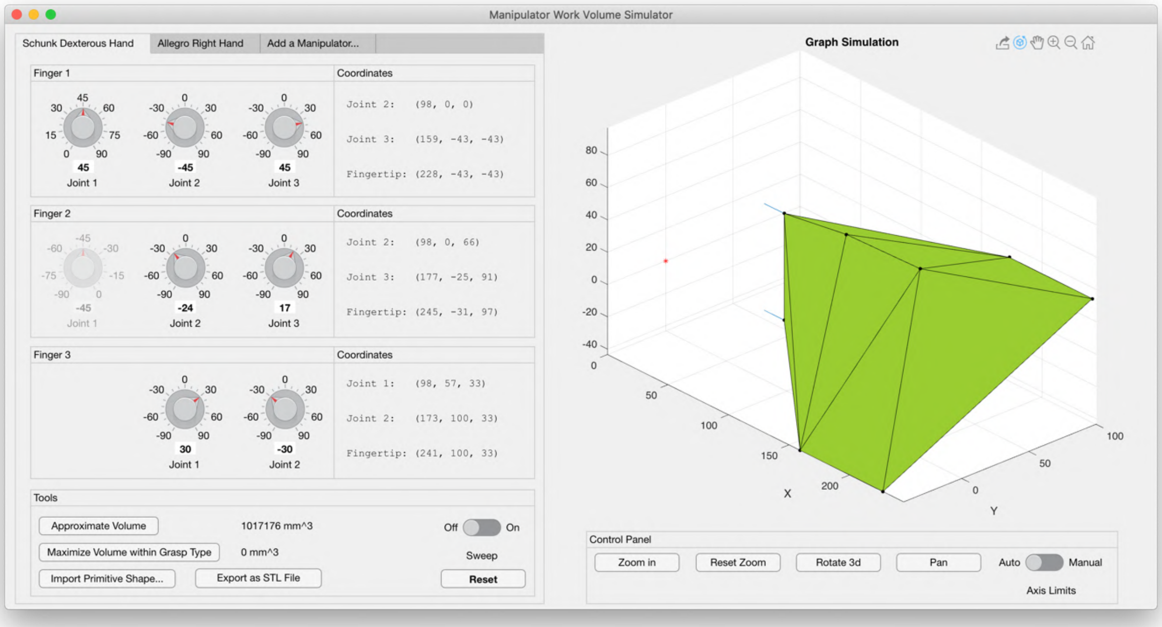
Investigating the Standardization of Work Volume for Robotic Manipulators

DOCUMENTATION **15 July 2019**

Adam Wathieu[1], Joe Falco[2]



Finger 1
- Joint 1: 45
- Joint 2: -45
- Joint 3: 45

Coordinates
```
Joint 2:    (98, 0, 0)
Joint 3:    (159, -43, -43)
Fingertip:  (228, -43, -43)
```

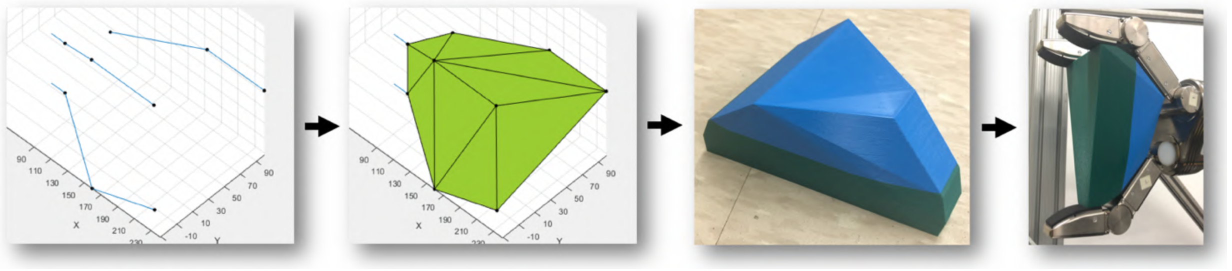
Finger 2
- Joint 1: -45
- Joint 2: -24
- Joint 3: 17

Coordinates
```
Joint 2:    (98, 0, 66)
Joint 3:    (177, -25, 91)
Fingertip:  (245, -31, 97)
```

Finger 3
- Joint 1: 30
- Joint 2: -30

Coordinates
```
Joint 1:    (98, 57, 33)
Joint 2:    (173, 100, 33)
Fingertip:  (241, 100, 33)
```

Tools
- Approximate Volume: 1017176 mm^3
- Maximize Volume within Grasp Type: 0 mm^3
- Import Primitive Shape...
- Export as STL File
- Sweep: Off / On
- Reset



[1] Georgetown University
[2] National Institute of Standards and Technology, 735.13

Table of Contents

Background and Overview 3
Work Volume Measures 4
Method 1 4
Method 2 5
Manipulator Work Volume Simulator 7
Software Documentation 8
Testing and Results, and Continuing Research 16
Showcase 17

Background and Overview

With growing robotic hand research and industrial use, there is an increasing need to capture the individual competencies and characteristics of these complex systems under a unified framework. To date, NIST has developed several robot hand performance metrics and test methods for cataloging not only the raw traits of the technology, but also task and function-level capabilities. These measures can be used to match capabilities to end-user needs as well as provide researchers and developers insight for improving their hardware and software designs. The current performance metrics can be found at:

https://www.nist.gov/el/intelligent-systems-division-73500/robotic-grasping-and-manipulation-assembly

and here:

https://www.nist.gov/el/intelligent-systems-division-73500/robotic-grasping-and-manipulation-assembly/grasping

The objective of this summer research was to add another performance metric pertaining to work volume of a hand. That is, how can work volume of a robotic hand be measured under a standard test method? Researchers developing these hands will be able to use this measure to benchmark their systems and improve the kinematic capabilities of their hands, while industries will be able to use this measure to adequately find robotic hands that fit their specific volumetric needs.

