was TPU, a 3D printer is able to produce prints of flexible TPU using the proper settings, a convenient option when the access to lab facilities is limited given the current circumstances worldwide. Unfortunately, the air leaks in the actuators made the physical model unusable. Having access to a common lab facility will allow for the production of an actuator prototype using silicone, a more flexible material. Using a more flexible material for the flat plate separating the two tail actuators, combined with the silicone tail, will result in larger tail fin angular deflection and more realistic motion.

In nature, the shape of fish scales that produce the skin textures have different hydrodynamic effects. When considering the hydrodynamics analyses of this theoretical prototype, there is more room for the exploration of the effects the variations of surface texture has on drag. To a higher extent, the control capabilities during navigation can be improved testing different shapes and sizes of tail and pectoral fins.

Future work also includes improving on the currently present subsystems. For the buoyancy system, it has been considered incorporating an additional 2-way solenoid valve to control the air exhaust. It seems possible that one more valve can be included in the present envelope of the front section of the robot if the existing components are meticulously rearranged, but considering additional modifications might force a change in dimensions of the fish envelope.

Also, it has to be noted that a relocation of the air exhaust out of the fish would be beneficial. The buoyancy sack would need an exhaust port at the top of the bag for release of CO2 when negative buoyancy is desirable. In the current design iteration, the fish would only be capable of filling with water when the bag is unpressurized, effectively resulting in a one time use retrieval system rather than an adjustable buoyancy system. Fine tuned buoyancy control is limited to the servo motor controlled pectoral fins, however with modification fine tune buoyancy could be extended to the robot's swim bladder as well.

One way to take this project to a higher level would be making the robotic fish autonomous. In order to accomplish this, several sensors would have to be added. Pressure sensors can generate inputs for the buoyancy system controls as the robot's depth in the water varies. Proximity and positioning sensors would help avoid obstacles and adjust to changing conditions during navigation by controlling parameters like duration of actuation period of the actuation speed if there is a way to vary the pressure in the system. Also, considering the small size of a CO2 canister and the fact that gas is continuously exiting through the exhaust, it is expected that the pressure source will be depleted fast. An alternative design can incorporate a closed hydraulic system for actuation instead of a pneumatic one. A small hydraulic pump can be responsible for circulating the hydraulic fluid and controlled valves would direct and alternate the pressure to the actuator as needed.

9. Appendix

9a. Material Properties of TPU

The following table lists values used in the SolidWorks simulation.

Table 2: Material Properties of TPU [5]

Modulus of Elasticity	2410 N/mm2
Poisson’s ratio	0.39

9b. MATLAB Code for Tail Velocity and Control Volume Calculations

Table 3: CFD Input Parameters

Parameter	Value
Fluid	Water
Surface Roughness	15 microinches
Flow Condition	Turbulent
Flow Velocity	Values as seen in Appendix C

9c. MATLAB Code for Tail Velocity and Control Volume Calculations

```
%Calculating for Angle
y_max = 1.38; %inches
y_min = -y_max;

dy = 0.01;
y = y_min : dy : y_max;

L = 9.96;   %including center of area for fin
```

```
theta_tail = asind(y/L);

A = max(theta_tail);

figure
title('Angle vs Actuation Distance')
ylabel('Angle in Degrees')
xlabel('Actuation Position in inches')
plot(y,theta_tail)
grid on

%Calculating Position of Tail vs Time
y_max = L * sind(A);

t_max = 30;
t_min = 0;

dt = 0.01;
t = t_min : dt : t_max;

p = 4; %seconds

f = (2 * pi) / p;   %in radians/sec

y_tail = y_max * sin ( f * t );

figure
plot(t,y_tail)
title('Actuation Position vs Time')
ylabel('Actuation Position in inches')
xlabel('Time in seconds')
grid on

%Angle Relative to Time
angle = A * sin ( f * t );  %in degrees
```

