# Hydraulics for Micro-Robotic Actuation

Chiraag Nataraj
Howell N. Tyson, Sr., SURF Fellow
Mentors: Dr. Jaret Riddick & Mr. Harris Edge
Associate Mentor: Dr. Joel Burdick
US Army Research Lab
Aberdeen Proving Grounds
09/27/13

# Contents

# List of Figures



# List of Tables



# List of Programs

**Abstract**

The motive of the proposed student research is to investigate hydraulic actuation for small and micro robotic systems, and develop control algorithms for their autonomous operation. The objective is to implement this idea in a test system to prove the viability of the concept. In order to ensure that the pump is as efficient as possible, equations and parameters are used in order to optimize the efficiency and force/pressure output of the pump. Furthermore, parametrized CAD drawings of the different parts of the pump are utilized in a programmatic manner to facilitate changing parameters. Once the pump has been optimized and fully developed, it will be used to power an actuator which can trigger an action such as e.g. an arm bringing down a hammer. For the case of an arm and hammer, pistons can be placed both above and below the joint to allow the arm to be raised and lowered.

# 1 Acknowledgments

I would like to thank the following people and organizations.

- **Dr. Jaret Riddick & Mr. Harris Edge** for being excellent mentors
- **Dr. Joel Burdick** for being an excellent associate mentor
- **US Army Research Lab** for this wonderful opportunity
- **Mr. and Mrs. Thomas Tyson** for generously donating funds for my SURF
- **Caltech and ASEE** for financial support

# 2 Introduction

Rapid prototyping (i.e., 3D printing) has enabled significant recent advances in structural design to support robotics to meet ambitious demands in manipulation and mobility. Researchers at Oak Ridge National Laboratory (ORNL) have fabricated a robotic arm to assist underwater repair of off-shore oil rigs. The robotic arm is fabricated using 3D printing where the plumbing for hydraulics is embedded in the structure, thus enabling hydraulic actuation of the robotic arm.

The proposed student research is a collaboration between US Army Research Lab, Vehicle Technology Directorate Mechanics Division (MD) and Autonomous Systems Division (ASD) to investigate hydraulic actuation for small and micro robotic systems, and develop control algorithms for their autonomous operation. The MD mentor will support the design and fabrication (by 3D printing) of a matrix of prototype embedded hydraulically actuated structures. The ASD mentor will support the development of control algorithms to facilitate autonomous function of the hydraulic actuators for characterization of performance, (power consumed, power output, response, fidelity, etc.), for the purposes of stabilization, manipulation, ambulation, tunable compliance, jumping, etc.

# 3 Background

## 3.1 Pump Types

Before this concept can be implemented, it must be discussed what type of pump will be used and what its parameters are. The following sections list the different types and their characteristics.

#### 3.1.1 Lobe Pump

- Handles solids without damaging them
- Handles low viscosity liquids poorly compared to other PD pumps
- Suction ability is low, and loading characteristics are not as good as other designs
- High viscosity liquids require reduced speeds to get acceptable performance

#### 3.1.2 Progressive Cavity Pump

- Fixed flow rate pump
- Hydrodynamic lubrication required (should not be a problem since we're using water)
- More torque required for starting, can deteriorate if "run dry" (again, should not be a problem)
- Good with transporting thick/lumpy fluids
- Abrasive fluids will shorten life of stator

#### 3.1.3 Gear Pump

- Generally used in:
  - Petrochemicals
  - Chemicals
  - Paint/ink
  - Resins/adhesives
  - Pulp/paper
  - Food
- As one can tell, usually used with slurries
- Good for high viscosity fluids
- Tight clearances ($\approx 10\,\mu m$)

#### 3.1.4 Axial Piston Pump

- Need an almost perfect seal between the rotary assembly and the pump body with rotary part turning at $\approx 3000$ rpm
- Pistons less than half an inch in diameter
- Very small clearances involved
- Usually a supporting vane pump

#### 3.1.5 Radial Piston Pump

- High efficiency
- High pressure
- Low flow and pressure ripple
- Low noise
- Very high load at lowest speed
- No axial internal forces
- High reliability
- Bigger radial dimensions compared to Axial Piston Pump
- Main uses:
  - Machine tools
  - High pressure units
  - Test rigs
  - Automatic transmission
  - Hydraulic suspension control
  - Plastic and powder injection moulding
  - Wind energy
- Main fluids:
  - mineral oil
  - biodegradable oil
  - oil in water
  - water-glycol
  - synthetic ester
  - cutting emulsion

#### 3.1.6 Diaphragm Pump

- Good suction lift characteristics
- Can handle sludges and slurries with a lot of grit
- Discharge pressure up to 1200 bar
- Good dry running characteristics
- Up to 97% efficient
- Good self-priming capabilities
- Can handle highly viscous fluids
- May cause water hammer (use pulsation damper to minimize)

#### 3.1.7 Screw Pump

- Suited for applications such as fuel-injection, oil burners, boosting, hydraulics, fuel, lubrication, and circulating

#### 3.1.8 Rotary Vane Pump

- Common uses:
  - High pressure hydraulic pumps
  - Supercharging
  - Power steering
  - Automatic transmission pumps
  - Carbonators
  - Vacuum pumps
  - Driving gyroscopic flight instruments, attitude indicator, and heading indicator on light aircraft
  - Secondary air injection
- Can be used in low-pressure gas environments

#### 3.1.9 Peristaltic Pump

- Low maintenance
- Able to handle slurries, viscous, shear-sensitive, and aggressive fluids
- Prevents back-flow and siphoning without valves
- Flexible tubing will tend to degrade with time
- Flow is pulsed (especially at low rotation speeds)

#### 3.1.10 Rope Pump

- Completely useless for our aims

#### 3.1.11 Flexible Impeller Pump

- Particularly suitable for transfer of viscous, delicate, and slurry fluids
- Used widely in oenological field, food processing, chemical industry, cosmetic, and marine fields

#### 3.1.12 Hydraulic Ram Pump

- Uses water hammer to operate valves and move fluid
- Not very relevant to our aim - requires incline in delivery pipe

#### 3.1.13 Pulser Pump

- Uses gravity to pump water to a higher elevation

#### 3.1.14 Airlift Pump

- Very reliable
- Liquid not in contact with any mechanical elements
- Acts as a water ærator
- Flow rate very limited
- Suction very limited
- Suitable only if head is very low
- Quantity of air to compress is very high compared to liquid flow required
- Geyser pump improves on this
  - Pumps with greater suction and less air
  - Permits proportional control

#### 3.1.15 Radial-Flow Pump

- Fluid exits radially
- Operates at higher pressures and lower flow rates than axial and mixed-flow pumps

#### 3.1.16 Axial-Flow Pump

- Relatively high discharge at relatively low head
- Smallest of the dimensions among many of the conventional pumps

#### 3.1.17 Mixed-Flow Pump

- Compromise between Axial-Flow and Radial-Flow
- Operates at higher pressures than axial-flow pumps and delivers higher discharges than radial-flow pumps

#### 3.1.18 Eductor-Jet Pump

- A jet creates a low-pressure area which sucks in fluid
- After passing through “throat”, velocity reduces

## 3.2 Pump Selection

After considering the many different types of pumps, the radial-flow pump was selected, as what was chiefly needed was a large velocity to generate enough force to drive the actuator. However, the next task is to optimize the parameters of the pump so as to provide maximum force at the exit nozzle. This will then set the operating range of the actuator.

## 3.3 Pump Optimization

### 3.3.1 Equations

The variables taken into account during the optimization are listed in Table 1, along with their units and meanings.

The equations used are listed in Equations (1) to (6).

$$v_1\left(\dot{m},\rho,A\right)=\frac{\dot{m}}{\rho A} \tag{1}$$

$$v_2\left(\dot{m},\rho,r_2,b,n,\omega\right)=\sqrt{\left(\frac{\dot{m}n}{2\pi\rho r_2 b}\right)^2+\left(\omega r_2\right)^2} \tag{2}$$

$$h\left(\dot{m},\rho,N_s,\omega\right)=\left(\frac{\omega\sqrt{\dot{m}}}{N_s\sqrt{\rho}}\right)^{\frac{4}{3}} \tag{3}$$

$$P_{\text{out}}\left(\dot{m},\rho,A,r_2,b,n,N_s,\omega\right)=\dot{m}\left[h\left(\dot{m},\rho,N_s,\omega\right)+\frac{1}{2}\left(\left\{v_2\left(\dot{m},\rho,r_2,b,n,\omega\right)\right\}^2-\left\{v_1\left(\dot{m},\rho,A\right)\right\}^2\right)\right] \tag{4}$$

$$\eta\left(\dot{m},\rho,A,r_2,b,n,N_s,\omega\right)=\frac{P_{\text{out}}\left(\dot{m},\rho,A,r_2,b,n,N_s,\omega\right)}{P_{\text{in}}} \tag{5}$$

$$F\left(\dot{m},\rho,A_i,A_f,r_2,b,n,\omega\right)=\dot{m}\left(\frac{A_i}{A_f}\right)v_2\left(\dot{m},\rho,r_2,b,n,\omega\right) \tag{6}$$

Examining these equations, many conclusions were drawn.