I have developed three methods of determining work volume, along with a MatLab application called the Manipulator Work Volume Simulator to investigate more about work volume. These three elements of my research will be discussed further in detail in the rest of this documentation.

I used the Schunk Dexterous Hand as the proof of concept for all my research. The Schunk Dexterous Hand is a three fingered fully actuated gripper. More information can be found here:

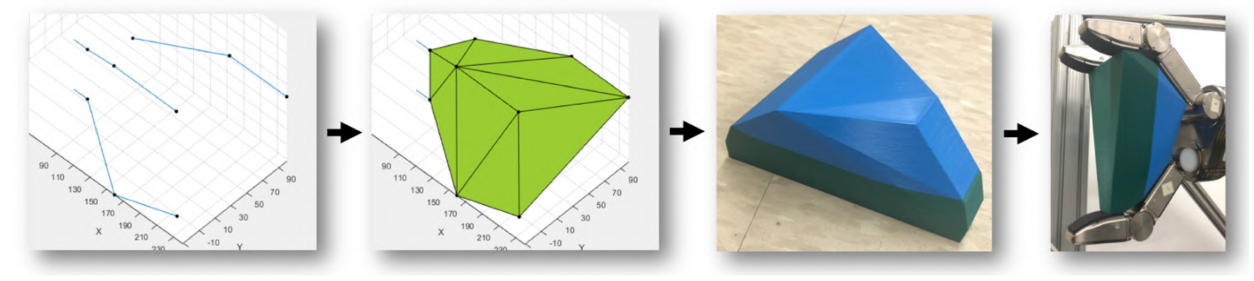
https://schunk.com/us_en/gripping-systems/series/sdh/

A copy of its documentation and its CAD files can be found in the GitHub repo of this project.

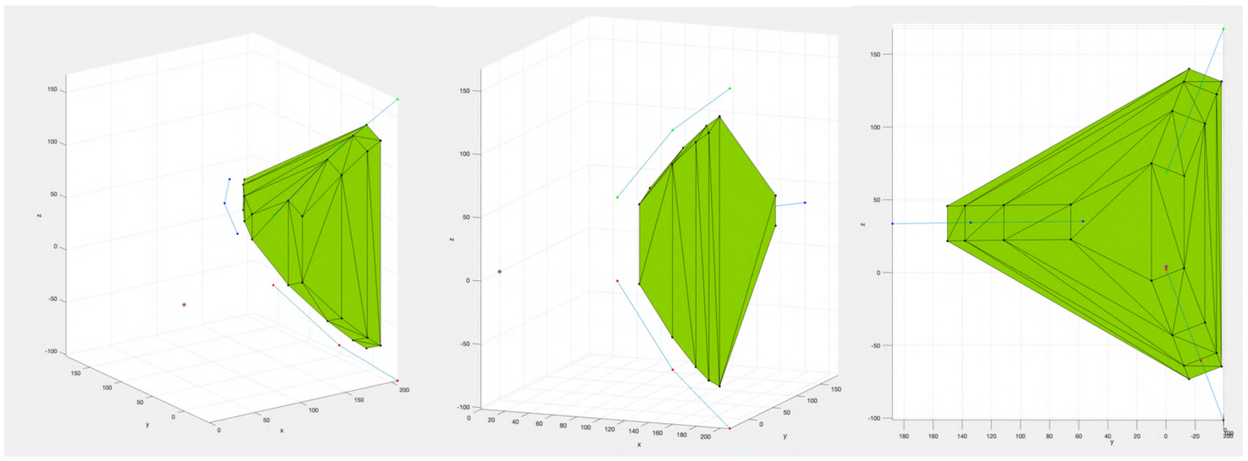
Work Volume Measures

Method 1

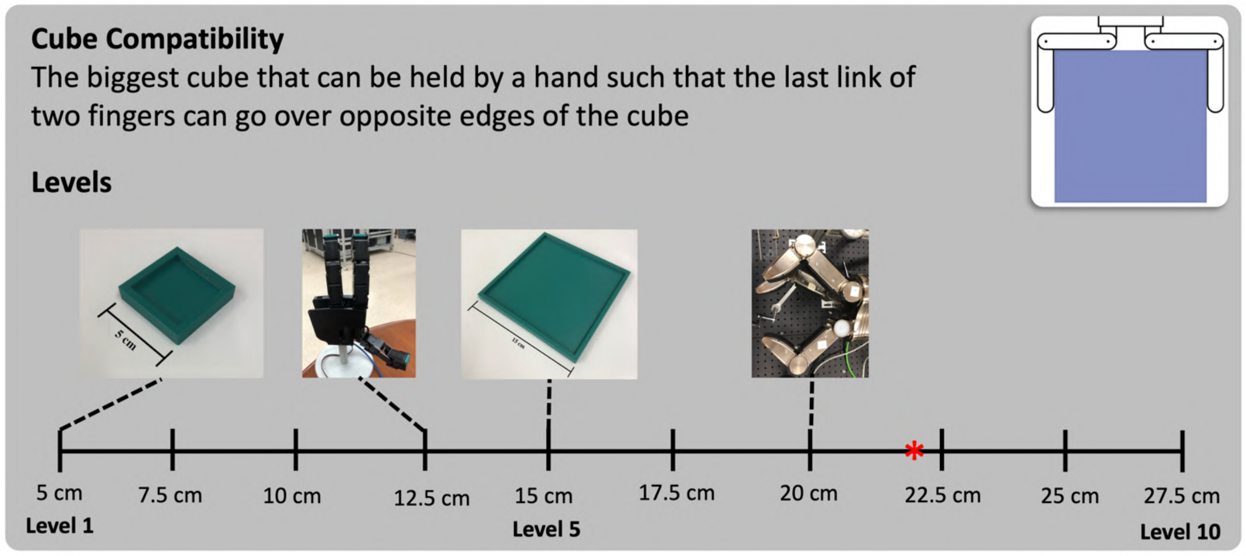
This method of determining work volume revolves around determining the largest polyhedron that can fit inside the hand, given the hand kinematics and joint limits. Finding results for this method is most easily achieved through software, namely the Manipulator Work Volume Simulator application I created.