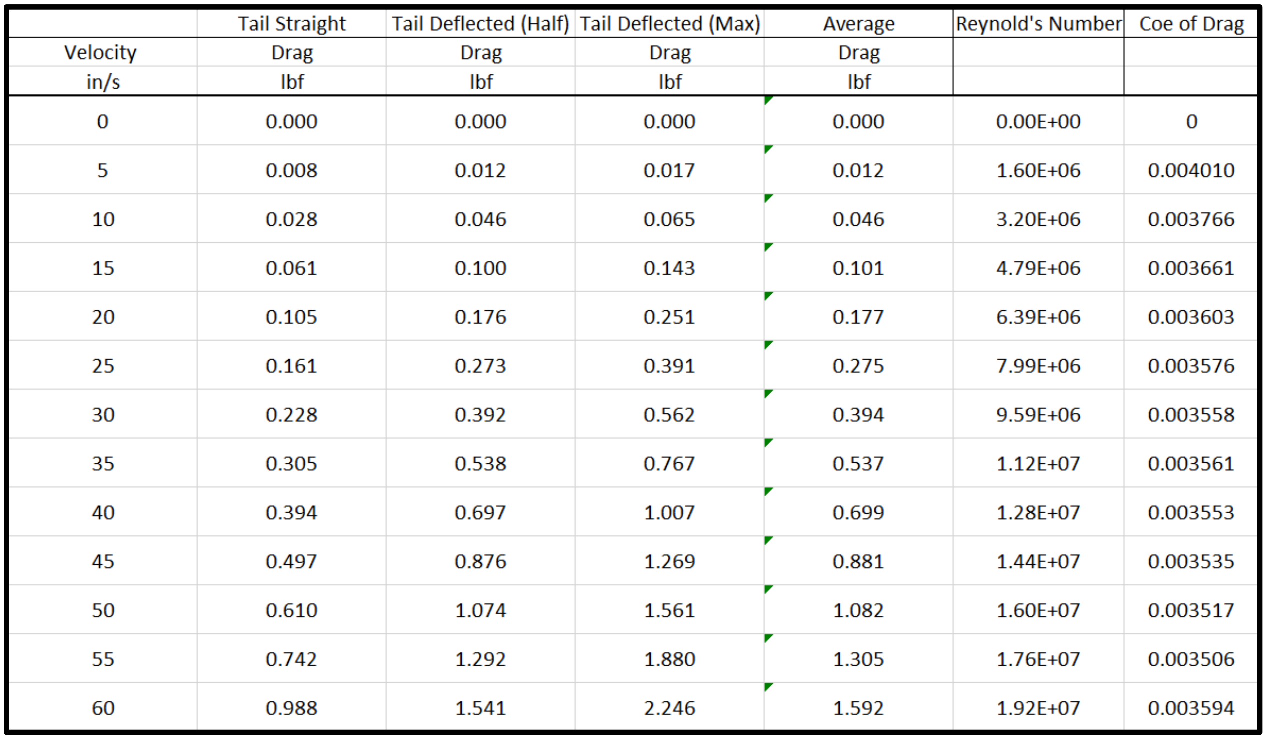
```
anglep = A * f * cos( f * t );  %degrees/s

figure
plot(t,angle)
hold on
plot(t,anglep)
legend('Angular Position','Angular Velocity')
title('Angle vs Time')
ylabel('Angle / Angular Speed  (degrees / degrees per second)')
xlabel('Time in seconds')
grid on
hold off

%Calculating Velocity of Tail
theta_fin = theta_tail; %degrees

v_tail = y_max * f * cos ( f * t );

figure
plot(t,v_tail)
title('Velocity vs Time')
ylabel('Velocity in inches/second')
xlabel('Time in seconds')
grid on

area = 2.95 * 6.30;     %inch^2

area_a = 9.92;    %inch^2 of actual surface of fin

CF = area_a / area ;    %correction factor that accounts for not square tail

c = 2.95; %length of fin

a = c * cosd(theta_fin);

b = c * sind(theta_fin);
```

```
h = 6.30;

rho = 0.03625;   %lbm/in^3

%Q Volumetric Flow Rate

Q = (1/2) * c^2 * h * cosd( 2 * angle ) .* anglep * pi / 180;

Q2 = area * v_tail .* cosd(angle);

Q1 = Q - Q2;

%use area is cross section of fluid leaving
%Calculating Thrust
%Force of Water Jet F = padV^2

Thrust = abs(CF* rho * Q1 .* ( v_tail .* cosd(angle) ).^2 ./ v_tail);     %lbf

AvThr = mean(Thrust);

F_req = CF * rho * Q2 .* ( v_tail - v_tail .* sind( angle ) ).^2 ./ v_tail;    %lbf

AvF_req = mean(F_req);

T_req = L/12 * F_req;   %foot-lbf

AvT_req = mean(T_req);

figure
plot(t,Thrust)
title('Thrust vs Time')
ylabel('Thrust in lbf')
xlabel('Time in seconds')
grid on
```

```
figure
plot(t,F_req)
title('Resistance Force vs Time')
ylabel('Resistance Force in lbf')
xlabel('Time in seconds')
grid on

figure
plot(t,T_req)
title('Resistance Torque (Torque Required) vs Time')
ylabel('Resistance Torque in foot-lbf')
xlabel('Time in seconds')
grid on
```

