Hong Kong University of Science and Technology
Robotics Institute

Term Report

Soft Gripper Manipulation Based on UR10 Robot Arm

Abstract: This document is created as a manuscript that provides detailed setups on controlling of a Universal robotic arm. It should be able be complete an object picking and place operation with help of RGB camera.

System Environment: Ubuntu 14.04

Software Requirements: Opencv 2.4, ROS_Indigo

Authors: JIANG Chunli, TSE Yu Alexander

Date: 2017.12.12

New Linux Users should follow the tutorials here to get a basic understanding on common command line tools for Linux. (http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html)

Content

1. Mechanical Part

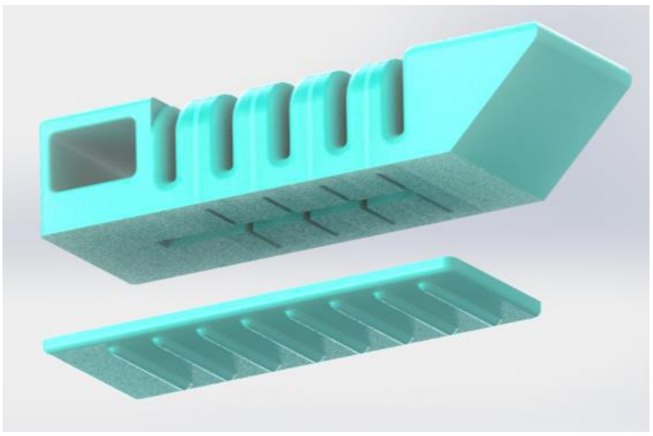
Abstract: This part will include all the mechanical design part of the project, primarily describing the design and fabrication of the soft end effector.

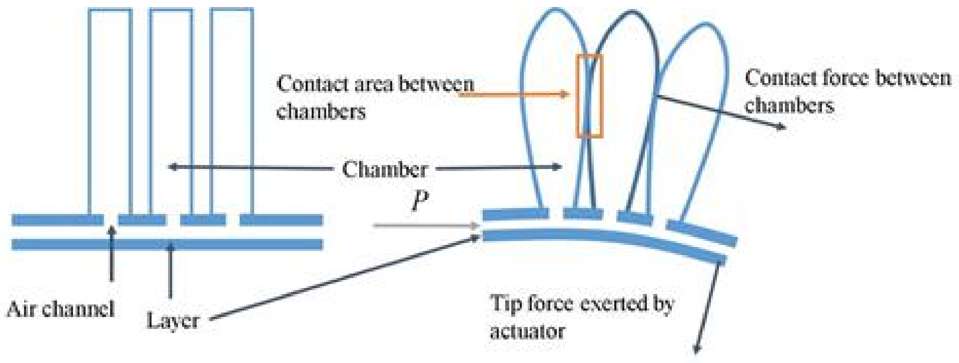
1.1 Mechanism of Soft End Effector

Conventional robots with rigid body links usually require complicated design of feedback control system to have excellent precision and high performance. For example, in order to achieve delicate motions such as grabbing an egg or other fragile objects, it is very expensive, complicated and time consuming to develop a working system. Herein, we propose a novel low cost soft end effector. It can accomplish the same goal with a simpler controller.

Soft robotics is a new area of robotics science, which is under fast development. Soft robotics is composed of elastomeric materials which can simulate the locomotion like contraction and expansion of muscles under proper actuation methods. It provides an innovative aspect for biomedical, mechanical, material, sensing, control based on multiple actuation methods including pneumatic, dialectic even combustion driven actuators. The soft end effector is expected to be a new solution of multiple robotic operations especially those which are included in large logistic activities. The end effector should also show an ability to handle fragile or soft objects that needs appropriate pressure control. Moreover, adaptability to various shapes is expected due to its intrinsic compliant nature.

The soft actuator is shown in the first figure below. The working principle of this soft actuator is by adding a pressure in the air chambers of the soft actuator, the chambers will expand and generate a force perpendicular to the finger layer, as shown in the second figure below. The parameters of the finger are designed to help improve the gripping performance. First, the fingertip has an obtuse angle, which is good because it has a larger contact area as well as it provides space to embed pressure sensors. Second, the number of chambers and the thickness of wall are also chosen based on test results, in order to increase the stability of gripping.


3D model of the soft actuator


Contact area between chambers
Contact force between chambers
Chamber
P
Air channel
Layer
Tip force exerted by actuator

Mechanism of the soft end effector

1.2 Conventional Fabrication

The conventional method of fabricating a soft actuator manually is extremely complex and time consuming. The casting mold used is manufactured by 3D printing. There are two parts of the mold: one is the air vessel and the other is bottom layer and we need to cast them separately. The first step is mixing silicone rubber with other materials to a homogenous state, after that we need to degas it and pour into the mold. Then, degas it again and finally put into oven at 50 c for 30 minutes. We don't recommend the method here though it achieves the best performance. However, anyone who is interested in the **details can refer to the Appendix.**

1.3 Current Fabrication Method

In this project, we make use of hybrid polyester material thermoplastic polyurethane(TPU) that can be **3D printed directly**. In such way, the soft end effector can be integrally formed, providing a larger strength. The downside is that the material is not as adaptive as silicone rubber. However, it is still good enough to accomplish the task in this project.

There are several points worth mention: 1. The density of object should set as 100% to prevent air-leakage. 2. Printing speed should be around 10 mm/s and the layer thickness should be around 0.1mm to balance overall printing time and quality.

