Grasp Recognized Objects in Pybullet Environment

Ram Charit Vyas Kurra,
Dept of EECS, Wichita State University,
Wichita, Kansas
rxkurra@shockers.wichita.edu

***Abstract*—In this we train a robot with a simulations and try to do the experiment in an open environment we use different CNN models to train the data and check the results and verify them whether they are correct or not in the environment. In this the environment will be unchanged while training in the simulation but out in the real world the environment changes and then it has to adapt to it based on that instinct.**

Keywords— Robotics, Internet, Virtual Worlds, Object Recognition, Robotic Grasping

I. INTRODUCTION

In this modern world there is rising the labor charges are acute problems demanding high indoor robot services. Service robots working in the home or office need to adapt a variety of grasping that require to detect the object in the complex background environment. Due to uncertain factors like illumination, occlusion, object posture and as well as challenge of executing a real time response by indoor service robots and choosing proper gripping positions it is difficult to design a light weight recognition algorithm that can define different target objects.

Research on robotic grasping has resulted in many different grasping methods. Recently, deep learning techniques have emerged as the most preferred methods among the approaches in the field of grasp synthesis. These methods use various versions of convolutional neural networks(CNNs) to identify the objects to be grasped, which means they demand a large amount of data as well as time for training and testing. The approach also requires an expensive hardware environment. However, the results of these methods often include problems with overfitting and lack reasonable generalization ability and the avility to be well interpreted. Therefore the methods based on deep learning technology are difficult to apply to indoor robotic

In the year, 2017 Amazon has multiplied the quantity of its automated armada. Up until now, the robo-laborers are there to move bundles through the immense stockrooms, yet it is just a short time until cutting edge robots will work connected at the hip with genuine individuals, performing progressively confused errands. This shows given current condition of the innovation, the capacity for robots to comprehend their situation in nature is crucial.

II. PROBABILISTIC GRAPHICAL MODELS

Probabilistic Graphical Model are also known as statistical methods that encode joint multivariate probability distributions using graphs. It captures independence relation between interacting random variables. By knowing the graph structure once can easily learn or inference. One can even try to learn the structure given the data.

There are two types of models; they are directed graphical model and undirected graphical model. The example for directed is Bayesian Model and the undirected models are known as Hidden Markov Random Fields.

PGM present a way to model relationships between random variables. They have fallen a but due to the rise of the neural networks. There is still future since they are vey explainable and intuitive. They allow modelling and evem may be useful for learning representations of high level concepts.

A. Intelligent Robots and Systems

Current robotic applications in automotive plants are limited primarily to material handling, and spot welding operations. The Ford Advanced Manufacturing Technology Development Center (AMTD) has been pursuing force controlled robotic technology since 1996, and adaptive learning techniques for the teaching of force controlled robots since 2001[4].

Humans learn complex assembly tasks with relative ease compared to robotic systems today. Conventional robotic systems use a position based robotic controls strategy, which is ineffective as an assembly tool in cases where the assembly tolerance is less than the positional uncertainty due to the potentially large contact forces that can be generated.

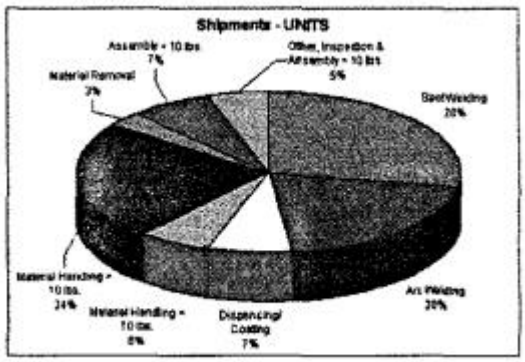


Fig. 1 Robotics Industries Association data [1]

In order to achieve assembly performance comparahle to, or exceeding humans, robots must exhibit two characteristics natural to humans. The first characteristic is a method for accommodating assembly tolerances less than the positional

uncertainty. The second characteristic is a method to learn how to use forces and torques of environmental interactions to converge on a solution that produces a successful assembly (e.g. meshing of gears, alignment and insertion of spline shafts, snapping of parts together ... etc.)