9d. Drag Force vs Swimming Velocity Simulation Results (CFD)

Table 4: CFD Results



	Tail Straight	Tail Deflected (Half)	Tail Deflected (Max)	Average	Reynold's Number	Coe of Drag
Velocity	Drag	Drag	Drag	Drag		
in/s	lbf	lbf	lbf	lbf		
0	0.000	0.000	0.000	0.000	0.00E+00	0
5	0.008	0.012	0.017	0.012	1.60E+06	0.004010
10	0.028	0.046	0.065	0.046	3.20E+06	0.003766
15	0.061	0.100	0.143	0.101	4.79E+06	0.003661
20	0.105	0.176	0.251	0.177	6.39E+06	0.003603
25	0.161	0.273	0.391	0.275	7.99E+06	0.003576
30	0.228	0.392	0.562	0.394	9.59E+06	0.003558
35	0.305	0.538	0.767	0.537	1.12E+07	0.003561
40	0.394	0.697	1.007	0.699	1.28E+07	0.003553
45	0.497	0.876	1.269	0.881	1.44E+07	0.003535
50	0.610	1.074	1.561	1.082	1.60E+07	0.003517
55	0.742	1.292	1.880	1.305	1.76E+07	0.003506
60	0.988	1.541	2.246	1.592	1.92E+07	0.003594

9e. Maximum Velocity Calculations

```
iterations = 23000;

V = ones(1,iterations);

V(1,1) = 0; %initial speed

for j = 2:iterations

for i = 2:j+1
   %k and l define rows within a matrix
   if V(1,i-1) >= 0 && V(1,i-1) <= 5
          k = 1;
          l = 2;
   elseif V(1,i-1) > 5 && V(1,i-1) <= 10
          k = 2;
          l = 3;
   elseif V(1,i-1) > 10 && V(1,i-1) <= 15
          k = 4;
          l = 5;
   elseif V(1,i-1) > 15 && V(1,i-1) <= 20
          k = 5;
          l = 6;
   elseif V(1,i-1) > 20 && V(1,i-1) <= 25
          k = 6;
          l = 7;
   elseif V(1,i-1) > 25 && V(1,i-1) <= 30
          k = 7;
          l = 8;
   elseif V(1,i-1) > 30 && V(1,i-1) <= 35
          k = 8;
          l = 9;
   elseif V(1,i-1) > 35 && V(1,i-1) <= 40
```

```
        k = 9;
        l = 10;
elseif V(1,i-1) > 40 && V(1,i-1) <= 45
        k = 10;
        l = 11;
elseif V(1,i-1) > 45 && V(1,i-1) <= 50
        k = 11;
        l = 12;
elseif V(1,i-1) > 50 && V(1,i-1) <= 55
        k = 12;
        l = 13;
else
        k = 13;
        l = 14;
end
%Interpolator
    XA = V(1,i-1);   %Desired Point

    X1 = drag(k,1); %Point Lower Limit

    Y1 =  drag(k,5); %Value Lower Limit

    X2 = drag(l,1); %Point Upper Limit

    Y2 = drag(l,5); %Value Upper Limit

    YA = Y1 + (Y2 - Y1)/(X2 - X1)  * (XA - X1);    %Desired Value

    %use projected are for the Cd

    A = 322.87; %surface area


    %Cd = 2 * YA / (rho * A * V(1,j-1)^2);

    Drag = YA;
```

```
        F_t = AvThr - Drag; %Total Forward Force

        mass =   4;   %in lbm

        Accel = (F_t / mass) / 12; %in/sec^2

        time = 0.5; %time resolution to accelerate

        V(1,i) = V(1,i-1) + Accel * time;

        if F_t < 0.001
            break
        end

 end

end

Velocity_Max = max(V);

disp(Velocity_Max)
```

9f. Visual Studio C# Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
```

```
namespace WindowsFormsApp1
{

        public partial class Form1 : Form
        {

        public static String movement = "slsl";
        String serialDataIn;
        public Form1()
        {
        InitializeComponent();