Using this method means that robotic hands will each have only one measure of volume. After doing tests on the SDH, for example, I discovered that the largest polyhedron the SDH could hold was 54.45 inches cubed and it was when the finger1 and finger 2 are 137.51 degrees away from each other at their first joint (More about the frame of reference I use for the SDH in the Software Documentation). The benefits of using Method 1 are that it is software based, so it can easily be distributed and used at no cost, and it is precise. The drawbacks are that the method calculates for

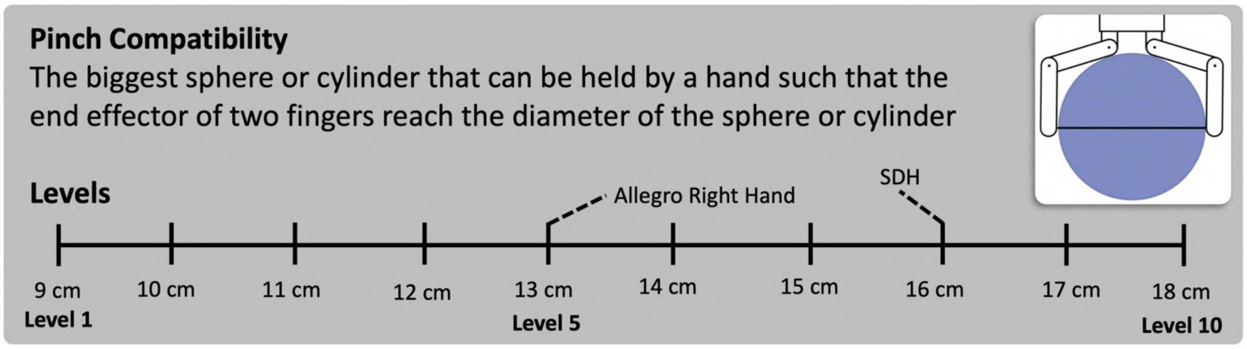


Method 2
After doing a literature search on grasp taxonomy, I learned that many grasp types can be classified by certain restrictions in the joint angles of the hands. Accordingly, this literature search led me on to working on Method 2. This method of determining work volume revolves around grabbing primitive shaped objects. Some advantages to this method are that it is more intuitive than Method 1, and it is directly applicable to industry. Some drawbacks are that in order to test for work volume, one must have these physical primitive objects at hand (made easier by my primitive shaped object STL Files on my github repo that can be printed and used for tests). I have developed three scales that can be used to determine Method 2 Work Volume:


Cube Compatibility
The biggest cube that can be held by a hand such that the last link of two fingers can go over opposite edges of the cube

Levels

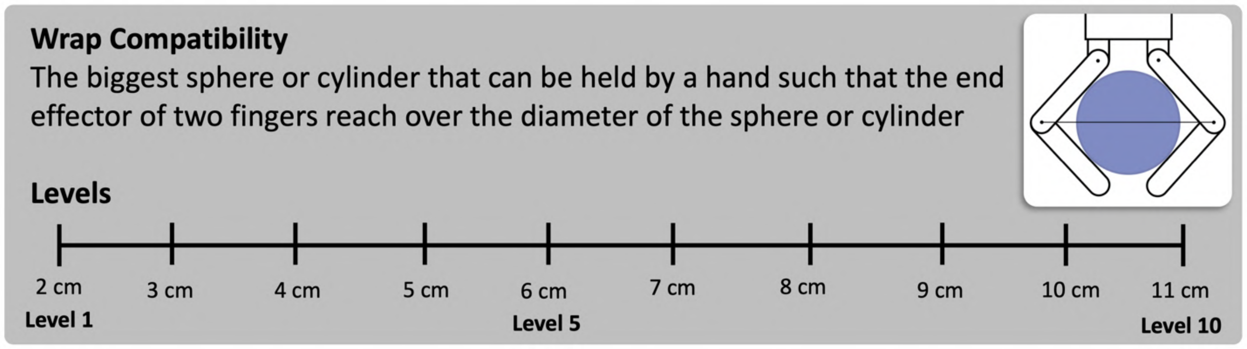
The first scale is called Cube Compatibility. As seen above, it is the level of the biggest cube that can be held by a hand such that the last link of two fingers can go over opposite edges of the cube. For example, the SDH would be said to be a "Level 7 Cube Compatible Hand." Introducing this Method brings about both a benefit and a drawback. The benefit is that it is directly applicable to industry. If a hospital is looking for a robotic hand that can hold a certain tool that can, let's say, fit inside a 15 cm cube, the hospital would accordingly look for a "Level 5 Cube Compatible Hand" to match their needs. A drawback is that this method loses precision. For example, the SDH can *actually* hold a 22.1 cm cube. However, it had to be pushed down to a Level 7 cube compatible Hand. Having fewer levels ensures that it can be easier for robotic hand researchers and people in industry to use this scale, albeit at the loss of some precision.