The articulated arms like Kawasaki and FANUC were more sluggish and heavily damped in their force control implementations than the MicroDexterity or Robotic Research platforms. This is a result of two things: 1) The massive inertia of the commercial robot links and the massive gear reductions in the drives used to increase the payload capacity of the robot, 2) Use of home grown force control algorithms, non-optimal force control algorithms on the commercial robot platforms unlike the well tuned used on the ParaDex manipulator and the Robots research arm[4]

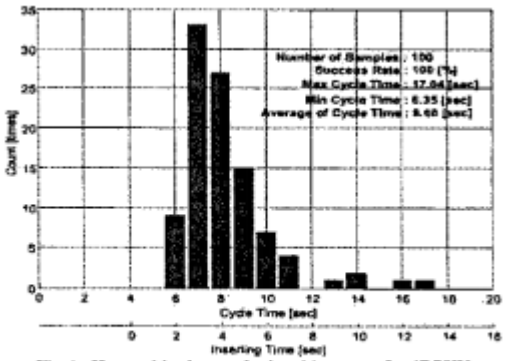


Fig 4. Kawasaki robot cycle time histogram for 4R70W forward clutch hub assembly

Previous attempts at CWRU used moment information at various points in the search field, trying to predict the location of a feature based on the moments existing at a single point. However, it was observed that in the region of the moment discontinuity the operator is drawn towards the discontinuity boundary. Upon reaching the discontinuity boundary, both the human operator and now the robot could be programmed to follow this

B. The 2016 Humanitarian Robotics and Automation Technology Challenge

In the past two decades, prior initiatives using robots and other mobile platforms aiming to increase the efficiency of the demining process have been proposed. Those initiatives proposed the use of military or commercial vehicles with mine detecting sensors or detonating tools attached to the vehicle, or the use of small, semi-autonomous mobile robots with a lightweight sensor payload linked through a rigid structure to the robot's body or attached to a scanning arm, for higher scanning flexibility and efficiency. The South African CASSPIR is a famous example of such vehicles, frequently employed in demining operations across wide territories of several African countries, including Mozambique and Angola. This vehicle has a V-shaped hull, in order to better resist mine blasts, and may carry a metal detector array or collect sampled vapors across a suspicious road for later analysis in the laboratory or by dogs. Mine resistant machines equipped with some type of soil processing and mine detonating tools, like tillers or flails, are also frequently used, in this case to detonate mines across smooth surfaces that are free from heavy vegetation and have low incline, e.g. agricultural terrains.

.

Wheeled robots with a robotic arm combine the ability to move efficiently across natural terrains with the capacity to scan hard to reach regions using their sensing arm. Examples of this class of platforms are the Gryphon-IV, built upon a 4x4 commercial platform, an ATV, with a long multiple degrees-of-freedom arm and the FSR Husky, a skid steered platform with a two degrees-of-freedom arm carrying a metal detector array[3].

Robotics challenges and competitions have for a long time been identified as a way to engage the robotics community into a subject and push forward the state-of-the-art in that subject. The idea is to structure a problem around a welldefined task that may hopefully be performed by a robot in different, but related, environments. Some examples of these challenges are the multiple varieties of the Micromouse contest whose goal is solving some kind of maze or finding a path to a given target employing simple mobile robots [14]. Or the multiple contests run by the RoboCup organization across the world, some aiming to improve robot motion control or cooperation strategies in order to optimize the performance in a given task, e.g. playing football [15]. Perhaps the most famous and demanding challenges are the multiple DARPA Challenges that have been proposed in the past ten years or so.


Fig. 1. FSR Husky robot during the competition, on the challenge field.