        }

        private void Form1_Load(object sender, EventArgs e)
        {
        SerialPort.Open();

        }

        private void button1_Click(object sender, EventArgs e)
        {
        movement.ToCharArray()[0] = 'l';
        SerialPort.Write(movement);
        }

        private void button_up_Click(object sender, EventArgs e)
        {
        movement.ToCharArray()[0] = 'u';
        SerialPort.Write(movement);
        }

        private void button_down_Click(object sender, EventArgs e)
        {
        movement.ToCharArray()[0] = 'd';
        SerialPort.Write(movement);
```

```
}

private void button_straight_Click(object sender, EventArgs e)
{
movement.ToCharArray()[0] = 's';
SerialPort.Write(movement);
}

private void button_right_Click(object sender, EventArgs e)
{
movement.ToCharArray()[0] = 'r';
SerialPort.Write(movement);
}

private void button_rise_Click(object sender, EventArgs e)
{
movement.ToCharArray()[1] = 'r';
SerialPort.Write(movement);
}

private void button_level_Click(object sender, EventArgs e)
{
movement.ToCharArray()[1] = 'l';
SerialPort.Write(movement);
}

private void button_sink_Click(object sender, EventArgs e)
{
movement.ToCharArray()[1] = 's';
SerialPort.Write(movement);
}

private void button_go_Click(object sender, EventArgs e)
{
movement.ToCharArray()[2] = 'g';
SerialPort.Write(movement);
```

```
        }

        private void button_stop_Click(object sender, EventArgs e)
        {
        movement.ToCharArray()[2] = 's';
        SerialPort.Write(movement);
        }

        private void button_exit_Click(object sender, EventArgs e)
        {
        SerialPort.Close();
        this.Close();
        }

        private void SerialPort_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
        serialDataIn = SerialPort.ReadExisting();
        this.Invoke(new EventHandler(ShowData));
        }

        private void ShowData(object sender, EventArgs e)
        {
        textBox_angle.Text = serialDataIn;
        }
        }
}
```

9g. Arduino IDE Code

```
#include <Servo.h>
#include "Wire.h"
#include <MPU6050_light.h>


Servo myservo;
Servo myservo2;
```

```
int angle;
int angle2;

int X;
int Y;
int Z;

String xstring;
String ystring;
String zstring;

//global variables
const int Relay_Enable = 2;
const int Relay2_Enable = 3;
const int RelayActuation_Enable = 6;

String receiveval;


//accelerometer
MPU6050 mpu(Wire);
unsigned long timer = 0;

void setup() {
  // put your setup code here, to run once:
  myservo.attach(9);
  myservo2.attach(10);
  pinMode(Relay_Enable,OUTPUT);
  pinMode(Relay2_Enable,OUTPUT);
  pinMode(RelayActuation_Enable,OUTPUT);

  Wire.begin();
  byte status = mpu.begin();
  while(status!=0){ } // stop everything if could not connect to MPU6050
  delay(1000);
  // mpu.upsideDownMounting = true; // uncomment this line if the MPU6050 is mounted upside-
down
  mpu.calcOffsets(); // gyro and accelero

  Serial.begin(9600);
}
```

```
void loop() {

  //read communication
if (Serial.available() > 0)
  {
   receiveval=Serial.readString();
            }
            else
            {}

            //get angles from accelerometer

  timer = millis();

//accelerometer
Accelerometer(timer);
            //servo system
ServoSystem(receiveval);

  //buoyancy system
Buoyancy(receiveval);

  //actuation system
 Actuation(receiveval);
}


int Accelerometer(unsigned long timer)
{
  mpu.update();

  if((millis()-timer)>1000){ // print data every 1s
  X = mpu.getAngleX();
  Y = mpu.getAngleY();
  Z = mpu.getAngleZ();
  xstring = String(X);
  ystring = String(Y);
  zstring = String(Z);
  Serial.print(xstring+' '+ystring+' '+zstring);
  }
```

```
}

int Buoyancy(String receiveval){
  if(receiveval[1]=='l')
  {
          digitalWrite(Relay_Enable,LOW);
          digitalWrite(Relay2_Enable,LOW);