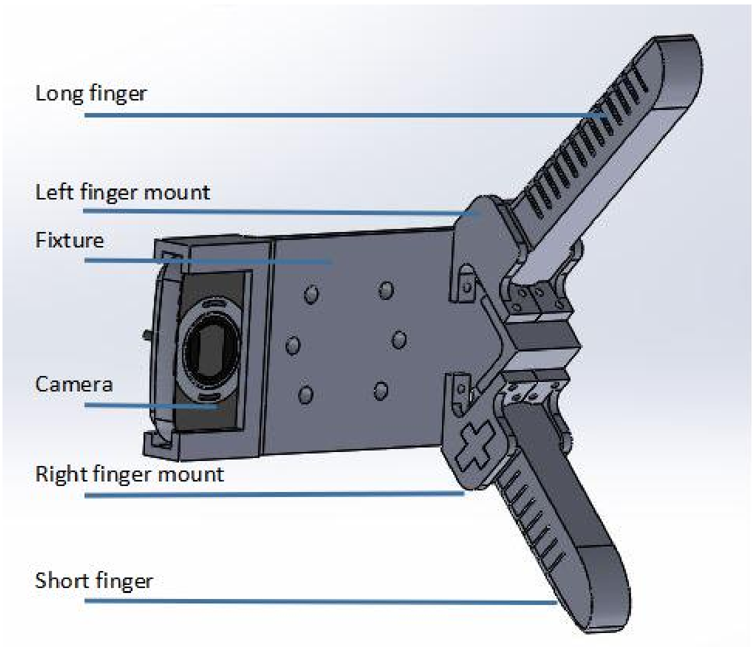
The material is available at:
(https://item.taobao.com/item.htm?spm=a230r.1.14.40.58a4b851XPrdMK&id=542621155585&ns=1&abbucket=8#detail)

1.4 Gripper Design

To try out the soft gripper on UR!0, we use 3D printing to fabricate the parts.The assembly is shown is shown in below. As the figure shows, there

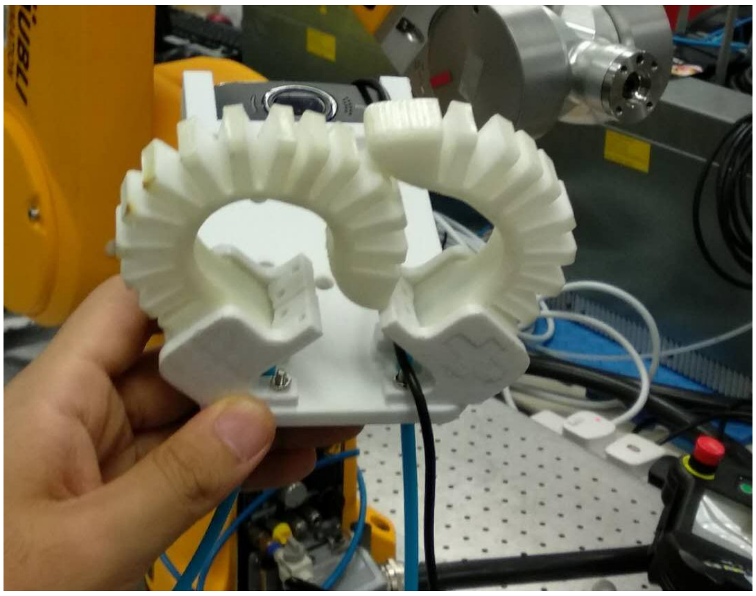
Assembly of soft end effector

are 6 majors parts. Fixture connects UR10 , the fingers as well as a holder



for camera to ensure the vision is clear and not blocked out by the fingers.

The finger lengths are not equal in this design to explore the advantage of grasp and release mechanism of non-symmetric structure. The printed-out and assembled gripper is tested out without obvious air-leakage as shown in Figure below.



Assembled gripper actuated by 4 Bar pressurized air

Based on the current test, the fingertip will enclosure at 2.5 bar, which is suitable to grasp the cube. The detailed control is introduced in Hardware section which is related to Arduino, RC Low Pass Filter and SMC Electric-Pneumatic Regulator.

2. Electronic Part

Abstract: This part will include all the hardware design part of digital control system that includes a piece of Arduino board, RC low-pass circuit and Electric-Pneumatic Regulator.

2.1 Descriptions

1. Arduino has the functions of digital output and PWM output. However, it doesn't have the analog output function, which means it can't output the desired signal which varies from 0 to 5v.

2. The SMC electro-pneumatic regulator selected for soft finger is ITV0030 (http://ca01.smcworld.com/catalog/en/frl/ITV-D-E/6-6-p0893-0946-itv_en/data/6-6-p0893-0946-itv_en.pdf). It has pressure controlled output from 0.001-5.00bar, which is suitable for the soft finger deformation control. The working principle of this electro-pneumatic regulator is that: the output pressure is linearly related to the input signal's voltage. It is shown as:

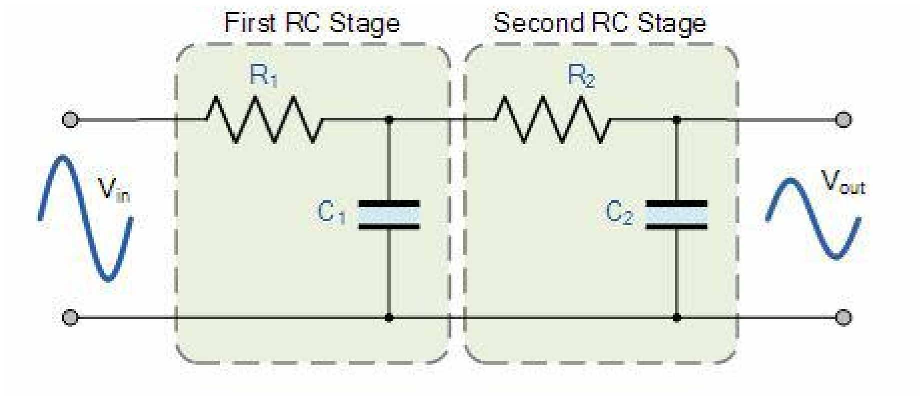
$$P=V*1 \qquad (0<V< 5)$$

in this relation. V's unit is in Volt and P's unit is in Bar.

2.2 Methodology

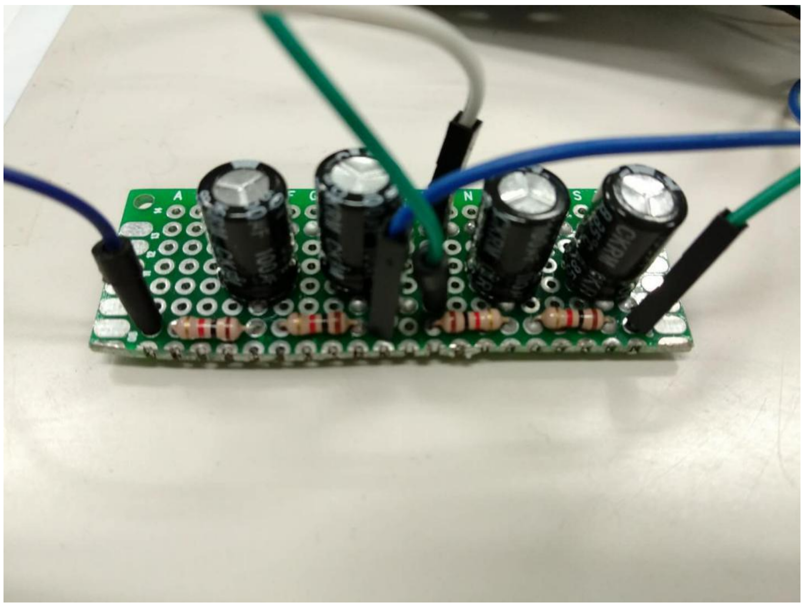
Detailed note is here for Arduino code and some knowledge background: (https://provideyourown.com/2011/analogwrite-convert-pwm-to-voltage/). Please do follow the steps from this tutorial web page.