Conclusion is that the most positive aspects, although not initially identified as goals, were at the educational level and at developing a robust framework supporting cooperative remote field robotics trials. While it can be argued that no significant progress has been achieved, it should be noted that increasing the awareness of the robotics community for the humanitarian demining problem and the possibility of addressing it using robots with student teams has been the primary contribution of HRATC in the last three editions. This software may be employed to test and run on a simulator, algorithms for robotics demining. Additionally, the Challenge provided free access to expensive field robotics equipment to groups all around the world. During these three years a total of 38 teams from 15

countries from Americas, Europe and Asia were involved in the Challenge[3].

C. *Continuum Robots for Medical Applications: A Survey*

Robotics has impacted human life in many significant ways. Apart from revolutionizing the manufacturing sector, robots have now found their way out of the factory and into such applications as agriculture, aerospace, and education, just to name a few. Over the past decade, robots have also been integrated into operating rooms around the world and have enabled or improved many new minimally invasive surgical procedures. Minimally invasive surgery is beneficial because it can reduce patient discomfort, costs, and hospital time. The use of robotic technology in surgery brings precision, intuitive ergonomic interfaces, and the ability to access surgical sites remotely with miniaturized instrumentation. Thus, robotics has the potential to further advance the benefits of minimally invasive surgery and make new procedures possible[2].

Teleoperated surgical robots offer advanced instrument-tation and versatile motion through small incisions directly controlled by the physician . However, typical surgical robots require a large footprint in the operating room and use instruments that are rigid and straight with a functional articulating tip. Hence, while the trend toward minimally invasive surgery is evident, and robotics has enabled enhanced performance in minimally invasive abdominal surgery, adaption to areas requiring more delicate, circuitous, access is challenging, and certain procedures must still be performed using a more invasive open approach.

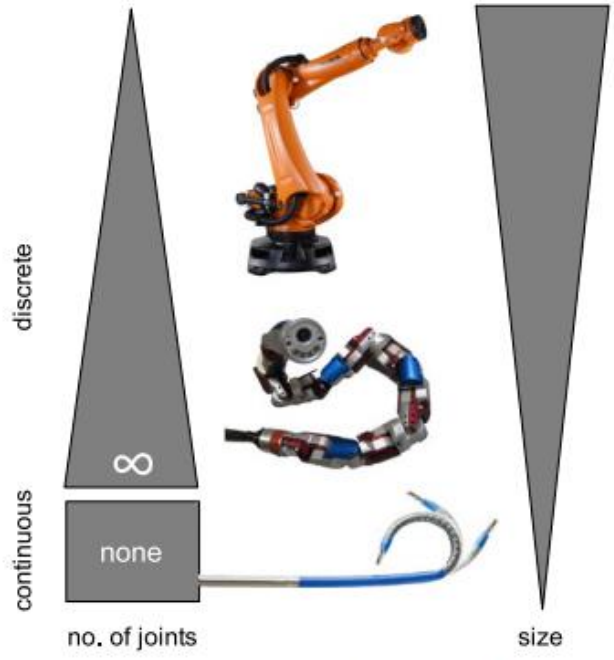
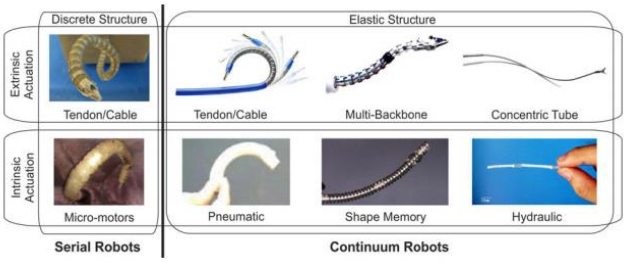


Fig. 1. Continuum robots have an infinite-DOF curvilinear elastic structure. In contrast, conventional serial kinematic chains or hyper-redundant manipulators are characterized by discrete links. (Images: top ©2015 Kuka Robotics Corp.; bottom ©2015 Hansen Medical Inc.)