  }
  else if(receiveval[1]=='r')
  {
          digitalWrite(Relay_Enable,LOW);
          digitalWrite(Relay2_Enable,HIGH);
  }
  else if(receiveval[1]=='s')
  {
          digitalWrite(Relay_Enable,HIGH);
          digitalWrite(Relay2_Enable,LOW);
  }
}



int Actuation(String receiveval){
  if(receiveval[2]=='s')
  {
          digitalWrite(RelayActuation_Enable,LOW);
  }
  else if(receiveval[2]=='g')
  {
          if(receiveval[3]=='l')
          {
          digitalWrite(RelayActuation_Enable,LOW);
          receiveval.remove(3);
          receiveval.concat('r') ;
          }
          else if(receiveval[3]=='r')
          {
          digitalWrite(RelayActuation_Enable,HIGH);
          receiveval.remove(3);
          receiveval.concat('l') ;
          }
  }
  delay(500);
return(receiveval);
}



int ServoSystem(String receiveval){
if(receiveval[0]=='r')
  {

            angle=45;
            angle2=135;

  }
  else if(receiveval[0]=='l')
  {
            angle=135;
            angle2=45;
  }
  else if(receiveval[0]=='s')
  {

            angle=90;
            angle2=90;
  }
  else if(receiveval[0]=='u')
  {

            angle=45;
            angle2=45;
  }
  else if(receiveval[0]=='d')
  {

            angle=135;
            angle2=135;
  }
  myservo.write(angle);
  myservo2.write(angle2);
}
```

References

[1]Dan Xia, Weishan Chen, Junkao Liu, Zhijun Wu, Yuhua Cao, The three-dimensional hydrodynamics of thunniform swimming under self-propulsion, Ocean Engineering, Volume 110, Part A, 2015, Pages 1-14, ISSN 0029-8018, https://doi.org/10.1016/j.oceaneng.2015.10.008. (https://www.sciencedirect.com/science/article/pii/S0029801815005454)

[2] Polygerinos, P., Correll, N., Morin, S. A., Mosadegh, B., Onal, C. D., Petersen, K., Cianchetti, M., Tolley, M. T., & Shepherd, R. F. (2017). Soft Robotics: Review of Fluid-Driven Intrinsically Soft Devices; Manufacturing, Sensing, Control, and Applications in Human-Robot Interaction. *Advanced Engineering Materials*, *19*(12), 1700016. https://doi.org/10.1002/adem.201700016

[3] Davis, Matthew & Chakrabarty, Prosanta. (2011). Tripodfih (Aulopiformes: Bathypterois) locomotion and landing behaviour from video observation at bathypelagic depths in the Campos Basin of Brazil. Marine Biology Research. 7. 297-. 10.1080/17451000.2010.515231.

[4] Marchese, A. D., Onal, C. D., & Rus, D. (2014). Autonomous Soft Robotic Fish Capable of Escape Maneuvers Using Fluidic Elastomer Actuators. Soft Robotics, 1(1), 75–87. https://doi.org/10.1089/soro.2013.0009

[5] Hyojeong Lee, Ran-i Eom, Yejin Lee, "Evaluation of the Mechanical Properties of Porous Thermoplastic Polyurethane Obtained by 3D Printing for Protective Gear", *Advances in Materials Science and Engineering*, vol. 2019, Article ID 5838361, 10 pages, 2019. https://doi.org/10.1155/2019/5838361

[6] “Fiber-Reinforced Actuators.” *Soft Robotics Toolkit*, softroboticstoolkit.com/book/fiber-reinforced-bending-actuators.

[7]Anthoine, Jerome & Olivari, Domenico & Portugaels, D.. (2009). Wind-tunnel blockage effect on drag coefficient of circular cylinders. Wind and Structures. 12. 10.12989/was.2009.12.6.541.

[8]Muhammad Rusydi Muhammad Razif,Ili Najaa Aimi Mohd Nordin,Mahrokh Bavandi,”Two chambers soft actuator realizing robotic gymnotiform swimmers fin”, Article, 7 pages, 2014

[9]Mosadegh, Bobak, Panagiotis Polygerinos, Christoph Keplinger,Sophia Wennstedt, Robert F. Shepherd, Unmukt Gupta, JongminShim, Katia Bertoldi, Conor J. Walsh, and George M. Whitesides.2014. “Pneumatic Networks for Soft Robotics That Actuate Rapidly.”Advanced Functional Materials 24 (15) (January 10): 2163–2170.doi:10.1002/adfm.201303288.

[10]Godfrey, Juleon Taylor, “Soft Robotic Actuators”, Article, 2017