The current solution is using second order RC low pass filter to convert the PWM signal to an analog signal. But a DAC module would have better performance but do the same job. The resistor is 1k Ohm and capacitor is 100uF in my case. In the note, it uses a first order Low Pass Filter, which is not very well performed in practical. Hence, I use a second order RC Low pass filter, which is just connecting 2 first order low pass filter together:



Second order Low pass filter

(The diagram is from http://www.electronics-tutorials.ws/filter/filter_2.html)



2 Second order RC Low Pass Filter for 2 fingers

2.3 Notes

1. The ITV0030 has 4 wires, Brown is Supply Power (24V DC), White is the input signal, Blue is GND and Black is monitor output (ignored in my case).

2. Remember to connect the GND of 24V power supply with Arduino GND so that they don't have a voltage difference.

3. Camera Calibration and Apriltag

Abstract: This part will include the vision part of the project and how to calibrate the camera.

3.1 USB _ CAM

The package is used as a driver to open the USB camera on ROS. The package can be downloaded through: https://github.com/ros-drivers/usb_cam.

3.2 Camera Calibration

We implement the monocular camera calibration package from ROS to accomplish camera calibration. Simply follow the steps on: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

3.2 Apriltag

Abstract: AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera. The AprilTag library is implemented in C with no external dependencies. It is designed to be easily included in other applications, as well as be portable to embedded devices. Real-time performance can be achieved even on cell-phone grade processors.

Methodology: AprilTags are conceptually similar to QR Codes, in that they are a type of two-dimensional bar code. However, they are designed to encode far smaller data payloads (between 4 and 12 bits), allowing them to be detected more robustly and from longer ranges. Further, they are designed for high localization accuracy— you can compute the precise 3D position of the AprilTag with respect to the camera.

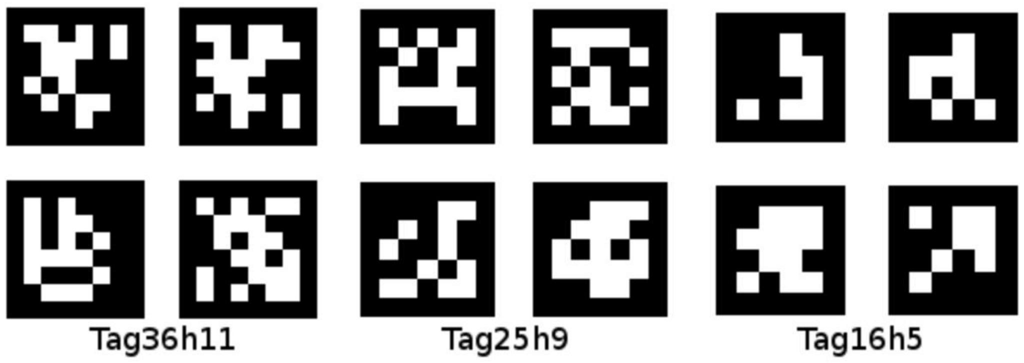
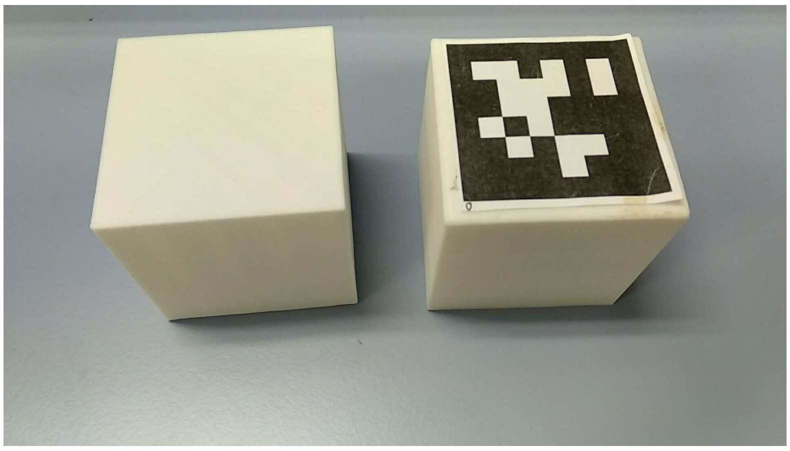


Figure above shows some sample tags which contains various ID information specifically. three individual tags are practically used in the project to represent the large dark box and two cubes (target object and reference cube) separately. We mark the target cube as ID – 0, the dark box as ID – 1 and the reference cube as ID – 2 by using Tag36h11 . Tags are printed out and stick onto the objects.



PLA cube with Apriltage ID = 0 on top

Taking two tags in the figure above as an example, the camera fixed on the gripper will detect and read the image. Afterwards, the algorithm will generate the gradient image and extract straight lines to testify the corners of the squares as shown in the figure below. Through sampled quads, the images and corresponding tag information will be matched.

We then acquire the focal length of length of the calibrated camera and the diameters of the tags to compute the 3D coordinates of the tags with respect the camera frame. Furthermore, since the tag itself contains 2D information, the rotational coordinates of the tag can also be computed. Therefore, the distance between camera (x, y, z) and angle relations (Rx, Ry, Rz) respect to camera frame.