1. To maximize Equation (5), Equation (4) needs to be maximized
2. To maximize Equation (4), Equation (3) and Equation (2) need to be maximized, while Equation (1) needs to be minimized
3. $\dot{m}$ and $\rho$ are properties of the fluid and so are not free parameters
4. Therefore, the only free parameter in Equation (1) is $A$, so $A$ needs to be as large as possible
5. To maximize Equation (3), $N_s$ needs to be as small as possible and $\omega$ needs to be as large as possible
6. To maximize Equation (2), $b$ needs to be as small as possible, $n$ needs to be as large as possible, and $\omega$ needs to be as large as possible ($r_2$ will be ignored for now, as its relationship is complicated)
7. To maximize Equation (6), $A_i$ needs to be as large as possible, $A_f$ needs to be as small as possible, and Equation (2) is already being maximized

One interesting note is that whenever multiple equations are all dependent on one variable — such as both Equations (2) and (3) on $\omega$ — they are consistent; in this case, they both require that $\omega$ be as large as possible.

| Variable | Meaning | Units |
|---|---|---|
| $v_1$ | Velocity of the fluid as it enters the inlet nozzle | $^{cm}/_{s}$ |
| $v_2$ | Velocity of the fluid as it exits the impeller and enters the outlet nozzle | $^{cm}/_{s}$ |
| $h$ | Head of the pump | cm |
| $P_{\text{out}}$ | Power generated by the impeller | $^{kg \cdot cm^2}/_{s^3}$ |
| $\eta$ | Efficiency | Unit-less |
| $F$ | Force generated by fluid flow | $^{kg \cdot cm}/_{s^2}$ |
| $\dot{m}$ | Mass flow rate | $^{kg}/_{s}$ |
| $\rho$ | Density | $^{kg}/_{cm^3}$ |
| $A$ | Cross-sectional area of inlet nozzle | $cm^2$ |
| $r_2$ | Outer radius of impeller | cm |
| $b$ | Thickness of blades of the impeller | cm |
| $n$ | Number of blades on the impeller | Unit-less |
| $\omega$ | Rotational speed of the motor | $^{rad}/_{s}$ |
| $N_s$ | Specific speed | Unit-less, but actual number depends on units used in formula |
| $A_i$ | Initial cross-sectional area of outlet nozzle | $cm^2$ |
| $A_f$ | Final cross-sectional area of outlet nozzle | $cm^2$ |

Table 1: Variables

#### 3.3.2 Constraints

Now machine and practical considerations needed to be taken into account.

- There is a limit to how big the inlet can be made — this is a *micro*-turbo pump, so the system as a whole must be small
- On the other hand, there is a limit as to how small the outlet can be — the 3D printer can only print in beads of 0.007 mm
- $\omega$ and $P_{\text{in}}$ are dependent on each other in that they are both set by the motor in use
- $N_s$ has ranges for different types of fluid flow — the type of flow necessary will determine the range for $N_s$
- $b$ and $A_i$ are interdependent — $A_i = b^2$ for a square cross-section

### 3.4 Pump CAD

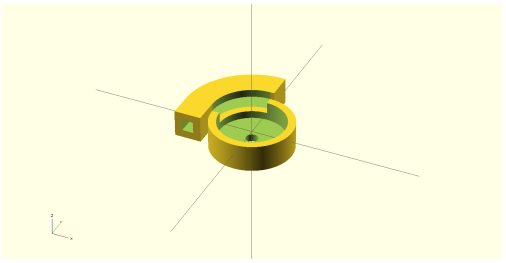
The initial pump design was taken from `http://thingiverse.com`. However, it was provided in the STL format, which is great for printing, but not so great for editing – somewhat like the PDF format, but for 3D CAD. Hence, the pump needed to be rebuilt from scratch so that parameters could be changed easily. Therefore, the software OpenSCAD (`http://openscad.org`) was used to design the pump in terms of several variables, which could easily be changed to optimize the pump. The different parts of the pump are shown in Figure 1, while the assembled pump is shown in Figure 2. The code is given in Appendix A.

### 3.5 Pump Parameter Values

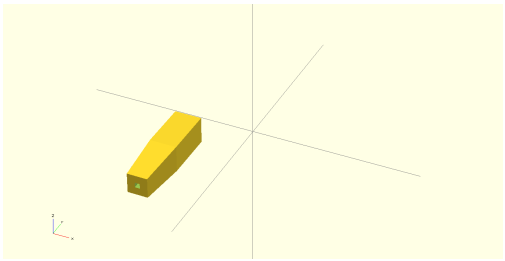
The list of design parameters used to optimize the pump is enumerated in Table 2.

| Parameter | Value |
|---|---|
| Angle | 120° |
| Height of the blade | 2 mm |
| Number of blades | 4 |
| Outer radius of the impeller | 6 mm |
| Cross-sectional area of the inlet nozzle | 20 $\text{mm}^2$ |
| Initial cross-sectional area of the outlet nozzle | 4 $\text{mm}^2$ |
| Final cross-sectional area of the outlet nozzle | 1 $\text{mm}^2$ |

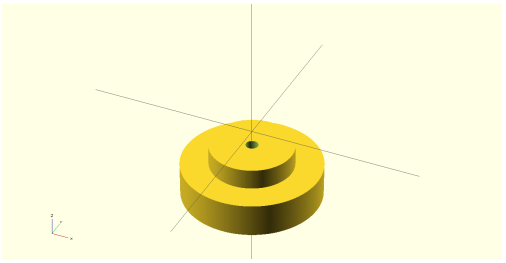
Table 2: Parameters



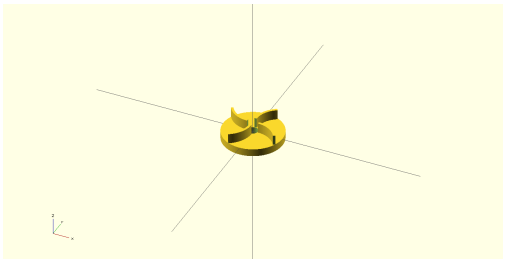
(a) Impeller Casing



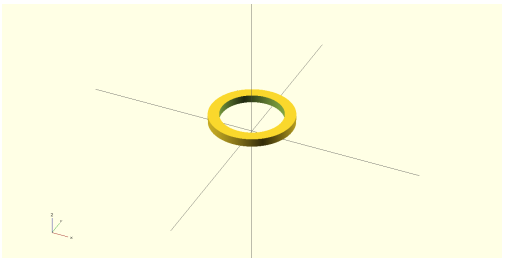
(b) Outlet Nozzle



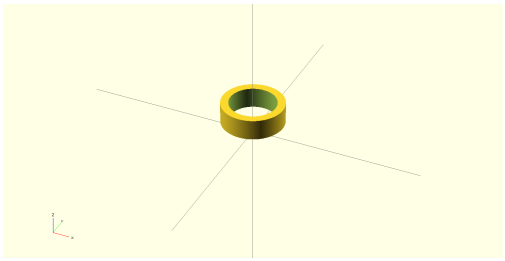
(c) Motor Casing



(d) Impeller



(e) Impeller Casing Cover



(f) Inlet Nozzle

Figure 1: Pump Parts

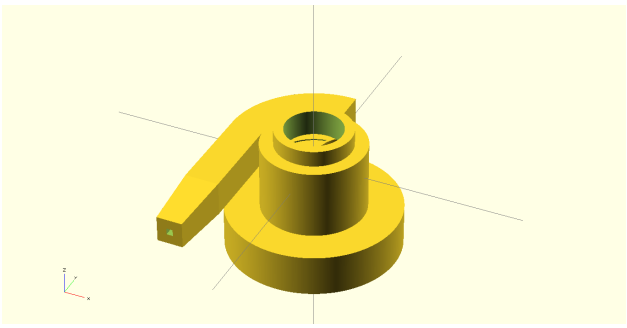


Figure 2: Assembled Pump

# 4 Calculations

Using Equations (1) to (6), one can calculate the estimated force and efficiency given values for the variables in those equations. As mentioned before, all but one variable ($r_2$) can be reduced to a number because of practical or machine considerations. Hence, the appearance of the graph will be determined by what range is chosen for $r_2$. The code given in Appendix B was used for evaluating the equations and for generating graphs such as the ones displayed in Figures 3 and 4.