Pinch Compatibility

The biggest sphere or cylinder that can be held by a hand such that the end effector of two fingers reach the diameter of the sphere or cylinder

This next scale is called Pinch Compatibility. As seen above, it is the level of the biggest sphere or cylinder that can be held by a hand such that the end effector of two fingers reach the diameter of the sphere or cylinder. In my tests, I determined that the SDH is a Level 8 Pinch Compatible Hand, and the Allegro Right Hand is a Level 5 Pinch Compatible Hand.

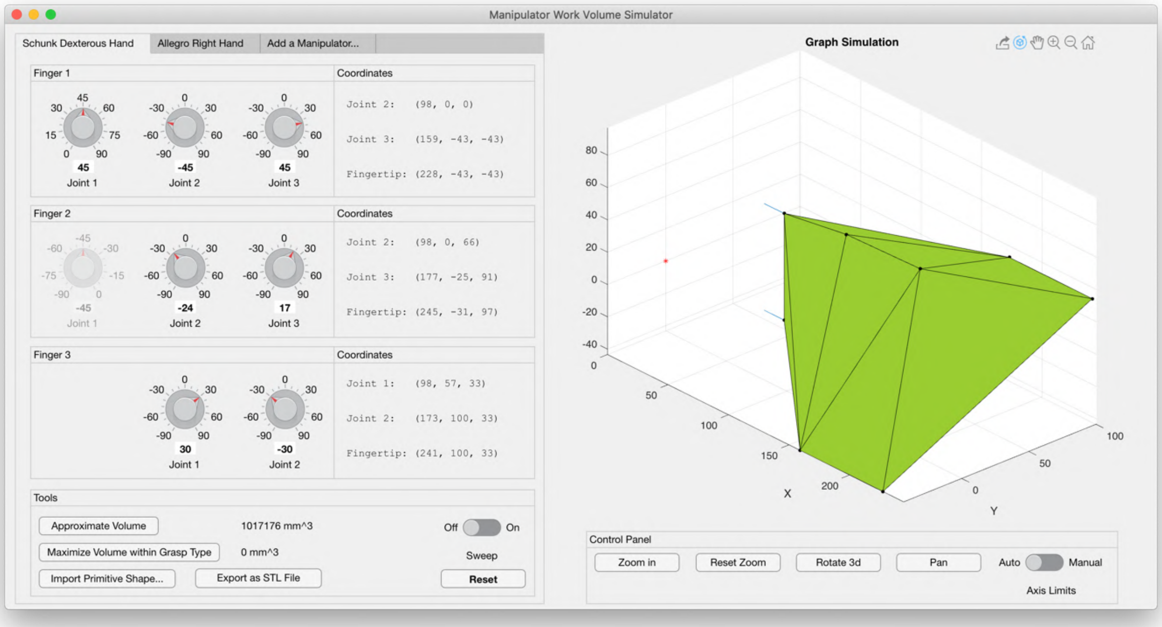


Wrap Compatibility

The biggest sphere or cylinder that can be held by a hand such that the end effector of two fingers reach over the diameter of the sphere or cylinder

The last scale that I developed is called Wrap Compatibility. It is the level of the biggest sphere or cylinder that can be held by a hand such that the end effector of two fingers reach over the diameter of the sphere or cylinder.

Manipulator Work Volume Simulator



```
Finger 1
Joint 1: 45   Joint 2: -45   Joint 3: 45
Coordinates
Joint 2:    (98, 0, 0)
Joint 3:    (159, -43, -43)
Fingertip:  (228, -43, -43)

Finger 2
Joint 1: -45   Joint 2: -24   Joint 3: 17
Coordinates
Joint 2:    (98, 0, 66)
Joint 3:    (177, -25, 91)
Fingertip:  (245, -31, 97)

Finger 3
Joint 1: 30   Joint 2: -30
Coordinates
Joint 1:    (98, 57, 33)
Joint 2:    (173, 100, 33)
Fingertip:  (241, 100, 33)

Tools
Approximate Volume                  1017176 mm^3
Maximize Volume within Grasp Type   0 mm^3
```

The Manipulator Work Volume Simulator is an application I developed to help facilitate deriving some of the relevant measures of work volume. I used the SDH as a Proof of Concept as I developed much of the program. In the program, the user moves around the hand simulator by turning the knobs on the left which represent the angles of the joints. Then, the user can click "Approximate Volume," which will then display the polyhedron in green, and calculate the volume of the polyhedron in mm^3. If the user wants to test to see if this polyhedron really will fit in the hand, they can click "Export as STL File" which will create a 3D object file of the polyhedron, which can then be 3D printed to ensure that it does indeed fit in the hand. Among other functions that are still being made are "Maximize Volume Within Grasp Type," which ideally iterates through proximate joint angles and determines which hand configuration gives the largest volume within a close proximity to what the user gave through the knobs.

"Import primitive shape" is an attempt to simulate Work Volume Method 2. Rather than 3D printing all the different levels, importing a certain cube, then using the knobs to fit the hand around the cube could be a nice software solution to Method 2.

The application is in the repo.