4. ROS_Arduino

Abstract: We implement Rosserial as a bridge between Arduino and ROS. In such method, Arduino becomes a node of ROS thus easier to manipulate. Detailed information can be retrieved from: http://wiki.ros.org/rosserial_arduino

4.1 Arduino Code

```
#include <ros.h>
  #include <ArduinoHardware.h>
  #include <std_msgs/Empty.h>

 ros::NodeHandle nh;
  int sigL = 5;      //Long Gripper
  int sigR = 6;      //Short Gripper
  int i=1;

  void messageCb( const std_msgs::Empty& toggle_msg){
   digitalWrite(13, HIGH-digitalRead(13));
   // blink the led
   analogWrite(sigL, 100*i);  // set the pressure of finger by pwm
  analogWrite(sigR, 155*i);
   i=i%2-1;
   }

  ros::Subscriber<std_msgs::Empty> sub("soft", &messageCb );

  void setup()
  {
   pinMode(sigL, OUTPUT);
   pinMode(sigR, OUTPUT);
   pinMode(13,OUTPUT);
   nh.initNode();
   nh.subscribe(sub);
  }

  void loop()
```

```
{
 nh.spinOnce();
delay(0.1);
            }
```

Explanation and reference of the code can be found through: http://wiki.ros.org/rosserial_arduino/Tutorials/Blink

4.2 Rosserial

In ROS python, control of gripper is achieved by publish the empty std_msg in such form:

```
 Arduino_pub = rospy.Publisher('/soft', Empty, queue_size = 1)
   Arduino_pub.publish()
```

It should be noted that the serial communication should be built-up first to ensure that Arduino receive the messages. Also the command will cause a 3 seconds delay due to the Rospy's setting when sending an Empty message to Arduino.

5. Moveit with UR10

Abstract: Here in this section, three packages with be illustrated, which are UR modern driver, Moveit planner and Python command.

5.1 Universal Robot with Modern Driver

Since the project includes the manipulation of a UR10 robotic arm, we must use the package - modern UR driver. The package contains the geometrical configuration of the UR10 robot.

The setups can be found here: **http://wiki.ros.org/universal_robot**

5.2 Moveit

We implement the Moveit package directly from ROS as motion planner of the robot. Here we specifically use the open motion planning library which is an open – source motion planning library that primarily implements randomized motion planners.

The setups can be found here:
http://moveit.ros.org/documentation/planners/

5.3 Terminal Setup

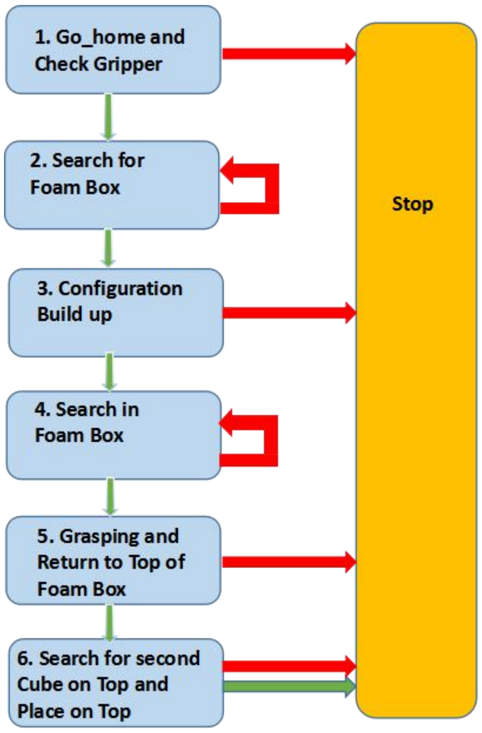
This part is essential to start the robot, where all the packages required must be opened. The terminal commands required are listed as follows:

```
#####################1.setup arduino
$ roscore
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
###  (optional )  To test out the Arduino code
$ rostopic pub soft std_msgs/Empty --once


#####################2.setup RGB camera
$ roslaunch usb_cam usb_cam-test.launch
$ ROS_NAMESPACE=usb_cam rosrun image_proc image_proc
$ roslaunch apriltags_ros example.launch
### (optional) To show up sensed AprilTag information in real time on Terminal
$ rostopic echo /tag_detections
```

```
#######################3.setup UR10
####check connection: ping 192.168.1.10
$ roslaunch ur_modern_driver ur10_bringup.launch robot_ip:=192.168.1.10
[reverse_port:=REVERSE_PORT]
$ roslaunch ur10_moveit_config ur10_moveit_planning_execution.launch limited:=true
$ roslaunch ur10_moveit_config moveit_rviz.launch config:=true
$ rosrun ur_moveit_myplan ur10_myplan_move_Zach.py
```

5.4 Python Code

Code Logic diagram


1. Go_home and Check Gripper
2. Search for Foam Box
3. Configuration Build up
4. Search in Foam Box
5. Grasping and Return to Top of Foam Box
6. Search for second Cube on Top and Place on Top
Stop

Step 1 - Go-Home and Check Gripper : The robotic arms reaches its home position to start the entire operation and the robot will check the status of its gripper once it reaches the home position and make sure it's ready for the launch of the operation. The **python code** for this part is explained **here**:

```
def go_to_home():
  ## define home position
  group.clear_pose_targets()

  ## Get the current set of joint values for the group
  group_variable_values =    group.get_current_joint_values()
  print "============ Joint values: ", group_variable_values
  ## Now, let's modify joints home position
  ## Make the gripper vertical to the plane, theta1+theta2+theta3 = -
pi/2
  ## Following two sets of values are for left and right arm
respectively
  group_variable_values[0] = 0
  group_variable_values[1] = -pi*80/180
  group_variable_values[2] = pi*55/180
  group_variable_values[3] = -pi/2-
(group_variable_values[1]+group_variable_values[2])
  group_variable_values[4] = -pi*1/2
  group_variable_values[5] = pi*1/2
  group.set_joint_value_target(group_variable_values)
  plan = group.plan()
  print "============ Waiting while RVIZ displays plan2..."
  rospy.sleep(1)
  scaled_traj2 = scale_trajectory_speed(plan, 0.2)
  group.execute(scaled_traj2)
  is_home = 1
  #Check Gripper
 rospy.sleep(5)
 global gripper_stat
 gripper_stat = 0
arduino_pub = rospy.Publisher('/soft', Empty, queue_size=1)
rospy.sleep(1.5)
arduino_pub.publish()
arduino_pub = rospy.Publisher('/soft', Empty, queue_size=1)
rospy.sleep(1.5)
arduino_pub.publish()
```

Step 2 – Search for Foam Box: The robot will search for the first red cube that contains Apriltag information of [Tag – ID == 0] The **python code** for this part is explained **here**:

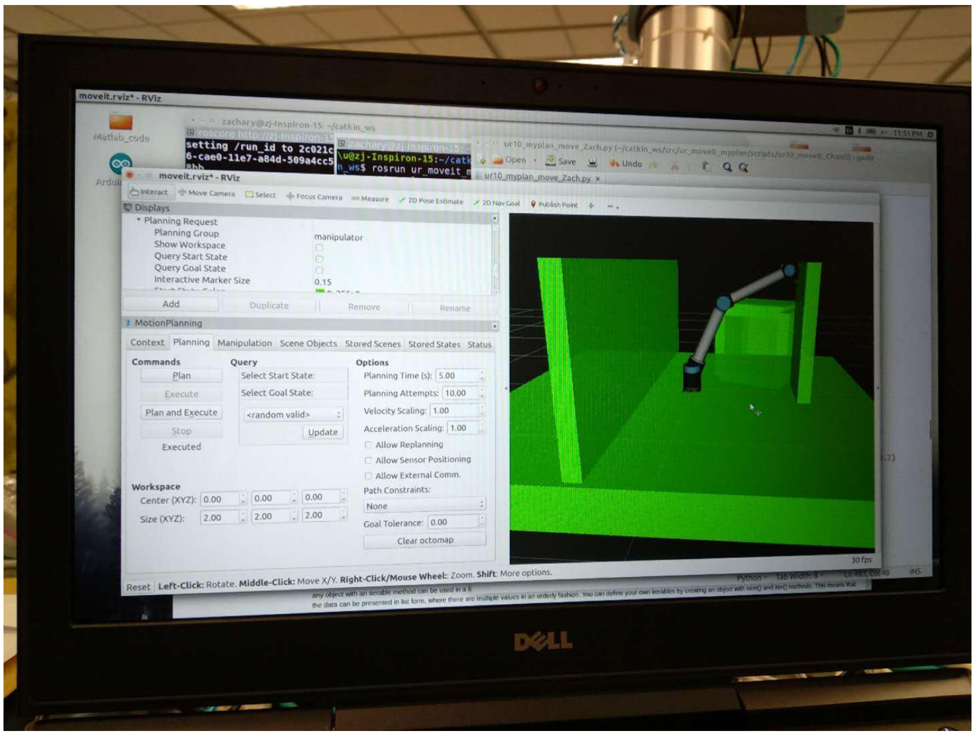
```
        elif cam_pose.detections == [] and is_collision_config == 0 :
            print "No April Tag Detected!! Detecting..."
            d_angle0 = pi*5/180
            kd_angle0=-pi*5/180

            if curr_joint0> pi*(-10)/180 and curr_joint0 < pi*75/180 and
step==1:
              print "turn left"
              rotate(d_angle0,0,0,0,0,0)
              print "11111111111111111111", curr_joint0*180/pi
            elif curr_joint0> pi*75/180 and curr_joint0 < pi*85/180 and
step==1:
              print "turn right"
              rotate(kd_angle0,0,0,0,0,0)
              step=0
              print "2222222222222222222222", curr_joint0*180/pi
            elif curr_joint0> pi*(-10)/180 and curr_joint0 < pi*75/180 and
step==0:
  print "turn right"
  rotate(kd_angle0,0,0,0,0,0)
  print "xxxxxxxxxxxxxxxxxxxxxxxxxxx", curr_joint0*180/pi
elif curr_joint0<pi*(-10)/180 :
  rotate(d_angle0,0,0,0,0,0)
  step=1
  print "xxxxxxxxxxxxxxxxxxxxxxxxxxx", curr_joint0*180/pi
elif curr_joint0> pi*81/180:
  rotate(kd_angle0,0,0,0,0,0)
  step=0
  print "xxxxxxxxxxxxxxxxxxxxxxxxxxx", curr_joint0*180/pi
```

Note：the code is very simple here since we implement a wide angle fish eye camera which can capture a lot more information. Since the workspace of the robot is minimal, the camera can detect the tag without failure.

Step 3 – Configuration Build up：A virtual geometrical configuration will be generated in Rviz to avoid collision.

Simply follow the **steps here**: http://wiki.ros.org/rviz

If successful, you will be able to see a display like:



Step 4 – Search in Foam Box：The robot will search for the obeject red cube that contains Apriltag information of [Tag – ID == 1]. The **python code** for this part is explained **here**:

```
elif cam_pose.detections == [] and is_collision_config == 0 :
    print "No April Tag Detected!! Detecting..."
    d_angle0 = pi*5/180
```

```
        kd_angle0=-pi*5/180

        if curr_joint0> pi*(-10)/180 and curr_joint0 < pi*75/180 and step==6:
          print "turn left"
          rotate(d_angle0,0,0,0,0,0)
          print "11111111111111111111", curr_joint0*180/pi
        elif curr_joint0> pi*75/180 and curr_joint0 < pi*85/180 and step==6:
          print "turn right"
          rotate(kd_angle0,0,0,0,0,0)
          step=0
          print "2222222222222222222222", curr_joint0*180/pi
        elif curr_joint0> pi*(-10)/180 and curr_joint0 < pi*75/180 and
step==0:
          print "turn right"
          rotate(kd_angle0,0,0,0,0,0)
          print "xxxxxxxxxxxxxxxxxxxxxxxxxx", curr_joint0*180/pi
        elif curr_joint0<pi*(-10)/180 :
          rotate(d_angle0,0,0,0,0,0)
          step=6
          print "xxxxxxxxxxxxxxxxxxxxxxxxxx", curr_joint0*180/pi
        elif curr_joint0> pi*81/180:
          rotate(kd_angle0,0,0,0,0,0)
          step=0
          print "xxxxxxxxxxxxxxxxxxxxxxxxxx", curr_joint0*180/pi

   elif is_arrive == 1 and gripper_stat == 1:
        print "target reached, waiting for grasping..."
        move_waypoints(0, 0, 0.2, 0.5)
        move_waypoints(-0.2, -0.2, 0, 0.5)
        move_waypoints(0, 0, -0.2, 0.4) #test
        arduino_pub.publish()
        is_arrive = 0
```