An efficiency $> 1$ just means that the pump is limited by the power input, not how much power it can produce. Hence, whenever the efficiency $> 1$, the force that is produced can be calculated by solving for $v_2$ in Equation (4) (assuming $P_{\text{out}} = P_{\text{in}}$) and then solving for $F$ in Equation (6).

$$P_{\text{in}} = \dot{m}\left[h + \frac{1}{2}\left(v_2^2 - v_1^2\right)\right] \tag{7}$$

$$h = \left(\frac{\omega\sqrt{\dot{m}}}{N_s\sqrt{\rho}}\right)^{\frac{4}{3}} \tag{8}$$

$$v_1 = \frac{\dot{m}}{\rho A} \tag{9}$$

$$\frac{P_{\text{in}}}{\dot{m}} = \left(\frac{\omega\sqrt{\dot{m}}}{N_s\sqrt{\rho}}\right)^{\frac{4}{3}} + \frac{1}{2}\left(v_2^2 - \frac{\dot{m}^2}{\rho^2 A^2}\right) \tag{10}$$

$$\frac{P_{\text{in}}}{\dot{m}} - \left(\frac{\omega\sqrt{\dot{m}}}{N_s\sqrt{\rho}}\right)^{\frac{4}{3}} = \frac{1}{2}\left(v_2^2 - \frac{\dot{m}^2}{\rho^2 A^2}\right) \tag{11}$$

$$v_2^2 = \frac{\dot{m}^2}{\rho^2 A^2} + 2\left(\frac{P_{\text{in}}}{\dot{m}} - \left(\frac{\omega\sqrt{\dot{m}}}{N_s\sqrt{\rho}}\right)^{\frac{4}{3}}\right) \tag{12}$$

$$v_2 = \sqrt{\frac{\dot{m}^2}{\rho^2 A^2} + 2\left(\frac{P_{\text{in}}}{\dot{m}} - \left(\frac{\omega\sqrt{\dot{m}}}{N_s\sqrt{\rho}}\right)^{\frac{4}{3}}\right)} \tag{13}$$

For the pump analyzed above,

$$v_2 = \sqrt{\left(\frac{0.06309}{10^{-3} \times 0.63617}\right)^2 + 2\left(\frac{6000}{0.06309} - \left(\frac{963.42\sqrt{0.06309}}{2.471346\sqrt{10^{-3}}}\right)^{\frac{4}{3}}\right)} \tag{14}$$

$$\approx 331.324270896\,\text{cm/s}$$

$$F = \dot{m}\left(\frac{A_i}{A_f}\right)v_2 \tag{15}$$

$$= 0.06309\left(\frac{0.16}{0.01}\right)(446.597657553) \tag{16}$$

$$\approx 3.34451972013\,\text{N}$$

$$\frac{F}{A} \approx 3.34451972013\,\text{MPa}$$

which should be more than sufficient to activate any reasonable actuator. In fact, it may lie outside the operating range of many otherwise viable actuators.

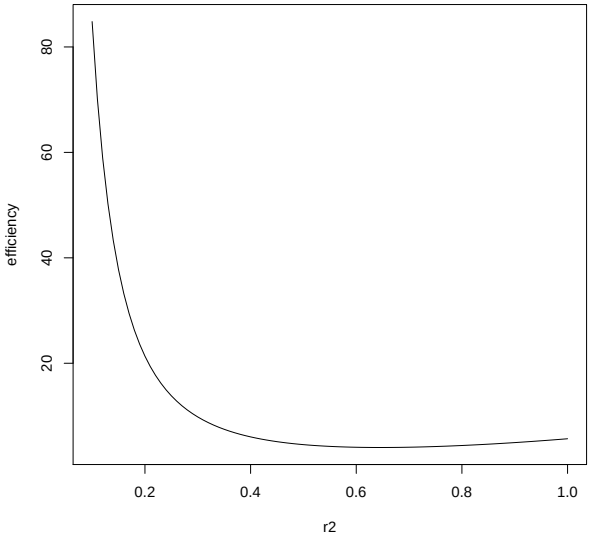



Figure 3: Efficiency versus $r_2$ (cm)

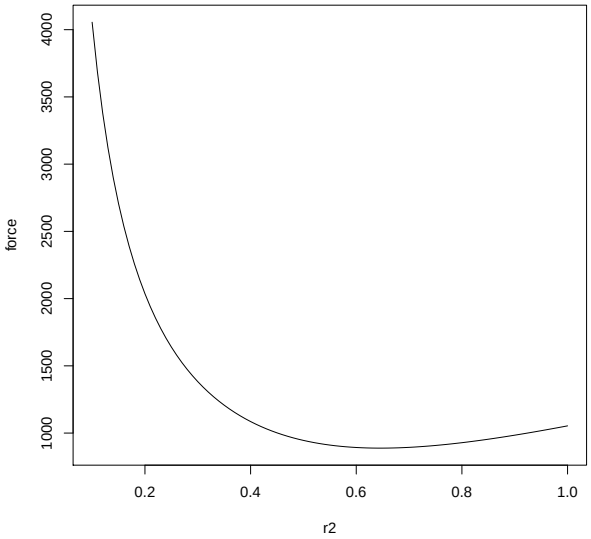



Figure 4: Force ($^{kg-cm}/_{s^2}$) versus $r_2$ (cm)

# 5 Experiment

The pump described above was projected to have an outlet pressure of $\sim 3.3\,\text{MPa}$. To test this theoretical output, an experiment was designed.

## 5.1 Equipment

See Table 3 for the list of equipment.

| Equipment | Number | Details |
|---|---|---|
| Pump | several | Manufactured by 3D printer - each one has different properties |
| Bucket | 1 | |
| Stopwatch | 1 | At least millisecond-level accuracy |
| Power Supply | 1 | Digital readout |

Table 3: Table of Equipment

## 5.2 Procedure

1. Fill the bucket with a set mass of water - say 10 kg
2. Record the voltage across the motor and the current flowing through the motor
3. Run the pump and start the timer
4. Record height of water jet
5. Stop the timer as soon as all the water has been transferred
6. Record time

## 5.3 Data Collection

| Blade Height | Input Voltage | Input Current | Time | Height of Jet |
|---|---|---|---|---|

Table 4: Data

# 6 Challenges

## 6.1 Pump

| Version | Issues | Solutions |
|---|---|---|
| 1 | • Too large | • Scaled it down |
| 2 | • Clasps broke off<br>• The pump was too small<br>• The walls were too thin (and therefore porous) | • Made clasps wider<br>• Made the impeller radius 1.5 x larger to 6 mm<br>• Increased the thickness of walls and such to 0.5 mm<br>• Fused outlet nozzle and casing<br>• Fused inlet nozzle and inlet |
| 3 | • Still too thin<br>• Clasps still broke off<br>• Might have used 0.01 mil bead size (applies to previous runs too) | • Made clasps even wider<br>• Increased the thickness of walls and such to 1 mm |
| 4 | • Impeller barely fit in casing | • Decreased impeller diameter to 5.75 mm and left the casing alone |
| 5 | • Clasps didn't work | • Attempted to print the pump all in one piece |
| 6 | • Impeller wouldn't spin - the casing size had accidentally been reset to be the same size as (4) while the impeller was the normal size | • Changed casing radius back to larger value |
| 7 | • Needed to hold motor and casing still | • Designed a holder for the motor<br>• Parameterized the motor holder such that it can be changed easily for different motor shapes |
| 8 | • Motor casing too large | • Re-measured the motor and made sure the casing would be snug around motor |
| 9 | • Motor casing shaft too tight | • Changed the radius of the shaft hole in the motor casing |

| Version | Issues | Solutions |
|---|---|---|
| 10 | • Kinetic energy of the water was not significantly changed by the pump | • Redesigned casing such that opening to outlet was only as big as the blade<br>• In addition, drastically reduced gap above impeller<br>• These two changes ensure that the water only exits after going through pump |
| 11 | • Kinetic energy of the water still not changed after going through pump | • Made radius much larger<br>• Made blade height much larger |
| 12 | • Stillbirth (impeller won't move) | • Found a trivial mistake with the positioning of the motor casing<br>• Changed blade thickness to try to make thicker blades |
| 13 | • Shaft hole slightly off on impeller | • Changed blade thickness to be what it was in (11)<br>• Changed the casing so that the shaft would penetrate all the way to the top of the blades<br>• Shortened inlet and made it wider |
| 14 | • WORKS! | • Made blade height = 2 mm<br>• Made radius = 6 mm<br>• Made exit hole = 1 $mm^2$<br>• Made support height below impeller smaller (0.25 mm, used to be 0.35 mm) so that there is less room for the water to leave without going through the impeller<br>• Changed blade angle to 120 degrees so that the blade fits inside the impeller |
| 15 | • Cavitation | • Further research |

Table 5: Versions of the pump and what went wrong

Table 5 summarizes the challenges I faced in manufacturing this pump.

### 6.2 Actuator

The first time I printed the actuator, it wouldn't move - the tolerances were too tight in the STL. When I reprinted it, it started moving.

### 6.3 General

The main challenge throughout this research was time. If the research is purely theoretical, the turn-around time from fixing something to seeing a result is minimal. In addition, one can run a computer program even while one sleeps. Not so with 3D printing. Because of the design of this pump, it must be printed with support material. This means it needs to go in a special bath to dissolve that support material. Printing itself takes a few hours - 2 to 3 hours for a typical pump. However, because of the scale of the pump, it must be left in the bath overnight, which means it is much harder to quickly correct a mistake and reprint something. Instead of taking hours, it takes a day for each iteration.