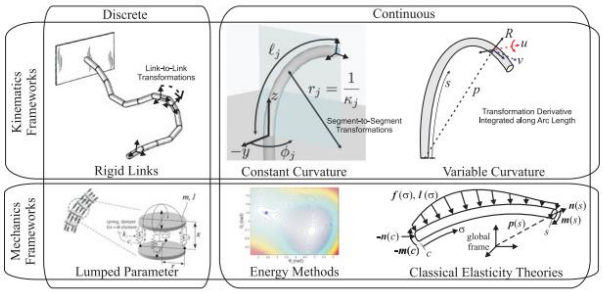
A continuum robot is an actuatable structure whose constitutive material forms curves with continuous tangent vectors. The set defined above is inclusive of definitions in prior papers and review articles, which have typically also included more specific descriptions such as "continuously bending," "infinite-DOF," "elastic structure," and "do not contain rigid links and identifiable rotational joints." Hyperredundant serial robots with many, discrete, rigid links and joints are not continuum robots because they can only approximately conform to curves with continuous tangent vectors. The boundary that separates continuum robots from other snake-like or hyper-redundant manipulators is sometimes obscured by manipulator designs that use continuously bending elastic elements along with conventional discrete joints in the same structure. While these robots may not technically be classified as continuum robots according to the conventional definitions, they could be referred to as pseudocontinuum robots or hybrid serial/continuum robots, as they are closely related to continuum robots and share many attributes with them. Continuum robots can be categorized in terms of their structural design and actuation strategy as we outline in the following sections. Fig. 2 illustrates the resulting broad categories of continuum robots and provides a reference point with examples for the discussion below[2]



Concentric-tube robots are composed of multiple, precurved, elastic tubes that are nested inside of each other. The base of each tube can be axially translated and rotated to control the shape of the robot structure. The tubes are typically made from the shape memory alloy NiTi in its superelastic phase, such that each tube can be set into a desired shape by heat treatment before being assembled concentrically. These robots could also be considered multibackbone, but the precurved backbone tubes are arranged concentrically rather than offset by spacer disks, and the ends are not fixed to each other, which allows the tubes to freely translate and rotate independently. Thus far, concentrictube robots have been constructed with smaller diameters than many other continuum robots. These robots have been the subject of much investigation over the last ten years [16]–[19]. A recent review paper describes the history and current state of the art of concentric-tube robots in particular.

The earliest continuum robot modeling approaches actually employed variable-curvature kinematic frameworks as tools to resolve redundancy and control the shape of hyper-redundant serial manipulators [19]. Recently, many efforts in continuum robotics have adopted similar approaches for robots, which do not conform accurately to a constant-curvature shape. Variable-curvature frameworks typically describe a material-

attached homogeneous reference frame comprising a position vector p(s) ∈ R3 and a rotation matrix R(s) ∈ SO(3) expressing the backbone pose as a function of arc length, s, along the robot. As depicted in Fig. 3, the rotation matrix evolves along the arc length according to the differential kinematic relationship dR ds = R(s)[u(s)]× (1) where u(s) ∈ R3 is the so-called curvature vector containing angular rates of change about the current axes of R(s), and [ ]× denotes the standard mapping from R3 to so(3) (the set of 3×3 skew-symmetric matrices).



In this section, prominent research and commercial systems that apply a continuum robot to surgical applications are summarized. We structure our discussion by surgical discipline, an overview of which is provided. In the interest of brevity, we do not discuss research systems which have a theoretical medical application that has not yet been investigated or those without a record of recent continuous publications. The research systems selected for this survey have either been evaluated in ex vivo or in vivo tissue experiments, published in medical journals, or studied in a realistic in vitro environment with close relation to the medical application. Based on these selection criteria, we have not extensively discussed several potential promising surgical applications. The interested reader can find information on continuum robots in ophthalmic surgery, single-port access surgery, arthroscopy, and colonoscopy.

While the first continuum robots were created almost 50 years ago, medical applications have clearly been a primary driving factor for continuum robot research over the last decade. Substantial progress has been made in design, modeling, control, sensing, and application to specific medical problems. We have surveyed the core principles underlying continuum robotics research in medical applications and given an overview of surgical systems that are either commercialized or relatively far advanced in terms of clinical readiness.