Step 5 – Object Detection and Grasping： Once the robot has detected the object red cube that contains Apriltag information of [Tag – ID == 1], the grasping operation will be initiated. The **python code** for this part is explained **here**:

```
if is_collision_config == 1 and getObject ==0 :
        curr_x = group.get_current_pose().pose.position.x
        curr_y = group.get_current_pose().pose.position.y
        curr_z = group.get_current_pose().pose.position.z
        print "1.Current height***************", curr_z
        if cam_pose.detections == [] and curr_z < 0.42:
          move_waypoints(0, 0, 0.05, 0.1)
        if cam_pose.detections != []:
          print "tag ID--------", cam_pose.detections[0].id
          qr_x = cam_pose.detections[0].pose.pose.position.x #- 0.099
          qr_y = cam_pose.detections[0].pose.pose.position.y #+ 0.058
          qr_z = cam_pose.detections[0].pose.pose.position.z
          qr_x_ori = cam_pose.detections[0].pose.pose.orientation.x
          qr_y_ori = cam_pose.detections[0].pose.pose.orientation.y
          qr_z_ori = cam_pose.detections[0].pose.pose.orientation.z
          qr_w_ori = cam_pose.detections[0].pose.pose.orientation.w
          qr_euler = quat2eular(qr_x_ori, qr_y_ori, qr_z_ori, qr_w_ori)
          qr_roll = qr_euler[0]
          qr_pitch = qr_euler[1]
          qr_yaw = qr_euler[2]

          if curr_z>0.46 and cam_pose.detections != []:
            if abs(qr_x)>0.018 or abs(qr_y)>0.018:
              move_waypoints(qr_x, -qr_y, 0, 0.2)
            move_waypoints(0, 0, -0.01, 0.2)
          elif curr_z<=0.46 and curr_z>=0.17:
            if abs(qr_x)>0.018 or abs(qr_y)>0.018:
              move_waypoints(qr_x, -qr_y, 0, 0.2)
            move_waypoints(0, 0, -0.04, 0.2)
          elif curr_z<=0.17 and curr_z >= 0.14 :
            if abs(qr_x)>0.018 or abs(qr_y)>0.018:
              move_waypoints(qr_x, -qr_y, 0, 0.2)
            else:
            move_waypoints(-0.135*sin(init_angle), -0.135*cos(init_angle), -
  0.035, 0.2)
              arduino_pub = rospy.Publisher('/soft', Empty, queue_size=1)
              arduino_pub.publish()
              rospy.sleep(4)
              getObject =1
              move_waypoints(0,0, 0.1, 0.2)
              x11=x_box-0.45*sin(init_angle)-curr_x
              y11=y_box-0.45*cos(init_angle)-curr_y
              move_waypoints(x11, y11, 0, 0.3)
              move_waypoints(0,0,0.7,0.3)
              move_waypoints(0.3*sin(init_angle), 0.3*cos(init_angle), 0, 0.3)
          elif curr_z<0.14:
            move_waypoints(0,0, 0.1, 0.2)
```

Step 6 – Search for the Second Cube and Place on Top: Once the grasping motion is successful, the robot will move its end effector to the top of the foam box and search for the other red cube which contains Apriltag information of [Tag – ID == 2] and place the object red cube on top of it.

```
    if getObject ==1 and is_collision_config == 1:
        qr_x = 0
        qr_y = 0
        qr_z = 0
        qr_x_ori = 0
        qr_y_ori = 0
        qr_z_ori = 0
        qr_w_ori = 0
        qr_euler = 0
        qr_roll = 0
        qr_pitch = 0
        qr_yaw = 0

        if cam_pose.detections[0].id == 2:
          qr_x = cam_pose.detections[0].pose.pose.position.x #- 0.099
        qr_y = cam_pose.detections[0].pose.pose.position.y #+ 0.058
        qr_z = cam_pose.detections[0].pose.pose.position.z
        qr_x_ori = cam_pose.detections[0].pose.pose.orientation.x
        qr_y_ori = cam_pose.detections[0].pose.pose.orientation.y
        qr_z_ori = cam_pose.detections[0].pose.pose.orientation.z
        qr_w_ori = cam_pose.detections[0].pose.pose.orientation.w
        qr_euler = quat2eular(qr_x_ori, qr_y_ori, qr_z_ori, qr_w_ori)
        qr_roll = qr_euler[0]
        qr_pitch = qr_euler[1]
        qr_yaw = qr_euler[2]
      if len(cam_pose.detections)==2:
        if cam_pose.detections[1].id == 2 :
          qr_x = cam_pose.detections[1].pose.pose.position.x #- 0.099
          qr_y = cam_pose.detections[1].pose.pose.position.y #+ 0.058
          qr_z = cam_pose.detections[1].pose.pose.position.z
          qr_x_ori = cam_pose.detections[1].pose.pose.orientation.x
          qr_y_ori = cam_pose.detections[1].pose.pose.orientation.y
          qr_z_ori = cam_pose.detections[1].pose.pose.orientation.z
          qr_w_ori = cam_pose.detections[1].pose.pose.orientation.w
          qr_euler = quat2eular(qr_x_ori, qr_y_ori, qr_z_ori, qr_w_ori)
          qr_roll = qr_euler[0]
          qr_pitch = qr_euler[1]
          qr_yaw = qr_euler[2]
          print "tag lololo", cam_pose.detections[1].id
      if abs(qr_x)>0.01 or abs(qr_y)>0.01:
        print "ttttttttttttttttttttttttttt", qr_x, "   ", qr_y,"   ", qr_z
        move_waypoints(qr_x, -qr_y, 0, 0.2)
      elif abs(qr_x)<=0.01 and abs(qr_y)<=0.01:
        if qr_z>0.15:
          move_waypoints(0,0, -0.02, 0.2)
        if qr_z>=0.12 and qr_z<=0.15:
          move_waypoints(-0.145*sin(init_angle), -0.145*cos(init_angle),
0.01, 0.2)
          arduino_pub = rospy.Publisher('/soft', Empty, queue_size=1)
          arduino_pub.publish()
          rospy.sleep(5)
          getObject =2
          move_waypoints(0,0, 0.2, 0.2)
          is_arrive =1
          getObject =1
        if qr_z <0.12:
          move_waypoints(0,0, 0.02, 0.2)
```

Appendix

Soft_End_Effector Fabrication - Mold Casting

1. *Overview*

   In this part, the recipe of making desired silicone (composite) and detailed procedures of mold casting will be introduced. The casting mold used here is the newly designed mold and manufactured by 3D printing. There are two parts of the mold: one is the air vessel and the other is bottom layer and we need to cast them separately. The first step is mixing silicone rubber with other materials to a homogenous state, after that we need to degas it and pour into the mold. Then, degas it again and finally put into oven at 50 c for 30 minutes.