## 7 Results

### 7.1 The Good

The pump worked — sometimes. Pump (14) worked fairly well, but it was not tested. In accord with a previous version of the experiment, the exit hole was facing the same direction as the inlet pipe. However, I had to modify my experiment in order to make it easier to get measurements. The design I had used was good for an infinite supply from the reservoir, but bad for measuring kinetic energy. Pump (15) had cavitation issues which prevented it from working all the time.

### 7.2 The Bad

Because of a lack of time, only one configuration was tested — the so-called optimized one developed above. The stream flew $\approx 0.3302\,\mathrm{m}$ into the air. This means the water was traveling $\sqrt{2 \times 9.80665 \times 0.3302} \approx 2.545\,{}^{\mathrm{m}}/_{\mathrm{s}}$ at the point of exit, which means $v_2 \approx 0.159\,{}^{\mathrm{m}}/_{\mathrm{s}}$, which is pitiful. A measurement of the mass flow rate would enable full analysis of the pump as detailed by the experiment.

One might notice that not all the data required by the experiment was collected. This is because the pump failed to work after the initial trial with the measurement made above. Instead, froth built up around the inlet. This indicates something called "Discharge Cavitation", where the fluid, instead of exiting, simply keeps circulating inside the pump. In doing so, it must pass through the tiny clearance between the impeller and the casing, which creates a vacuum. This in turn causes the liquid to turn into a vapor.

In addition, a pump connected to an actuator was tested. Because of the cavitation, running the pump actually caused the actuator to contract rather than expand. This was puzzling at first, until it was understood that the pump was undergoing cavitation. This was verified by attaching a makeshift actuator constructed out of a pen to the larger pump that always works — the actuator did what was expected and expanded.

## 8 Discussion

### 8.1 Significance

The significance of a one-piece pump-actuator combination is tremendous. Printing the pump in one piece means there are no joints to break loose, no glue to break off, and no screws to come loose. A one-piece

pump ensures that leakage is minimal, as there are no joints through which water could force its way out. Furthermore, it is extremely durable because of the processes under which it is manufactured.

In addition, printing the actuator attached to the pump has its own benefits. The need for seals to secure the actuator to the pump is gone and the likelihood that a leak will emerge is minimized.

### 8.2 Cavitation

The pump that was tested was wracked by cavitation. However, there are a couple of things which may help with that problem.

- A larger pump — one of the earlier pumps I tested (the first one that worked) was much larger and didn't ever undergo cavitation
- A different working fluid — different fluids vaporize at different temperatures, so a different working fluid may not undergo cavitation even though water does

### 8.3 Inefficiencies

In addition, there are inefficiencies in the system. The major one is that the base of the impeller can be heard grinding against the bottom of the casing. This might explain why, even when there was a reading, it wasn't as good as predicted. Another explanation might be that there was still cavitation, just not enough to impede the flow completely.

## 9 Conclusions

The advent of 3D printing technology has ushered in a new era of technological advances. It enables stronger, more durable, and more versatile structures with the use of support material to allow one to print moving assemblies in one piece. The goal of this project was to develop a working micro-turbo pump-actuator assembly which could then be used for stabilization, manipulation, ambulation, and so on.

The present design takes advantage of a 3D printer's capability to print moving assemblies in one piece to create a strong, durable pump-actuator assembly. However, it suffers from cavitation problems, which can be rectified by suitable modification of the parameters.

Further research will need to be conducted to see how the pump reacts to different working fluids and whether a fluid other than water would be better for the particular application. In addition, the inefficiencies in the pump need to be addressed in some way — maybe by somehow ensuring a gap between the bottom of the impeller and the bottom of the casing.

## A OpenSCAD Code

### A.1 Pump

#### A.1.1 Variables

Program 1: variables.scad

```
/*
  Units are mm
*/
```

```
/*
  OpenSCAD Variables
  ------------------
  Shouldn't need to be modified
*/

$fn = 400;

/*
  General Variables
  -----------------
  Not part of any part in particular
*/

thickness = 1.5;
sup_h = 0.25;

/*
  Blade variables
  ---------------
  Variables necessary for creating the blade
*/

angle = 120;
h = 2;
rad_i = 10;
rad_o = 11;

/*
  Motor variables
  ---------------
  Variables which characterize the motor
*/

sh = 9;
sr = 1;
bh = 2;
br = 3;
mr = 12;
mh = 5;
wt = thickness;

/*
  Impeller variables
  ------------------
  Variables needed to generate the impeller
*/

number = 4;
imp_rad_i = 1.1 * sr;
imp_rad_o = 6;
base_t = thickness;

/*
  Casing variables
  ----------------
  Variables necessary to generate the pump casing
*/
```

```
cas_rad = imp_rad_o;
mult_square = 1.1;
mult_round = 1.05;
pipe_l = 5;
pipe_t = thickness;
cas_h = h + sup_h;
cas_t = thickness;
cas_extend = 0.1;
gap_ang = 90;
pipe_w = cas_h;

/*
  Cover variables
  ---------------
  Variables necessary to generate the casing cover
*/

cov_rad_r = mult_round * cas_rad + cas_t;
cov_rad_s = mult_square * cas_rad + cas_t;
cov_tt = thickness;

/*
  Nozzle variables
  ----------------
  Variables used to generate the inlet and outlet nozzles
*/

out_noz_ai_square = pow(pipe_w, 2);
out_noz_ai_round = PI * pow(pipe_w, 2);
out_noz_af = 1;
out_noz_hb = 10;
out_noz_ht = 10;
out_noz_t = thickness;
in_noz_ai = PI * pow(3 * cas_rad / 4, 2);
in_noz_af = in_noz_ai;
in_noz_ht = 2;
in_noz_hb = 2;
in_noz_t = thickness;

/*
  Miscellaneous variables
  -----------------------
  Variables which don't fit in any good category
*/

// How far up the casing needs to be moved to rest on the x-y plane
diff_casing_z = cas_h / 2;
// Output nozzle width
out_noz_wf = sqrt(out_noz_af);
// Parameters for constructing the corner pipe
hose_rad_i = out_noz_t;
hose_rad_o = hose_rad_i + out_noz_wf;
// How far the pipe should extend beyond the turn
out_noz_h = 0;
// Allows shaft to slip through casing
cas_sr = 1.2 * sr;
// Calculates how much extra shaft there is
cas_sh = sh - base_t - cas_t - h;
// Ensures less support material is used
```

```
cas_st = mult_square * cas_rad - cas_sr + cas_t;
cas_bt = mult_square * cas_rad - br + cas_t;
```

### A.1.2 Geometry

Program 2: geometry.scad

```
/*
  Generates a sector of a cylinder
  --------------------------------
  module: makeSector
  rad: radius
  h: height
  angle: angle swept by the sector
*/

module makeSector(rad, h, angle) {
  difference() {
    cylinder(r = rad, h = h, center = true);
    translate([0, -rad, 0]) {
      cube(2 * rad, center = true);
    }
    rotate([0, 0, 180 - angle]) {
      translate([0, -rad, 0]) {
        cube(2 * rad, center = true);
      }
    }
  }
}

/*
  Generates a torus
  -----------------
  module: makeTorus
  inner_rad: inner radius
  outer_rad: outer_radius
*/

module makeTorus(inner_rad, outer_rad) {
  rotate_extrude($fn = 100) {
    translate([inner_rad + (outer_rad - inner_rad) / 2, 0, 0]) {
      circle(r = (outer_rad - inner_rad) / 2);
    }
  }
}

/*
  Generates a pipe that turns corners (square cross-section)
  ----------------------------------------------------------------------
  module: squareCornerPipe
  rad_i: inner radius
  rad_o: outer radius
  h: height
  angle: swept angle
  thickness: thickness
*/

module squareCornerPipe(rad_i, rad_o, h, angle, thickness) {
  difference() {
```

```
    union() {
      difference() {
        makeSector(rad_o + thickness, h + 2 * thickness, angle);
        rotate([0, 0, 0.95 * angle]) {
          makeSector(rad_i - thickness, 1.1 * (h + 2 * thickness), 2.2 * angle);
        }
      }
    }
    union() {
      difference() {
        makeSector(rad_o, h, angle);
        rotate([0, 0, 0.95 * angle]) {
          makeSector(rad_i, 1.1 * (h), 2.2 * angle);
        }
      }
    }
  }
}

/*
  Generates a connecting pipe from the opening in the casing to the exit pipe
  ---------------------------------------------------------------------------
  module: roundCornerPipe
  rad_i: inner radius
  rad_o: outer radius
  angle: swept angle
  thickness: thickness of pipe
*/

module roundCornerPipe(rad_i, rad_o, angle, thickness) {
  difference() {
    makeTorus(rad_i - thickness, rad_o + thickness);
    makeTorus(rad_i, rad_o);
    translate([0, -(rad_o + thickness), 0]) {
      cube(2 * (rad_o + thickness), center = true);
    }
    rotate([0, 0, angle]) {
      translate([0, -(rad_o + thickness), 0]) {
        cube(2 * (rad_o + thickness), center = true);
      }
    }
  }
}

/*
  Section of pipe (square cross-section)
  --------------------------------------
  module: squarePipe
  pipe_width: width and height of pipe
  pipe_l: length of pipe
  pipe_t: thickness of pipe
*/

module squarePipe(pipe_width, pipe_l, pipe_t) {
  difference() {
    cube([pipe_width + 2 * pipe_t, pipe_width + 2 * pipe_t, pipe_l], center = true
        );
    cube([pipe_width, pipe_width, 2 * pipe_l], center = true);
  }
```

```
}

/*
  Section of pipe (round cross-section)
  -------------------------------------
  module: roundPipe
  pipe_width: diameter of pipe
  pipe_l: length of pipe
  pipe_t: thickness of pipe
*/

module roundPipe(pipe_width, pipe_l, pipe_t) {
  difference() {
    cylinder(r = pipe_width / 2 + pipe_t, h = pipe_l, center = false);
    cylinder(r = pipe_width / 2, h = 2 * pipe_l, center = true);
  }
}

/*
  Creates a square pyramid with the top chopped off
  -------------------------
  module: squarePyramid
  bw: width of one side of the base
  tw: width of one side of the top
  h: height of the pyramid
  o: origin in the z direction
*/

module squarePyramid(bw, tw, h, o) {
  polyhedron(points = [[bw, bw, o], [bw, -bw, o],
                       [-bw, -bw, o], [-bw, bw, o],
                       [tw, tw, o + h], [tw, -tw, o + h],
                       [-tw, -tw, o + h], [-tw, tw, o + h]],
             triangles = [[0, 1, 5, 4], [1, 2, 6, 5],
                          [2, 3, 7, 6], [3, 0, 4, 7],
                          [1, 0, 3], [2, 1, 3],
                          [4, 5, 7], [5, 6, 7]]);
}
```