D. Decentralized Asynchronous Learning in Cellular Neural Networks

TWO MAJOR variations of cellular neural networks (CNNs) have been studied in the neural networks community. The CNN introduced by Chua and Yang in 1988 consists of individual units (cells) connected to each of their neighbors on a cellular structure[1]. Each cell of such a CNN is a computational unit and has been applied to pattern recognition and image processing. The CNN is a highly nonlinear system and its stability is important for real applications. Multistability of CNNs is discussed in. In, Werbos introduced a cellular implementation of simultaneous recurrent neural networks (SRNs), where each "cell" is an SRN with the same set of weights but a different set of inputs. Such a CNN consisting of SRNs as cells is called a cellular SRN (CSRN); a CNN containing multilayer perceptrons (MLPs) as cells is called a cellular MLP (CMLP). CSRNs have been used in the maze navigation problem, facial recognition, and image processing. Stability of recurrent neural networks in the presence of noise and time delays is discussed in. Thus, and together provide a basis for the stability of such CNNs containing NNs in each of its cells. In the original CNN, each cell is connected only to its adjacent cells. However, in CMLP and CSRN, the connection of different cells to each other is application dependent, as is shown in application to bus voltage prediction in a power system. However, even with variations, most of the CNNs studied so far consist of identical units in each cell of the CNN and learning is centralized and synchronous, i.e., the NNs in each cell of the CSRN or CMLP are trained simultaneously in one location.

The learning unit can have supervised, unsupervised, or reinforcement-based learning and provides a measure for the evaluation of performance based on which a cell can improve itself. Since the individuals interact with their neighbors in a collaborative learning system (where neighborhood may be defined on the basis of certain parameters which are application dependent, e.g., electrical distance in the application shown in this paper), a communication unit is also present in the cell. This unit consists of an input/output interface for sending to and receiving from other cells in the network. Communication takes place according to a predefined rule with as many neighbors as is required by the application.[1].

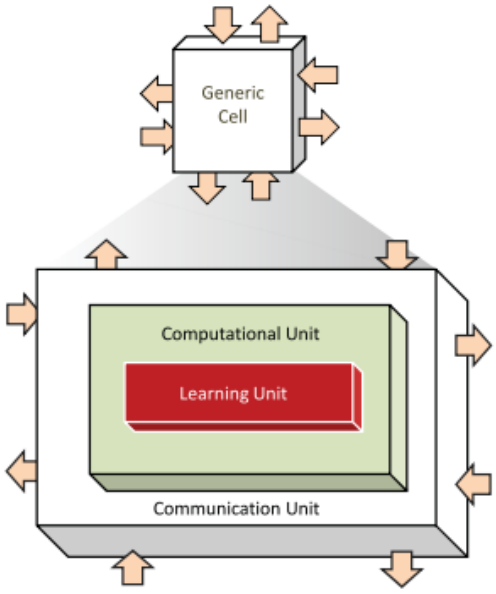


Fig. 1. Internal units of a generic cell in a CNN.

LOLS is a social behavior of swarms where each individual learns at different pace, at different times, and in different environment while still interacting with the other individuals of the society. This behavior is comparable to students performing certain projects in a virtual classroom in which distant students learn at their own pace, in their own environment at different times while still working on a common objective. For the same reason, decentralized learning has developed as an instructional paradigm in the field of distance education where both synchronous as well as asynchronous learning methods are utilized.

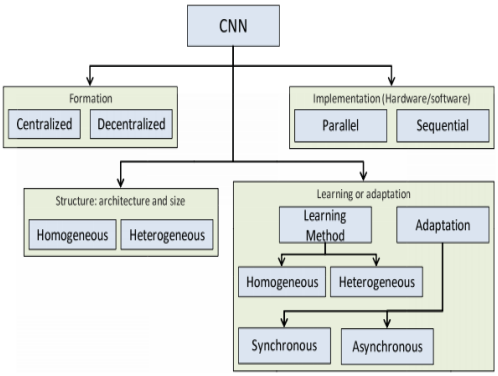
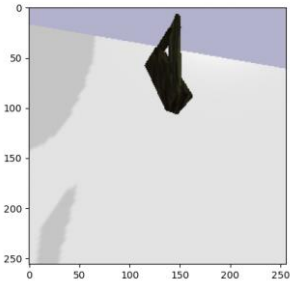