2. *Material list:*
   1. *Shin-Etsu silicone KE-1310ST*
   2. *Curing agent CAT-1310*
   3. *Silicone oil, viscosity 50*
   4. *Glass fiber powder-1000 mesh*
   5. *pigment*

3. *Accessories:*
   1. *casting mold*
   2. *Container*
   3. *Digital electronic weight meter*
   4. *Centrifuge*
   5. *stick*

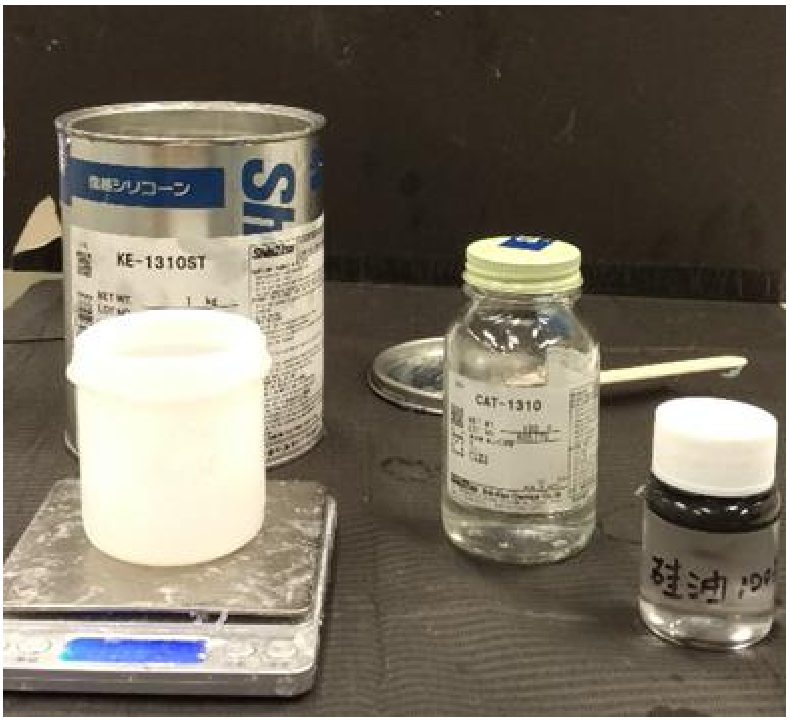


Figure. 21 experiment setups

4. *Experiment procedures:*

Pour a certain amount of silicone rubber into the container.

Approximately 50 grams of silicone rubber is needed to cast one air vessel and bottom layer. The amount of rubber here should be excessive. Around 35 grams of silicone rubber are needed for casting the air vessel part, and 10 grams are needed to cast the bottom layer.

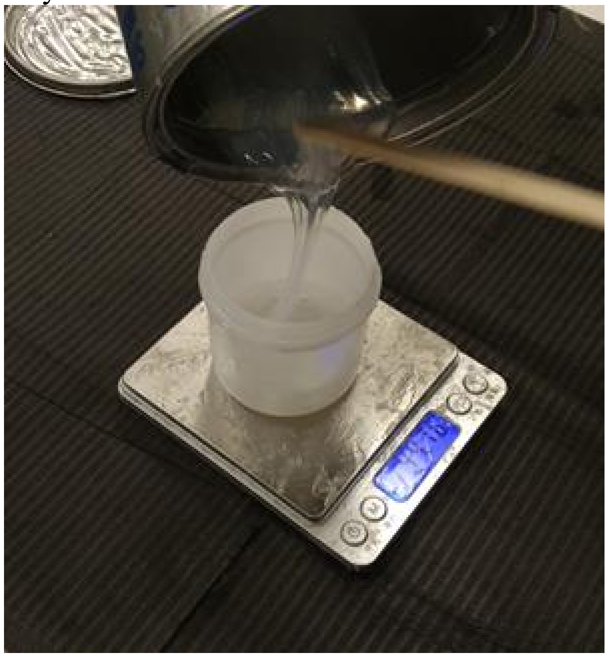


Figure. 22 silicone pouring

Add 2% silicone oil, 10% curing agent and 5% glass fiber powder, all in weight percentage, silicone oil guarantee the flow property of silicone rubber.

Curing agent is used to harden the silicone rubber by crosslinking of polymer chains, brought about by heat. The amount of curing agent could affect the curing time. Glass fiber powder served as fiber enforcement in the composite, which could change the material properties like young's modulus and tensile strength.

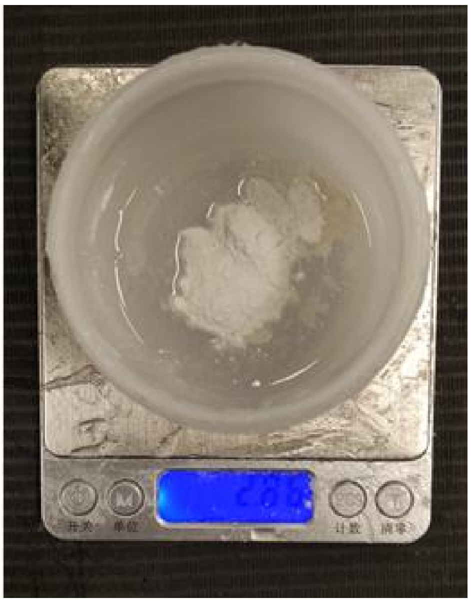


Figure. 23 adding glass fiber powder

Add proper amount of pigment into the composite and mix them by stick. The amount of pigment used here is negligible, so we could reasonably assume that the material property will not change.