### A.1.3 Casing

Program 3: mCasing.scad

```
/*
  Units are mm
*/

/*
  Variables
*/

include <variables.scad>;

/*
  Geometry
*/

include <geometry.scad>;
```

```
/*
  Modules
*/

/*
  Generates the main portion of the casing
  ----------------------------------------
  module: casingBody
  rad: casing radius
  h: casing height
  thickness: casing thickness
  sr: shaft radius
*/

module casingBody(rad, h, thickness) {
  difference() {
    cylinder(r = rad + thickness, h = h, center = true);
    cylinder(r = rad, h = 2.2 * h, center = true);
  }
}

module casingBottom(rad, h, thickness, sr) {
  difference() {
    cylinder(r = rad + thickness, h = h + thickness, center = true);
    translate([0, 0, thickness]) {
      cylinder(r = rad, h = h + thickness, center = true);
    }
    cylinder(r = sr, h = 100 * h, center = true);
  }
}

/*
  Utilizes the modules above to create a casing
  for the impeller with a square exit pipe
  ----------------------------------------
  module: mSquareCasing
  cas_rad: casing radius
  cas_h: casing height
  cas_t: casing thickness
  sr: shaft radius
  angle: swept angle to clear out for fluid flow
  pipe_w: width and height of pipe
  pipe_l: length of (exit) pipe
  pipe_t: thickness of pipe
  -------------------------
  cp_rad_i: connecting pipe inner radius
  cp_rad_o: connecting pipe outer radius
  cp_h: connecting pipe height
  cap_w: width and height of pipe cap
         (closes off end which is not connected to nozzle)
  cap_h: thickness of pipe cap
*/

module mSquareCasing(cas_rad, cas_h, cas_t, h, bh, sr, angle, pipe_w, pipe_l, pipe
    _t) {
  cp_rad_i = mult_square * cas_rad + 2 * cas_t;
  cp_rad_o = cp_rad_i + pipe_w;
  cp_h = pipe_w;
  cap_w = pipe_w + 2 * pipe_t;
```

```
  cap_h = pipe_w + pipe_t;
  difference() {
    union() {
      casingBody(mult_square * cas_rad, cas_h, cas_t);
      translate([-mult_square * cas_rad - cas_t - pipe_w / 2 - pipe_t, -pipe_l /
          2, 0]) {
        rotate([90, 0, 0]) {
          squarePipe(pipe_w, pipe_l, pipe_t);
        }
      }
      squareCornerPipe(cp_rad_i, cp_rad_o, cp_h, angle, pipe_t);
      translate([(pipe_t - cap_h) / 2, mult_square * cas_rad + cas_t + cap_w / 2,
          0]) {
        difference() {
          cube([cap_h, cap_w, cap_w], center = true);
          translate([-pipe_t / 2, 0, 0]) {
            cube([cap_h, 2 * cap_w, 2 * cap_w], center = true);
          }
        }
      }
    }
    translate([0, 0, 0]) {
      makeSector(cas_rad + 2 * cas_t + pipe_w, cas_h, angle);
    }
  }
  translate([0, 0, -(bh + cas_t + cas_h) / 2]) {
    casingBottom(mult_square * cas_rad, bh, cas_t, sr);
  }
}

/*
  Utilizes the above modules to generate a casing
  for the impeller with a round exit pipe
  ---------------------------------------
  module: mRoundCasing
  cas_rad: casing radius
  cas_h: casing height
  cas_t: casing thickness
  sr: shaft radius
  angle: swept angle to clear out for fluid flow
  pipe_w: width and height of pipe
  pipe_l: length of (exit) pipe
  pipe_t: thickness of pipe
  -------------------------
  cp_rad_i: connecting pipe inner radius
  cp_rad_o: connecting pipe outer radius
*/

module mRoundCasing(cas_rad, cas_h, cas_t, sr, angle, pipe_w, pipe_l, pipe_t) {
  cp_rad_i = cas_rad + cas_t;
  cp_rad_o = cas_rad + pipe_w + cas_t;
  difference() {
    union() {
      casingBody(mult_round * cas_rad, cas_h, cas_t, sr);
      translate([-cas_rad - pipe_w / 2 - cas_t, -pipe_l / 2, 0]) {
        rotate([90, 0, 0]) {
          roundPipe(pipe_w, pipe_l, pipe_t);
        }
      }
```

```
      roundCornerPipe(cp_rad_i, cp_rad_o, angle, pipe_t);
      translate([-pipe_w / 2, cas_rad + pipe_w / 2 + cas_t, 0]) {
        rotate([0, 90, 0]) {
          difference() {
            cylinder(r = pipe_w / 2 + pipe_t, h = cas_h + pipe_t, center = true);
            translate([0, 0, -pipe_t / 2]) {
              cylinder(r = 2 * (pipe_w + pipe_t), h = cas_h + pipe_t, center =
                  true);
            }
          }
        }
      }
    }
    makeSector(cas_rad + cas_t + 0.5 * pipe_w, cas_h - 2 * cas_t, angle);
  }
}

/*
  End of modules
*/
```

### A.1.4 Exit Pipe

Program 4: mExitPipe.scad

```
/*
  Units are mm
*/

/*
  Variables
*/

include <variables.scad>;

/*
  Geometry
*/

include <geometry.scad>;

/*
  Modules
*/

/*
  Generates the pipe for changing flow direction
  ----------------------------------------------
  module: mSquareExitPipe
  rad_i: inner radius
  rad_o: outer radius
  noz_wf: width of nozzle
  ang: angle
  pipe_t: pipe thickness
  noz_h: height (length) of nozzle
  noz_t: thickness of nozzle
*/

module mSquareExitPipe(rad_i, rad_o, noz_wf, ang, pipe_t, noz_h, noz_t) {
```

```
  squareCornerPipe(rad_i, rad_o, noz_wf, ang, pipe_t);
  translate([noz_h / 2, (rad_i + rad_o) / 2, 0]) {
    rotate([0, 90, 0]) {
      squarePipe(noz_wf, noz_h, noz_t);
    }
  }
}

/*
  End of modules
*/
```