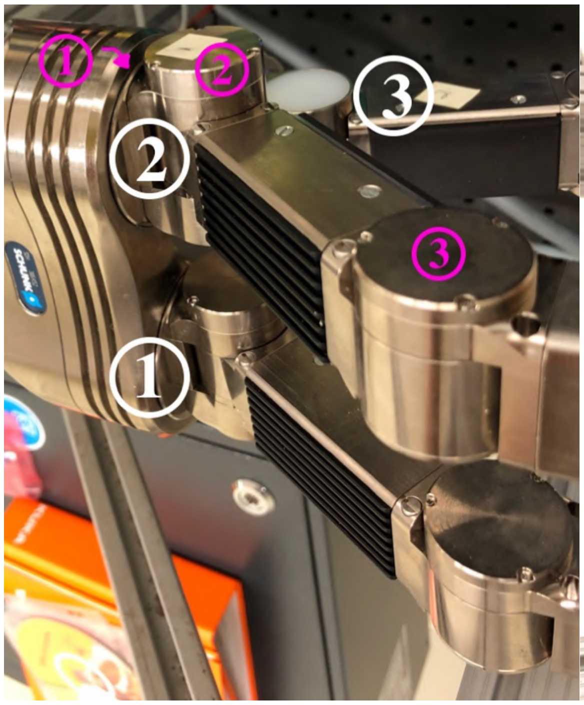
Software Documentation

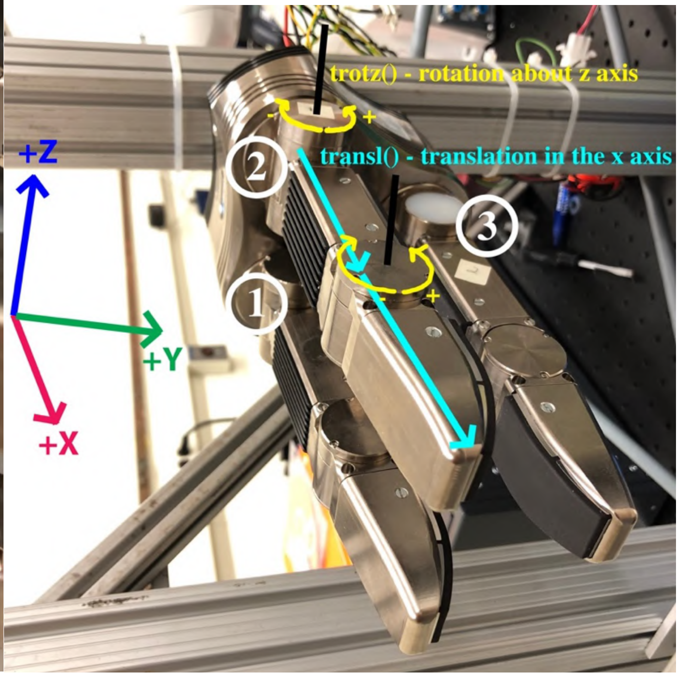
The GitHub Repository for the application and other relevant files can be found at:

https://github.com/adamwathieu-bit/manipulator-work-volume.git

I used MatLab and Peter Corke's open source Robotics Toolbox to create all my programs. The repo contains all the software that was created for this project, as well as this documentation file and some directories and files relevant to the Robotics Toolbox.

Before walking through all the files in the repo, it is important to establish a standard reference frame, which I will use in my code. Below are two pictures which describes the frame of reference used:




trotz() - rotation about z axis
transl() - translation in the x axis
+Z
+Y
+X
2
3
1
-
+

The numbers in white are the numbers of each finger. In my programs, I call the fingers of the SDH finger1, finger2, and finger3. The numbers in pink are the joint numbers, which are referred to in the program. The yellow markings describe a rotation about the z axis, and the blue markings describe a translation in the x axis. When looking through the finger class, hand class, and SDH class for the application, you will see rotation functions (trotx, troty, trotz) and translation functions (transl), which are functions from the Corke Toolbox that compute forward kinematic calculations. Put more simply,

`trotx(x)` or `trchain('Rx(x)')` refers to a rotation of x radians about the x axis,
`troty(x)` or `trchain('Ry(x)')` refers to a rotation of x radians about the y axis,
`trotz(x)` or `trchain('Rz(x)')` refers to a rotation of x radians about the z axis,

`transl(x,0,0)` or `trchain('Tx(x)')` refers to a translation of x mm in the x direction,

`transl(0,x,0)` or `trchain('Ty(x)')` refers to a translation of x mm in the y direction,
`transl(0,0,x)` or `trchain('Tz(x)')` refers to a translation of x mm in the z direction.

More information on this can be found in the Peter Corke Robotics Toolbox Documentation:

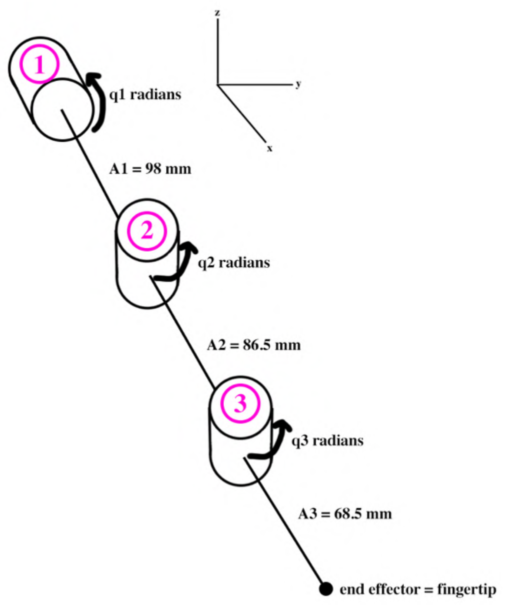
http://petercorke.com/wordpress/?ddownload=343