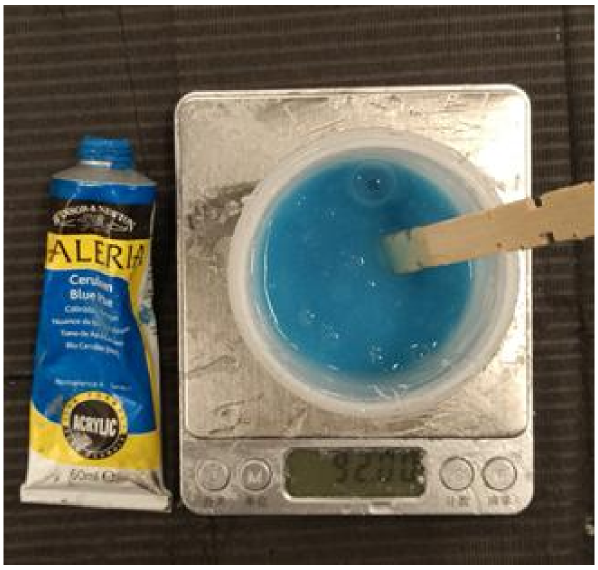


Figure. 24 adding color agent

Put the container into centrifuge. Set the program: mixing mode 40s, 2000rmp; degassing mode: 20s 1000rmp. The gross mass of container (with the composite silicone rubber we made) should be weighted on the digital electronic weight meter. The solid base for holding the container has a mass of 90g. The total mass should be calculated and used as a reference to set the centrifuge.

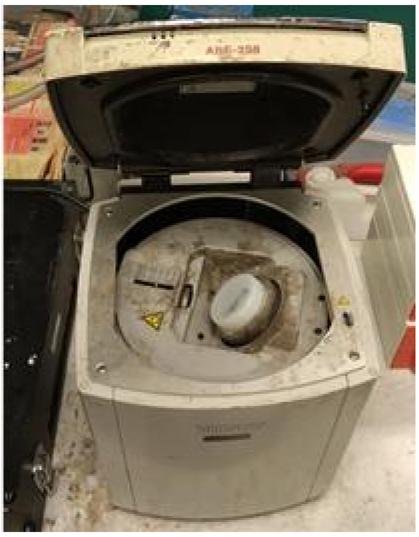


Figure. 25 centrifuge

Put the container into vacuum chamber and pump the air out to a low pressure. After the air bubbles blow out, keep the pressure for 5 minutes until there is no bubble coming out.

This step is necessary because there are still lots of air trapped in the silicone rubber after degassing with centrifuge. The porosity in material will weaken the material properties a lot.

The casting mold need to be assembled well before casting process. In our new design, each part of the mold should be inserted to slot with a certain precedence. Two solid cores should be fixed in the mold in advance. Make sure there is no gap between each part. Insert two screws into the holes at each side to align the mold in lateral direction. Spray Teflon coating to the inner surface of mold for easy demolding.

Pour the well degassed composite into casting mold. The key point here is controlling the pouring speed and let silicone rubber flows into each vessel due to self-weight. It should be noted that the pouring speed should be show enough to make sure no air is trapped in the mold.

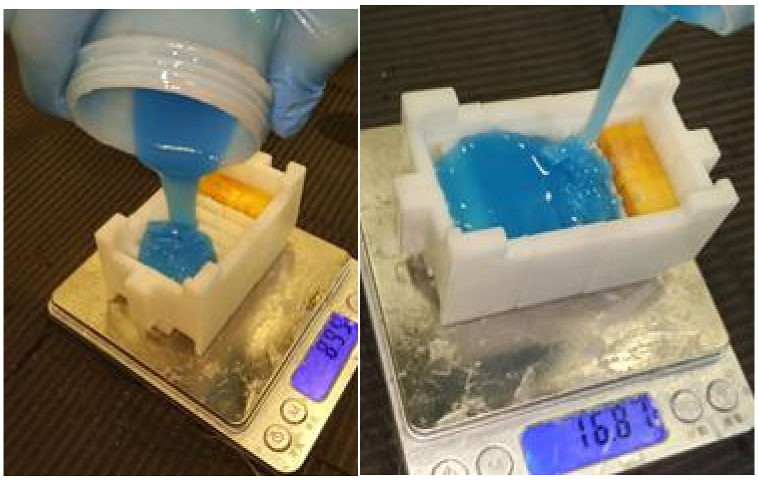


Figure. 26 filling the mold

Degas the material again with vacuum chamber. Although the silicone rubber has been degassed before casting, this step need to done because air could be trapped during the casting process. The final state should look like the picture below.

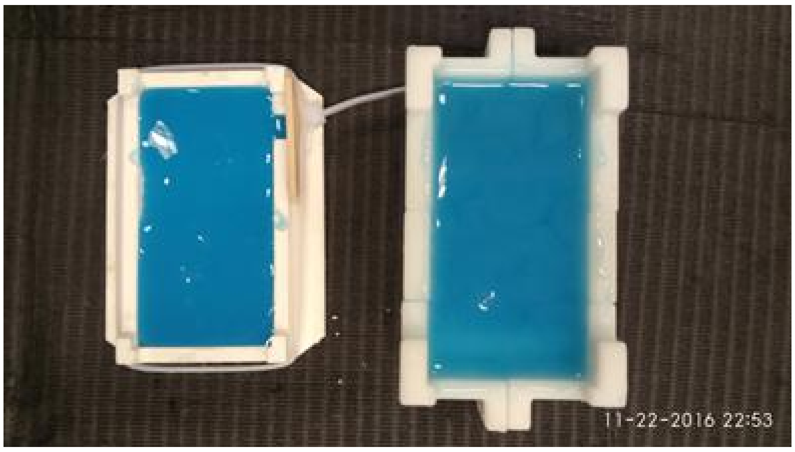


Figure. 27 filled mold

Insert the cover mold and seal the casting mold well. The excess of silicone rubber will flow out of the holes on the cover to guarantee the precise shape of the product.

A screw should be inserted into the hole and served as the core, which is the pipeline in the product.

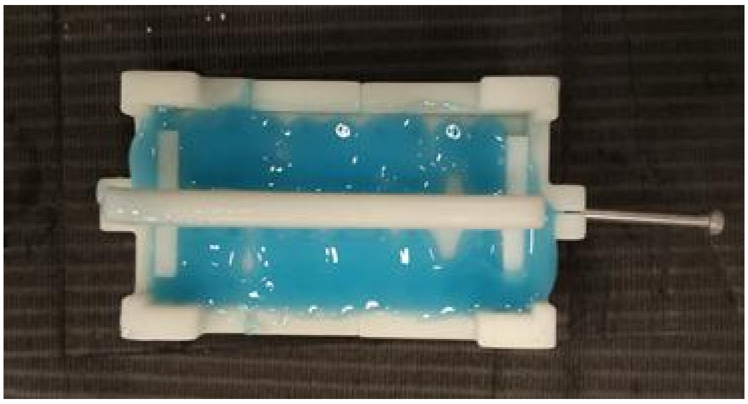


Figure. 28 screwed mold