### A.1.5 Impeller

Program 5: mImpeller.scad

```
/*
  Units are mm
*/

/*
  Variables
*/

include <variables.scad>;

/*
  Modules
*/

/*
  Generate one blade of the impeller
  ----------------------------------
  module: blade
  h: thickness of the blade in the "z" direction
  angle: angle of tilted end with horizontal
  rad_i: inner radius of blade
  rad_o: outer radius of blade
  rad_o - rad_i: thickness of the blade in the radial direction
*/

module blade(h, angle, rad_i, rad_o) {
  render(convexity = 10) {
    translate([(rad_o + rad_i) / 2, 0, 0]) {
      difference() {
        linear_extrude(height = h, center = true) {
          difference() {
            circle(rad_o, center = true);
            circle(rad_i, center = true);
          }
        }
        translate([0, -1.1 * rad_o, 0]) {
          cube(size = [rad_o * 2.2, 2.2 * rad_o, 1.1 * h], center = true);
        }
        rotate([0, 0, angle]) {
          translate([0, -1.1 * rad_o, 0]) {
            cube(size = [2.2 * rad_o, 2.2 * rad_o, 1.1 * h], center = true);
          }
        }
```

```
      }
    }
  }
}

/*
  Generate an impeller
  --------------------
  module: impeller
  number: number of blades
  imp_rad_i: how far away from the center the inner edges of the blades should be
  imp_rad_o: how large the impeller should be
  h: height of the blades
  angle: angle swept by the blade
  rad_i: blade inner radius (only necessary for constructing the blade)
  rad_o: blade outer radius (only necessary for constructing the blade)
*/

module impeller(number, imp_rad_i, imp_rad_o, h, angle, rad_i, rad_o) {
  x = rad_o * sin(180 - angle) / sin((angle) / 2);
  for(i = [1:number]) {
    rotate([0, 0, i * 360 / number]) {
      translate([imp_rad_i * cos(360 / number),
                 imp_rad_i * sin(360 / number),
                 0]) {
        rotate([0, 0, 0]) {
          scale([imp_rad_o / x * sin(angle / 2),
                 imp_rad_o / x * sin(angle / 2),
                 1]) {
            blade(h, angle, rad_i, rad_o);
          }
        }
      }
    }
  }
}

/*
  Generates the base of the impeller
  ----------------------------------
  module: impeller_base
  imp_rad_o: impeller outer radius
  thickness: impeller base thickness
  sr: shaft radius
*/

module impeller_base(imp_rad_o, thickness, sr) {
  r = 0.2;
  pad = 0.01;
  difference() {
    cylinder(r = imp_rad_o, h = thickness, center = true);
    cylinder(r = sr, h = 1.01 * thickness, center = true);
  }
}

/*
  Generates a complete impeller
  -----------------------------
  module: mImpeller
```

```
  n: number of blades
  imp_rad_i: impeller inner radius
  imp_rad_o: impeller outer radius
  h: blade height
  angle: angle swept out by the impeller blade
  rad_i: blade inner radius (necessary only for creating the blade)
  rad_o: blade outer radius (necessary only for creating the blade)
  bt: impeller base thickness
  sr: shaft radius
*/

module mImpeller(n, imp_rad_i, imp_rad_o, h, angle, rad_i, rad_o, bt, sr) {
  impeller(n, imp_rad_i, imp_rad_o, h, angle, rad_i, rad_o);
  translate([0, 0, -(h/2 + bt/2)]) {
    impeller_base(imp_rad_o, bt, sr);
  }
}

/*
  End of modules
*/
```

### A.1.6 Inlet

Program 6: mInlet.scad

```
/*
  Units are mm
*/

/*
  Variables
*/

include <variables.scad>;

/*
  Modules
*/

/*
  Build the bulk of the cover
  ---------------------------
  module: cover
  cov_rad: cover radius
  cov_tt: cover thickness
*/

module cover(cov_rad, cov_tt) {
  translate([0, 0, cov_tt / 2]) {
    cylinder(r = cov_rad, h = cov_tt, center = true);
  }
}

module casExtend(cas_r, cas_t, cas_h) {
  translate([0, 0, cas_h / 2]) {
    difference() {
      cylinder(r = cas_r + cas_t, h = cas_h, center = true);
      cylinder(r = cas_r, h = 2 * cas_h, center = true);
```

```
    }
  }
}

/*
  Creates a cover with a hole for the (square) inlet nozzle
  ---------------------------------------------
  module: mSquareInlet
  cas_h: casing height
  cov_rad: cover radius
  cov_tt: cover thickness
  in_a: inlet nozzle final area
  in_noz_t: inlet nozzle thickness
  ------------------------------
  hole_w: width (and height) of the hole for the nozzle
*/

module mSquareInlet(cas_h, cov_rad, cov_tt, in_a, in_noz_t) {
  hole_w = sqrt(in_a) + in_noz_t;
  difference() {
    cover(cov_rad, cov_tt);
    cube([hole_w, hole_w, 100 * (cas_h + cov_tt)], center = true);
  }
}

/*
  Creates a cover with a hole for the (round) inlet nozzle
  ---------------------------------------------------------
  module: mRoundInlet
  cas_h: casing height
  cov_rad: cover radius
  cov_tt: cover thickness
  in_a: inlet nozzle final area
  in_noz_t: inlet nozzle thickness
  ------------------------------
  hole_r: radius of the hole for the nozzle
*/

module mRoundInlet(cas_r, cas_t, cas_h, cov_rad, cov_tt, in_a, in_noz_t) {
  hole_r = sqrt(in_a / PI) + in_noz_t;
  difference() {
    cover(cov_rad, cov_tt);
    cylinder(r = hole_r, h = 100 * (cas_h + cov_tt), center = true);
  }
  translate([0, 0, -cas_h]) {
    casExtend(cas_r, cas_t, cas_h);
  }
}

/*
  End of modules
*/
```

### A.1.7 Motor Casing

Program 7: mMotorCasing.scad

```
/*
  Units are mm
```

```
*/

/*
  Variables
*/

include <variables.scad>;

/*
  Modules
*/

/*
  Creates a holder for the extra shaft
  ------------------------------------
  module: shaft_holder
  sr: shaft radius
  sh: shaft height
  st: shaft casing thickness
*/

module shaft_holder(sr, sh, st) {
  difference() {
    cylinder(r = sr + st, h = sh, center = true);
    cylinder(r = sr, h = 2 * sh, center = true);
  }
}

/*
  Creates a holder for the bearing at the base of the shaft
  ---------------------------------------------------------
  module: bearing_holder
  sr: shaft radius
  br: bearing radius
  bh: bearing height
  bt: bearing casing thickness
*/

module bearing_holder(sr, br, bh, brt, bht) {
  difference() {
    cylinder(r = br + brt, h = bh + bht, center = true);
    translate([0, 0, -bht / 2 - 1]) {
      cylinder(r = br, h = 1 + bh, center = true);
    }
    cylinder(r = sr, h = 100 * bh, center = true);
  }
}

/*
  Creates a casing for the (round) motor
  --------------------------------------
  module: round_motor_holder
  sr: shaft radius
  br: bearing radius
  mr: motor radius
  mh: motor height
  mt: motor casing thickness
*/
```

```
module round_motor_holder(sr, br, mr, mh, mt) {
  difference() {
    cylinder(r = mr + mt, h = mh + mt, center = true);
    translate([0, 0, -mt / 2 - 1]) {
      cylinder(r = mr, h = 1 + mh, center = true);
    }
    cylinder(r = sr, h = 100 * mh, center = true);
    cylinder(r = br, h = 100 * mh, center = true);
  }
}

/*
  Puts the above modules together to create a casing for the motor
  which can be used to restrain the motor from moving
  ---------------------------------------------------
  module: mRoundMotorCasing
  sr: shaft radius
  sh: shaft height
  st: shaft casing thickness
  br: bearing radius
  bh: bearing height
  bwt: bearing casing thickness (wall)
  btt: bearing casing thickness (top)
  mr: motor radius
  mh: motor height
  mt: motor casing thickness
*/

module mRoundMotorCasing(sr, sh, st, br, bh, brt, bht, mr, mh, mt) {
  translate([0, 0, -sh / 2]) {
    shaft_holder(sr, sh, st);
    translate([0, 0, -(bh + bht) / 2 - sh / 2 + bht]) {
      bearing_holder(sr, br, bh, brt, bht);
    }
    translate([0, 0, -(mh + mt) / 2 - sh / 2]) {
      round_motor_holder(sr, br, mr, mh, mt);
    }
  }
}

/*
  End of modules
*/
```