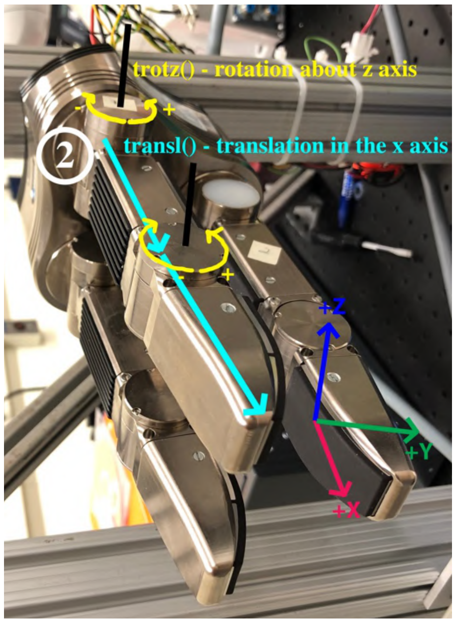
Below are descriptions of the major .m files in my github repo.

v_001.m
This was the first program written for the project. I was experimenting with the Robotics Toolbox and its forward kinematics functions, as well as investigating the kinematics of the SDH. The angles of the SDH joints are inputted by the user, and a plot of the coordinates of the fingers is given, along with a polyhedron and its respective volume. I will walk through some of the critical components of the program.

```
56 jOneMat_1 = trchain('Rx(q1)Tx(A1)', [q1]);
57 jTwoMat_1 = trchain('Rx(q1)Tx(A1)Rz(q2)Tx(A2)', [q1, q2]);
58 endEffMat_1 = trchain('Rx(q1)Tx(A1)Rz(q2)Tx(A2)Rz(q3)Tx(A3)', [q1, q2,
q3]);
```

trchain is a function found in the robotics toolbox. Refer to the Corke documentation listed above for further detail. As previously mentioned, I switched from trchain to trotx, troty, trotz, and transl for the forward kinematic calculations for the classes and the application. Essentially, however, they perform the same function. The trchain function returns a 4x4 forward kinematics matrix describing the orientation and position of the end effector of a manipulator. Two parameters are passed into the function, a string which describes the kinematics of the manipulator being analyzed (in our case, a finger of the SDH), and a matrix of the joint angles. Let us look at line 58 as an example. `Rx(q1)Tx(A1)Rz(q2)Tx(A2)Rz(q3)Tx(A3)` means that we want the toolbox to give us the forward kinematics matrix for a manipulator which, at its zero state, begins with a joint that revolves q1 radians around the x axis, then is connected with a link that is A1 mm long in the x direction, then is connected with a joint that revolves q2 radians around the z axis, which is then connected to another link that is A2 mm long in the x direction. The diagram below shows this state, and relates it to the SDH.





trotz() - rotation about z axis
2
transl() - translation in the x axis
+
+Z
+Y
+X

```
103 criticalPoints = [
104     jOneMat_1(1,4) jOneMat_1(2,4) jOneMat_1(3,4)
105     jTwoMat_1(1,4) jTwoMat_1(2,4) jTwoMat_1(3,4)
106     endEffMat_1(1,4) endEffMat_1(2,4) endEffMat_1(3,4)
107     jOneMat_2(1,4) jOneMat_2(2,4) jOneMat_2(3,4)+66
108     jTwoMat_2(1,4) jTwoMat_2(2,4) jTwoMat_2(3,4)+66
109     endEffMat_2(1,4) endEffMat_2(2,4) endEffMat_2(3,4)+66
110     jOneMat_3(1,4) jOneMat_3(2,4)+57.158 jOneMat_3(3,4)+33
111     jTwoMat_3(1,4) jTwoMat_3(2,4)+57.158 jTwoMat_3(3,4)+33
112     endEffMat_3(1,4) endEffMat_3(2,4)+57.158 endEffMat_3(3,4)+33
113 ]
```

criticalPoints is a 9x3 matrix describing the coordinates of the three fingers of the SDH. The first three rows are the coordinates of finger 1, the second three rows are the coordinates of finger 2, and the last three rows are the coordinates of finger 3. This matrix is later parsed into the individual fingers. A forward kinematics matrix is a 4x4 matrix with these elements:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The area in red describes the orientation of the end effector, while the area in blue are the position of the end effector. Since we are only concerned about the position of the end effector, we will be focusing on the blue area, and ignoring the red area. In the criticalPoints, we extract the position of the end effectors of all the forward kinematics matrices from the program. Let us look at the 3 rows for the first finger:

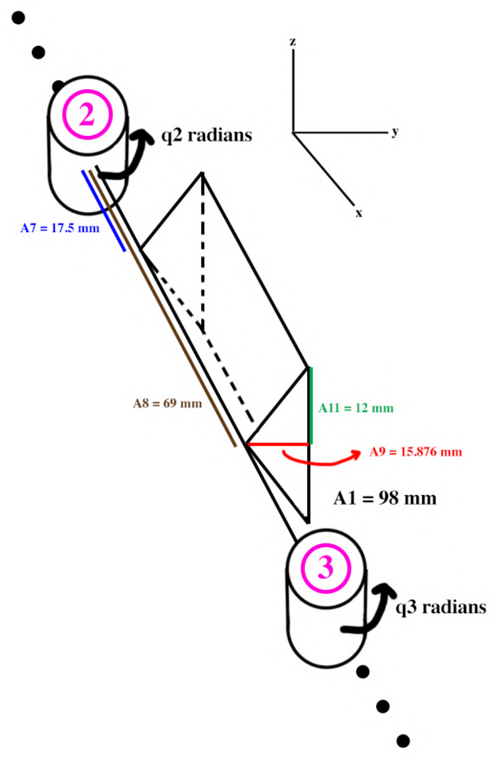
```
104     jOneMat_1(1,4) jOneMat_1(2,4) jOneMat_1(3,4)
105     jTwoMat_1(1,4) jTwoMat_1(2,4) jTwoMat_1(3,4)
106     endEffMat_1(1,4) endEffMat_1(2,4) endEffMat_1(3,4)
```