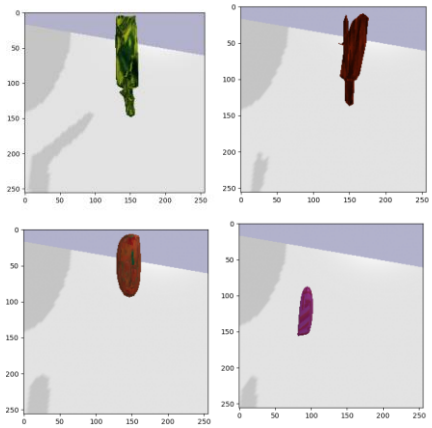
Fig. 2. CNN variants: structure and implementation.

LOLS is a challenge due to individual learning systems interacting with other members of the swarm in a collaborative environment. The objective of such learning is to improve the overall performance of the system. However, this is achieved only by improving each individual's (subsystem) outputs. Therefore, cognitive as well as social learning is required[1]. This was addressed in this paper by developing a decentralized asynchronous collaborative learning framework for CNN. In such a framework, learning takes place locally on spatially distributed cells that learn asynchronously.

III. Project Description and Results.

In this we use a python file to grab the image from the set of the images and then we save the numpy array as the image and then we use that image to train the object detection and then using that we try to pick the object placed infront of the robot. We need to find the position of the of the robot arm and then give it to the sweyer file so that it picks accordind to the co ordinates here are the few images of the robot that is trained on





Here is the code of the images that are labelled to train the model and then we use the trained model to feed the robot to pick the object up.

```
<annotation>
        <folder>TestingData</folder>
        <filename>multiimg
(1).png</filename>
        <path>C:\Users\vidya.madderla\De
sktop\Project
2\Data\TestingData\multiimg
(1).png</path>
        <source>

        <database>Unknown</database>
        </source>
        <size>
                <width>640</width>
                <height>480</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>v249b387_105</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>261</xmin>
                        <ymin>218</ymin>
                        <xmax>297</xmax>
                        <ymax>307</ymax>
                </bndbox>
        </object>
        <object>
                <name>v249b387_106</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>312</xmin>
```

```
                        <ymin>277</ymin>
                        <xmax>368</xmax>
                        <ymax>407</ymax>
                </bndbox>
        </object>
        <object>
                <name>v249b387_108</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>310</xmin>
                        <ymin>102</ymin>
                        <xmax>367</xmax>
                        <ymax>277</ymax>
                </bndbox>
        </object>
</annotation>
```

After giving them we need to get the positions of the robot arm we can do that by placing the

Thumb Position: (1.0988989838814713, 0.011248375237725327, -0.02982957840067072)
Thumb Orientation: (0.3700974103504305, 0.6818981384255702, -0.43216559524152903, 0.45964740176920565)
Index Position: (1.1126972753003213, 0.03819062673723799, -0.02304806962776716)
Index Orientation: (0.23571759477043636, 0.6522878305662545, -0.2954821179837039, 0.6569993299560314)
Mid: Position: (1.2173737114430558, -0.00789872587133551, -0.04307569060507856)
Mid: Orientation: (-0.6585641828262834, -0.18195210095565126, 0.6925911701074384, -0.23130957858546103)
Ring Position: (1.2021018185256271, 0.00010690227979461356, -0.06511072487522948)
Ring Orientation: (-0.6440914004249444, -0.22168496955179462, 0.6809147305116744, -0.26899288456326803)
Pinky Position: (1.1894623447718549, 0.001709074050430054, -0.08579113335098613)
Pinky Orientation: (-0.637799471455646, -0.2371318439480155, 0.6756904502350024, -0.2835890305670028)
Palm Position: (1.1086097032742372, -0.047049561418915344, -0.08678931519059442)
Palm Orientation: (-0.043833708106287304, 0.715976595022727, -0.012907801231346204, 0.6966272389741064)