### A.1.8 Nozzle

Program 8: mNozzle.scad

```
/*
  Units are mm
*/

/*
  Variables
*/

include <variables.scad>;

/*
```

```
  Geometry
*/

include <geometry.scad>;

/*
  Modules
*/

/*
  Creates the top (narrowing part) of the (round) nozzle
  ------------------------------------------------------
  module: roundNozzleTop
  area_i: initial cross-sectional area of the nozzle
  area_f: final cross-sectional area of the nozzle
  h: height (or length) of this part of the nozzle
  thickness: thickness of the walls of the nozzle
  ----------------------------------------------
  r1: initial radius of the nozzle
  r2: final radius of the nozzle
*/

module roundNozzleTop(area_i, area_f, h, thickness) {
  r1 = sqrt(area_i / PI);
  r2 = sqrt(area_f / PI);
  difference() {
    cylinder(r1 = r1 + thickness, r2 = r2 + thickness, h = h, center = false);
    cylinder(r1 = r1, r2 = r2, h = h, center = false);
  }
}

/*
  Puts the above modules together to form a complete (round) nozzle
  -----------------------------------------------------------
  module: mRoundNozzle
  area_i: initial cross-sectional area of the nozzle
  area_f: final cross-sectional area of the nozzle
  bh: height (or length) of the straight portion of the nozzle
  th: height (or length) of the narrowing portion of the nozzle
  thickness: thickness of the walls of the nozzle
*/

module mRoundNozzle(area_i, area_f, bh, th, thickness) {
  union() {
    roundPipe(2 * sqrt(area_i / PI), bh, thickness);
    translate([0, 0, bh]) {
      roundNozzleTop(area_i, area_f, th, thickness);
    }
  }
}

/*
  Creates the top (narrowing part) of the (square) nozzle
  -------------------------------------------------------
  module: squareNozzleTop
  area_i: initial cross-sectional area of the nozzle
  area_f: final cross-sectional area of the nozzle
  h: height (or length) of this part of the nozzle
  thickness: thickness of the walls of the nozzle
```

```
  ------------------------------------------------
  r1: initial width of the nozzle
  r2: final width of the nozzle
*/

module squareNozzleTop(area_i, area_f, h, thickness) {
  r1 = sqrt(area_i);
  r2 = sqrt(area_f);
  difference() {
    squarePyramid(r1 / 2 + thickness, r2 / 2 + thickness, h, 0);
    squarePyramid(r1 / 2, r2 / 2, h, 0);
  }
}

/*
  Puts the above modules together to form a complete (square) nozzle
  ------------------------------------------------------------
  module: mSquareNozzle
  area_i: initial cross-sectional area of the nozzle
  area_f: final cross-sectional area of the nozzle
  bh: height (or length) of the straight portion of the nozzle
  th: height (or length) of the narrowing portion of the nozzle
  thickness: thickness of the walls of the nozzle
*/

module mSquareNozzle(area_i, area_f, bh, th, thickness) {
  translate([0, 0, bh / 2]) {
    union() {
      squarePipe(sqrt(area_i), bh, thickness);
      translate([0, 0, bh / 2]) {
        squareNozzleTop(area_i, area_f, th, thickness);
      }
    }
  }
}

/*
  End of modules
*/
```

### A.1.9 Pump

Program 9: mPump.scad

```
/*
  Units are mm
*/

/*
  Casing modules
*/

include <mCasing.scad>;

/*
  Impeller modules
*/

include <mImpeller.scad>;
```

```
/*
  Inlet modules
*/

include <mInlet.scad>;

/*
  Nozzle modules
*/

include <mNozzle.scad>;

/*
  Motor Casing modules
*/

include <mMotorCasing.scad>;

/*
  Exit pipe module
*/

include <mExitPipe.scad>;

/*
  Generate pump
*/

module mPumpWithExitCorner() {

  difference() {

    union() {

      translate([0, 0, diff_casing_z]) {
        mSquareCasing(cas_rad, cas_h, cas_t, h, base_t, cas_sr, gap_ang, pipe_w,
            pipe_l, pipe_t);
      }

      translate([-pipe_w / 2 - pipe_t - mult_square * cas_rad - cas_t, -pipe_l,
          diff_casing_z]) {
        rotate([90, 180, 0]) {
          mSquareNozzle(out_noz_ai_square, out_noz_af,
                        out_noz_hb, out_noz_ht, out_noz_t);
          translate([0, (hose_rad_i + hose_rad_o) / 2, out_noz_hb + out_noz_ht]) {
            rotate([90, 0, 90]) {
              mSquareExitPipe(hose_rad_i, hose_rad_o, out_noz_wf, 90, pipe_t,
                              out_noz_h, out_noz_t);
            }
          }
        }
      }

      translate([0, 0, -base_t - cas_t]) {
        mRoundMotorCasing(cas_sr, cas_sh, cas_st, br, bh, cas_bt, 0.5, mr, mh, wt)
            ;
      }
```

```
      translate([0, 0, h / 2 + sup_h]) {
        mImpeller(number, imp_rad_i, imp_rad_o, h, angle, rad_i, rad_o, base_t, sr
            );
      }

      translate([0, 0, cas_h + cas_extend]) {
        rotate([0, 0, 90]) {
          mRoundInlet(mult_square * cas_rad, cas_t, cas_extend, cov_rad_s, cov_tt,
              in_noz_af, in_noz_t);
        }
        mRoundNozzle(in_noz_af, in_noz_ai, in_noz_hb, in_noz_ht, in_noz_t);
      }
    }

    // translate([0, -50, 0]) {
    //   cube(100, center = true);
    // }
  }
}

module mPumpWithoutExitCorner() {

  difference() {

    union() {

      translate([0, 0, diff_casing_z]) {
        mSquareCasing(cas_rad, cas_h, cas_t, h, base_t, cas_sr, gap_ang, pipe_w,
            pipe_l, pipe_t);
      }

      translate([-pipe_w / 2 - pipe_t - mult_square * cas_rad - cas_t, -pipe_l,
          diff_casing_z]) {
        rotate([90, 180, 0]) {
          mSquareNozzle(out_noz_ai_square, out_noz_af,
                        out_noz_hb, out_noz_ht, out_noz_t);
        }
      }

      translate([0, 0, -base_t - cas_t]) {
        mRoundMotorCasing(cas_sr, cas_sh, cas_st, br, bh, cas_bt, 0.5, mr, mh, wt)
            ;
      }

      translate([0, 0, h / 2 + sup_h]) {
        mImpeller(number, imp_rad_i, imp_rad_o, h, angle, rad_i, rad_o, base_t, sr
            );
      }

      translate([0, 0, cas_h + cas_extend]) {
        rotate([0, 0, 90]) {
          mRoundInlet(mult_square * cas_rad, cas_t, cas_extend, cov_rad_s, cov_tt,
              in_noz_af, in_noz_t);
        }
        mRoundNozzle(in_noz_af, in_noz_ai, in_noz_hb, in_noz_ht, in_noz_t);
      }
    }

    // translate([0, -50, 0]) {
```

```
    //    cube(100, center = true);
    // }
  }
}

mPumpWithoutExitCorner();
```

## A.2 Actuator

### A.2.1 Variables

Program 10: variables.scad

```
/*
  Units are mm
*/

/*
  Global variables
*/

$fn = 400;
sup_h = 1;

/*
  Actuator variables
*/

stroke = 30;
in_cyl_r = 5;
cyl_th = 1.5;
cap_r = in_cyl_r + 3;
cap_h = 1;
lip_h = 1.5;
bot_h = 1;
hole_area = 1;

/*
  Slot variables
*/

slot_w = 3;
slot_h = 5;
slot_th = 2 * cyl_th;
slot_loc_h = stroke / 2;
```

### A.2.2 Actuator

Program 11: actuator.scad

```
/*
  Units are mm
*/

/*
  Variables
*/

include <variables.scad>
```

```
module actuatorInside(cyl_rad, cyl_height, cap_rad, cap_height) {
  translate([0, 0, cap_height]) {
    cylinder(r = cyl_rad, h = cyl_height, center = false);
  }
  cylinder(r = cap_rad, h = cap_height, center = false);
}

module actuatorOutside(cyl_rad, cyl_height, cyl_th, lip_rad, lip_th, slot_th, slot
    _w, slot_h, slot_loc_h) {
  difference() {
    cylinder(r = cyl_rad, h = cyl_height, center = false);
    cylinder(r = lip_rad, h = 100 * cyl_height, center = true);
    translate([0, 0, -lip_th]) {
      cylinder(r = cyl_rad - cyl_th, h = cyl_height, center = false);
    }
    translate([cyl_rad - slot_th / 2, 0, 1 + slot_loc_h]) {
      cube([slot_th, slot_w, slot_h], center = true);
    }
  }
}

module actuatorBottomSquareHole(bot_r, bot_h, hole_area) {
  difference() {
    cylinder(r = bot_r, h = bot_h, center = false);
    cube([sqrt(hole_area), sqrt(hole_area), 100 * bot_h], center = true);
  }
}

module mActuator(stroke, in_cyl_r, cyl_th, cap_r, cap_h, lip_h, bot_h, hole_area,
    slot_th, slot_w, slot_h, slot_loc_h) {
  out_cyl_h = stroke + lip_h;
  out_cyl_r = 1.2 * cap_r + cyl_th;
  in_cyl_h = 1.5 * (out_cyl_h - cap_h);
  translate([0, 0, bot_h]) {
    translate([0, 0, sup_h]) {
      actuatorInside(in_cyl_r, in_cyl_h, cap_r, cap_h);
    }
    actuatorOutside(out_cyl_r, out_cyl_h, cyl_th, in_cyl_r + 0.3, lip_h, slot_th,
        slot_w, slot_h, slot_loc_h);
  }
  actuatorBottomSquareHole(out_cyl_r, bot_h, hole_area);
}

mActuator(stroke, in_cyl_r, cyl_th, cap_r, cap_h, lip_h, bot_h, hole_area, slot_th
    , slot_w, slot_h, slot_loc_h);
```