Line 104 will give us the x,y,z coordinate of the position of the first joint, line 105 will give us the coordinates of the second joint of the finger, and line 106 will give us the coordinates of the end effector, or fingertip.

v_002.m

This is second program written for the project. I added a "triangle system," where fingers of the SDH were given volume, rather than being implemented as lines. This uses more forward kinematics. This made the polyhedron volume measurements more accurate. Please look through v_001.m and its documentation prior to investigating this program. We will look at the triangle system applied to one pad of one finger.

```
jOneMat_triPrism_test = trchain('Rx(q1)Tx(A1)Rz(q2)Tx(A7)Rz(test)Tx(A9)
Rx(test2)Ty(A11)', [q1, q2, test, test2]);
jOneMat_triPrism_test2 = trchain('Rx(q1)Tx(A1)Rz(q2)Tx(A7)Rz(test)Tx(A9)
Rx(test3)Ty(A11)', [q1, q2, test, test3]);
jOneMat_triPrism_test3 = trchain('Rx(q1)Tx(A1)Rz(q2)Tx(A8)Rz(test)Tx(A9)
Rx(test2)Ty(A11)', [q1, q2, test, test2]);
jOneMat_triPrism_test4 = trchain('Rx(q1)Tx(A1)Rz(q2)Tx(A8)Rz(test)Tx(A9)
Rx(test3)Ty(A11)', [q1, q2, test, test3]);
```



This bit of code is creating four 4x4 forward kinematic matrices, each giving the coordinate of one of the corners of the triangular prism. These are the coordinates that are then used for the polyhedron and volume calculations, rather than the coordinates of the line. This is what makes the polyhedron volumes calculated in v_002.m more accurate than v_001.m

v_003.m
This is third program written for the project. This is my first attempt at finding the largest polyhedron that the SDH is capable of holding. This program finds the largest polyhedron by iterating through all joint angles, and then computing the computations described in v_002.m for each iteration. This program has a very long runtime, and later versions of the program (v_003_01.m, v_003_02.m) have more optimized code and faster running times.

v_003_01.m
This is fourth program written for the project. This is my second attempt at finding the largest polyhedron that the SDH is capable of holding. I dispersed where I was doing the forward kinematic calculations within the nested for loops, hence making the computer calculate the forward kinematics less often. This greatly improved the runtime. Future programs (v_003_03.m) provide a more efficient way of finding the same shape. Please look through programs written prior to this one to understand the extent of the code.

v_003_03.m
This is fifth program written for the project. This is my third attempt at finding the largest polyhedron that the SDH is capable of holding. This program finds the largest polyhedron based off of results from v_003_01.m. Based off the largest angles given by v_003_01.m, in this program I span through only those angles, and now with greater precision in order to find an even bigger volume. Since I am iterating though less angles while also increasing the precision, this program has similar running time as v_003_01.m. Please look through programs written prior to this one to understand the extent of the code.

v_004.m
This is sixth program written for the project. This program computes the largest cube that the SDH can hold, with the last links of fingers 1 and 2 wrapping around opposite edges of the cube. The cube is calculated to measure 207.248 mm in side length. A plot is outputted and the cube is visualized. An open source function found online was used to plot the cube. The joint angles are preconfigured to the angles that maximize the dimensions of the cube.

v_005.m
This is seventh program written for the project. This program attempts to compute the largest diameter of a circular object which would fit in the SDH. This program is not fully functional, and its objective could be better met with further work on the MatLab application, where a user could import a shape onto the plot, and wrap the hands around the shape to see if it fits.

Finger.m
This is the first program that incorporates classes and OOP which is used for the MatLab application. This class was created using the SDH as a proof of concept, meaning that capabilities of robotic fingers that are not apparent in the SDH fingers were not added into this finger. Further work on this class could include the use of the triangle system found in v_002.m, generalized enough so that it is easy and intuitive for a user to describe the thickness and geometry of the finger. Currently, this class only represents fingers as lines.

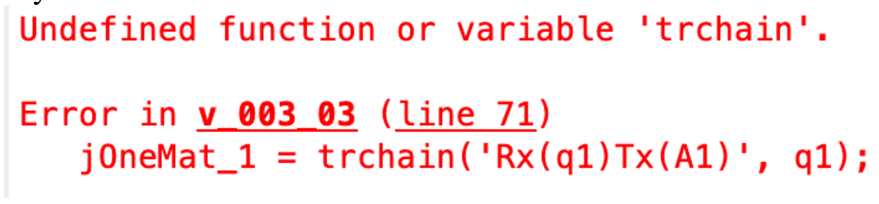
```
function criticalCoordinates = getLineCoordinates(obj, JointAngles)
   chainStream = eye(4); %4x4 identity matrix
   criticalCoordinates = [0 0 0];
   for i = 1:obj.Links %for every link in this finger add a Joint and a
  Link into the matrix computations
       switch obj.JointDirection(1,i)
           case 'x'
               chainStream = chainStream * trotx(JointAngles(1,i));
           case 'y'
               chainStream = chainStream * troty(JointAngles(1,i));
           case 'z'
               chainStream = chainStream * trotz(JointAngles(1,i));
       end
       switch obj.LinkDirection(1,i)
           case 'x'
               chainStream = chainStream *
  transl(obj.LinkLength(1,i),0,0);
           case 'y'
               chainStream = chainStream *
  transl(0,obj.LinkLength(1,i),0);
           case 'z'
               chainStream = chainStream *
  transl(0,0,obj.LinkLength(1,i));
       end
       criticalCoordinates = [
           criticalCoordinates
           chainStream(1,4) chainStream(2,4) chainStream(3,4)
       ];
   end

end
```