IV. Conclusion

We were able to get the result on Pybullet. The robot was picking up the object and everything was working fine. Therefore, we were able to modify the source code and make the robot arms work according to our project requirement. This was a great example on how the robot actually works. While we studied all the theories behind it, it was an outstanding feeling to see it actually work and implement mechanics such as inverse kinetics on it.In this paper we finally conclude that any thing can move accordingly in the environment if they have proper data and also given a closed environment.
Average precision is 65.8 and average recall is 69.4

V. Future Work

In future there might be different situation if the robot is left in an open space it might not get chance of second time to pick . So we have to work with pre-knowledge that is, if there is a wall in front it has to identify a the first glance and try to avoid it and proceed forward. This might be a bit difficult because we have feed each and every object to robot in prior. This might actually occupy more memory but this will be efficient so that it might not make any mistakes.

VI. APPENDIX

Get_data.py

```
import pybullet as p
import time
import math
import random
import numpy as np
import pybullet_data
import random
import glob
import os
import matplotlib.pyplot as plt
from datetime import datetime


##################################################
#########                Simulation                Setup
##################################################
########################


clid = p.connect(p.GUI)
if (clid < 0):
    p.connect(p.GUI)
```

```
    # p.setGravity(0,0,-9.8
    p.setAdditionalSearchPath(pybullet_data.getDataPath())
    plane_path = 'Data/plane/plane.urdf'
    planeId = p.loadURDF("plane.urdf", [0, 0, -1])

    # p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 0)
    # the path of robot urdf, should be under the folder ../Data/sawyer_robot/sawyer_description/urdf/sawyer.urdf
    robot_path = 'Data/sawyer_robot/sawyer_description/urdf/sawyer.urdf'
    sawyerId = p.loadURDF(robot_path, [0, 0, 0],
                          [0, 0, 0, 3])
    # the path of the table, should be under the folder ../Data/table/table.urdf
    table_path = 'Data/table/table.urdf'
    tableId = p.loadURDF(table_path, [1.4, 0, -1], p.getQuaternionFromEuler([0, 0, 1.56]))  # load table


    ################################################ Load Object Here!!!!################################################################


    # load the assigned objects, change file name to load different objects
    # p.loadURDF(finlename, position([X,Y,Z]), orientation([a,b,c,d]))
    # center of table is at [1.4,0, -1], adjust postion of object to put it on the table

    # the path of the objects, should be under Data/random_urdfs/000/000.urdf
    # Example
    object_1_path = 'Data/random_urdfs/105/105.urdf'
    objectId1 = p.loadURDF(object_1_path, [1.07, -0.06, 0], p.getQuaternionFromEuler([0, 0, 1.56]))
    #object_2_path = 'Data/random_urdfs/106/106.urdf'
    #objectId2 = p.loadURDF(object_2_path, [1.2, 0.2, 0], p.getQuaternionFromEuler([0, 0, 1.57]))
    #object_3_path = 'Data/random_urdfs/107/107.urdf'
    #objectId3 = p.loadURDF(object_3_path, [1.1, 0.1, 0], p.getQuaternionFromEuler([0, 0, 1.56]))
    #object_4_path = 'Data/random_urdfs/108/108.urdf'
    #objectId4 = p.loadURDF(object_4_path, [1.1, 0.1, 0], p.getQuaternionFromEuler([0, 0, 1.56]))
    #object_5_path = 'Data/random_urdfs/109/109.urdf'
    #objectId5 = p.loadURDF(object_5_path, [1.1, 0.1, 0], p.getQuaternionFromEuler([0, 0, 1.56]))

    # apply texture to objects
    # apply randomly textures from the dtd dataset to each object, each object's texture should be the different.

    # texture_paths = Data/dtd/images/banded/banded_0002.jpg