# B R Code

Program 12: generate.R

```
##############################
#         Functions          #
##############################

# Velocity at inlet - cm/s

v1 = function(mdot, rho, A) {
```

```
    return(mdot/(rho * A));
}

# Velocity after impeller and before nozzle - cm/s

v2 = function(mdot, rho, r2, b, n, omega) {
    Vn = (mdot * n)/(2*pi*rho*r2*b);
    Vt = omega * r2;
    return(sqrt(Vn^2 + Vt^2));
}

# Head (times gravitational acceleration) - cm^2/s^2

head = function(mdot, rho, Ns, omega) {
    return(((omega * sqrt(mdot)) / (Ns * sqrt(rho)))^(4/3));
}

# Power generated by the impeller - kg-cm^2/s^3

power_out = function(mdot, rho, A, r2, b, n, Ns, omega) {
    head_contribution = head(mdot, rho, Ns, omega);
    v1sq = (v1(mdot, rho, A))^2;
    v2sq = (v2(mdot, rho, r2, b, n, omega))^2;
    velocity_contribution = (1/2) * (v2sq - v1sq);
    return(mdot * (head_contribution + velocity_contribution));
}

# Efficiency

efficiency = function(mdot, rho, A, r2, b, n, Ns, omega) {
    return(power_out(mdot, rho, A, r2, b, n, Ns, omega) / power_in);
}

# Force generated by fluid flow - kg-cm/s^2

force = function(mdot, rho, Ai, Af, r2, b, n, omega) {
    return(mdot * (Ai / Af) * v2(mdot, rho, r2, b, n, omega));
}

##############################
#          Variables         #
##############################

##############################
#      Fluid Properties      #
##############################

# Mass flow rate - kg/s

# mdot = seq(1e-5, 1e-4, 1e-7);

mdot = 0.06309;

# Density of the fluid - kg/cm^3

rho = 1e-3;

##############################
#       Pump Properties      #
```

```
##############################

# Area of the inlet - cm^2
# Want to maximize this based on manufacturability

A = 0.2;

# Inner radius of the impeller - cm
# Not needed anymore in any formulae

r1 = 0;

# Outer radius of the impeller - cm

r2 = seq(0.1, 2, 0.01);

# Thickness of the impeller - cm
# Want to minimize this based on manufacturability

b = 0.4;

# Number of blades in the impeller
# Want to maximize this based on manufacturability

n = 4;

# Initial cross-sectional area of outlet nozzle - cm^2
# (assuming square cross-section)
# Want to maximize this based on manufacturability

Ai = b^2;

# Final cross-sectional area of outlet nozzle - cm^2
# Want to minimize this based on manufacturability

Af = 0.04;

# Specific speed conversion factor

Ns_conversion = (pi * sqrt(2271)) / (180 * (762 / 25)^(3/2));

# Specific speed - unitless
# Different ranges depending on what type of pump we want
# 500 - 2000 (imperial) is radial (more or less)

Ns = 500 * Ns_conversion;

##############################
#       Motor Properties     #
##############################

# Angular speed of the impeller - rad/s
# Want to maximize this based on motor choice

omega = 963.421747104;

# Power imparted to impeller - kg-cm^2/s^3

power_in = 6000;
```

```
##############################
#          Constants         #
##############################

extension = ".pdf";

data_extension = ".txt";

working_directory = "/home/chiraag/Documents/Caltech/SURF_2013/Plots/";

##############################
#       File Functions       #
##############################

# Create a file with the parameters in the file name

open_file = function(args) {
    name = "";
    for(i in 1:length(args)) {
        name = paste0(name,
                      (names(args))[i],
                      "_",
                      args[[(names(args))[i]]],
                      "-");
    }
    name = paste0(substr(name, 1, (nchar(name) - 1)), extension);
    pdf(file = name);
}

data_file_name = function(args) {
    name = "";
    for(i in 1:length(args)) {
        name = paste0(name,
                      (names(args))[i],
                      "_",
                      args[[(names(args))[i]]],
                      "-");
    }
    name = paste0(substr(name, 1, (nchar(name) - 1)), data_extension);
    return(name);
}

##############################
#          Main Code         #
##############################

build_loops = function(function_to_call, variable) {
    toReturn = "";
    argument_list = names(formals(function_to_call));
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != bquote(.(variable))) {
            toReturn = paste0(toReturn,
                              "for(i",
                              i,
                              " in 1:length(",
                              argument_list[i],
                              ")) { ");
        }
```

```
    }
    toReturn = paste0(toReturn, "open_file(list(");
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != variable) {
            toReturn = paste0(toReturn,
                              "\"",
                              argument_list[i],
                              "\" = ",
                              argument_list[i],
                              "[i",
                              i,
                              "],");
        }
    }
    toReturn = paste0(substr(toReturn, 1, (nchar(toReturn) - 1)), "));");
    toReturn = paste0(toReturn, " plot(");
    toReturn = paste0(toReturn, bquote(.(variable)));
    toReturn = paste0(toReturn, ", ", bquote(.(function_to_call)), "(");
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != variable) {
            toReturn = paste0(toReturn,
                              argument_list[i],
                              " = ",
                              argument_list[i],
                              "[i",
                              i,
                              "], ");
        }
    }
    toReturn = paste0(toReturn,
                      bquote(.(variable)),
                      " = ",
                      bquote(.(variable)));
    toReturn = paste0(toReturn, "), type=\"l\", xlab = \"",
                      variable,
                      "\", ylab = \"",
                      bquote(.(function_to_call)),
                      "\"); dev.off(); ");
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != variable) {
            toReturn = paste0(toReturn, "} ");
        }
    }
    return(toReturn);
}

build_data = function(function_to_call, variable) {
    toReturn = "";
    argument_list = names(formals(function_to_call));
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != bquote(.(variable))) {
            toReturn = paste0(toReturn,
                              "for(i",
                              i,
                              " in 1:length(",
                              argument_list[i],
                              ")) { ");
        }
    }
```

```
    toReturn = paste0(toReturn, "write.table(file = data_file_name(list(");
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != variable) {
            toReturn = paste0(toReturn,
                              "\"",
                              argument_list[i],
                              "\" = ",
                              argument_list[i],
                              "[i",
                              i,
                              "],");
        }
    }
    toReturn = paste0(substr(toReturn, 1, (nchar(toReturn) - 1)), ")), ");
    toReturn = paste0(toReturn, " matrix(c(");
    toReturn = paste0(toReturn, bquote(.(variable)));
    toReturn = paste0(toReturn, ", ", bquote(.(function_to_call)), "(");
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != variable) {
            toReturn = paste0(toReturn,
                              argument_list[i],
                              " = ",
                              argument_list[i],
                              "[i",
                              i,
                              "], ")
        }
    }
    toReturn = paste0(toReturn,
                      bquote(.(variable)),
                      " = ",
                      bquote(.(variable)));
    toReturn = paste0(toReturn,
                      ")), ncol = 2, byrow = FALSE)",
                      ", sep = \"\t\", row.names = FALSE, col.names = c(\"",
                      bquote(.(variable)),
                      "\", \"",
                      bquote(.(function_to_call)),
                      "\")); ");
    for(i in 1:length(argument_list)) {
        if(argument_list[i] != variable) {
            toReturn = paste0(toReturn, "} ");
        }
    }
    return(toReturn);
}

generate_plots = function(evaluating) {
    function_to_call = tolower(substr(evaluating, 3, nchar(evaluating)));
    argument_list = names(formals(function_to_call));
    for(i in 1:length(argument_list)) {
        loops = build_loops(paste(bquote(.(function_to_call))),
                            paste(bquote(.(argument_list[i]))));
        eval(parse(text=loops));
    }
}

generate_plots_variable = function(evaluating, variable) {
    function_to_call = tolower(substr(evaluating, 3, nchar(evaluating)));
```

```
    loops = build_loops(paste(bquote(.(function_to_call))),
                        paste(bquote(.(variable))));
    eval(parse(text=loops));
}

generate_data = function(evaluating, variable) {
    function_to_call = tolower(substr(evaluating, 3, nchar(evaluating)));
    loops = build_data(paste(bquote(.(function_to_call))),
                       paste(bquote(.(variable))));
    eval(parse(text=loops));
}

run = function() {
    setwd(working_directory);
    dirs = list.dirs(recursive=FALSE);
    for(i in 1:length(dirs)) {
        setwd(working_directory);
        directory = dirs[i];
        setwd(directory);
        generate_plots(directory);
    }
}

run_function = function(func, variable) {
    setwd(working_directory);
    dirs = list.dirs(recursive=FALSE);
    for(i in 1:length(dirs)) {
        toCmp = tolower(substr(dirs[i], 3, nchar(dirs[i])));
        if(toCmp == paste(bquote(.(func)))) {
            setwd(dirs[i]);
            generate_plots_variable(dirs[i], paste(bquote(.(variable))));
        }
    }
}

run_function_data = function(func, variable) {
    setwd(working_directory);
    dirs = list.dirs(recursive=FALSE);
    for(i in 1:length(dirs)) {
        toCmp = tolower(substr(dirs[i], 3, nchar(dirs[i])));
        if(toCmp == paste(bquote(.(func)))) {
            setwd(dirs[i]);
            generate_data(dirs[i], paste(bquote(.(variable))));
        }
    }
}
```