This is the function in Finger.m that replaces the `trchain()` commands in previous files. Since a finger is always in the sequence of a joint, a link, a joint, a link, etc., then it can be placed into a for loop. Using trot and transl rather than trchain allowed for flexibility of adding the next joint and link onto the already existing chain, hence reducing computations that are necessary by the computer. This optimization is critical because the application updates the coordinates of the hand alongside the user changing the joint limits with knobs.

Hand.m
This program uses Finger object to create a Hand object. The SDH class is a child of this class. This class is used for the application.

Common Error
If you encounter this error:



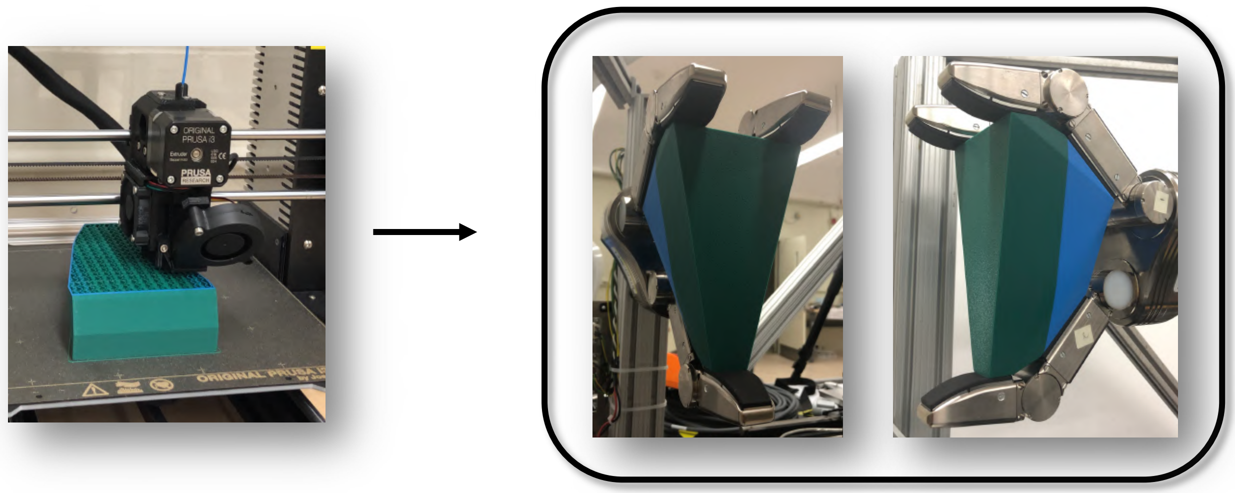
```
Undefined function or variable 'trchain'.

Error in v_003_03 (line 71)
   jOneMat_1 = trchain('Rx(q1)Tx(A1)', q1);
```

You should run the startup_rvc.m file from the Peter Corke Robotics Toolkit. This file is in the Repo.

Testing and Results

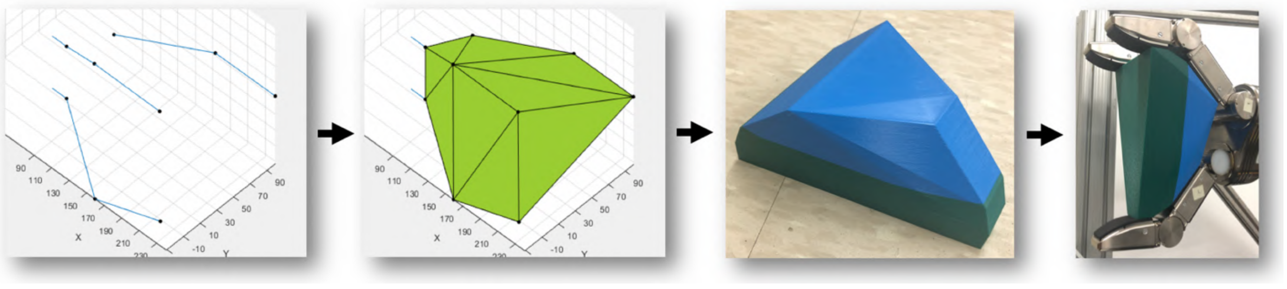
Initial testing produced expected results. To see if the Forward Kinematics Mathematics were correct in my application, I created a random hand configuration, then printed the STL file for the given polyhedron. As shown below, the polyhedron fits comfortably in the hand, leading me to conclude that my forward kinematics math were accurate.



Continuing Research

The Work Volume of a hand and its metrics and test methods continues to be refined by the project team. Some functionality in the application continues to be implemented. My research primarily revolved around the SDH, using it as a Proof of Concept for much of my work. Furthermore, future work on *minimum* work volume of a hand could be done. Some objects may be small enough such that that a large enough hand will not be able to grasp it. Working on a metric that captures this idea could be valuable.

Showcase


90
110
130
150
170
190
210
230
X
90
70
50
30
10
-10
Y