    # example:
    text_paths1 = 'Data/dtd/images/banded/banded_0002.jpg'
    text_id1 = p.loadTexture(text_paths1)
    p.changeVisualShape(objectId1,-1,textureUniqueId=text_id1)
    #text_paths2 = 'Data/dtd/images/blotchy/blotchy_0003.jpg'
    #text_id2 = p.loadTexture(text_paths2)
    #p.changeVisualShape(objectId2,-1,textureUniqueId=text_id2)
    #text_paths3 = 'Data/dtd/images/braided/braided_0002.jpg'
    #text_id3 = p.loadTexture(text_paths3)
    #p.changeVisualShape(objectId3,-1,textureUniqueId=text_id3)
    #text_paths4 = 'Data/dtd/images/bubbly/bubbly_0038.jpg'
    #text_id4 = p.loadTexture(text_paths4)
    #p.changeVisualShape(objectId4,-1,textureUniqueId=text_id4)
    #text_paths5 = 'Data/dtd/images/bumpy/bumpy_0014.jpg'
    #text_id5 = p.loadTexture(text_paths5)
    #p.changeVisualShape(objectId5,-1,textureUniqueId=text_id5)


    ################################################ ########Insert Camera################################################ ########################################


    # Using the inserted camera to caputure data for training. Save the captured numpy array as image files for later training process.

    width = 256
```

```
    height = 256


    fov = 60
    aspect = width / height
    near = 0.02
    far = 1


    # the view_matrix should contain three arguments, the first one is the [X,Y,Z] for camera location
    #                                                                                the second one is the [X,Y,Z] for target location
    #                                                                                the third one is the [X,Y,Z] for the top of the camera
    # Example:
    # viewMatrix = pb.computeViewMatrix(
    #     cameraEyePosition=[0, 0, 3],
    #     cameraTargetPosition=[0, 0, 0],
    #     cameraUpVector=[0, 1, 0])
    view_matrix = p.computeViewMatrix([1.15, -0.07, 0.25], [1.1, 0, 0], [-1, 0, 0])


    projection_matrix = p.computeProjectionMatrixFOV(fov, aspect, near, far)



    # Get depth values using the OpenGL renderer
    images = p.getCameraImage(width,
                              height,
                              view_matrix,
                              projection_matrix,
                              shadow=True,

renderer=p.ER_BULLET_HARDWARE_OPENGL)
    rgb_opengl = np.reshape(images[2], (height, width, 4)) * 1. / 255.
    depth_buffer_opengl = np.reshape(images[3], [width, height])
    depth_opengl = far * near / (far - (far - near) * depth_buffer_opengl)
    seg_opengl = np.reshape(images[4], [width, height]) * 1. / 255.
    plt.imshow(rgb_opengl)
    plt.savefig('Data/object1images/21.jpg')
    time.sleep(1)



    ################################################################################################################################################


    # p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 1)
    p.resetBasePositionAndOrientation(sawyerId, [0, 0, 0], [0, 0, 0, 1])


    # bad, get it from name! sawyerEndEffectorIndex = 18
    sawyerEndEffectorIndex = 16
    numJoints = p.getNumJoints(sawyerId)  # 65 with ar10 hand
    # print(p.getJointInfo(sawyerId, 3))
    # useRealTimeSimulation = 0
    # p.setRealTimeSimulation(useRealTimeSimulation)
    # p.stepSimulation()
    # all R joints in robot
    js = [3, 4, 8, 9, 10, 11, 13, 16, 21, 22, 23, 26, 27, 28, 30, 31, 32, 35, 36, 37, 39, 40, 41, 44, 45, 46, 48, 49, 50,
          53, 54, 55, 58, 61, 64]
    # lower limits for null space
    ll = [-3.0503, -5.1477, -3.8183, -3.0514, -3.0514, -2.9842, -2.9842, -4.7104, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
          0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.17, 0.85, 0.34,
          0.17]
    # upper limits for null space
    ul = [3.0503, 0.9559, 2.2824, 3.0514, 3.0514, 2.9842, 2.9842, 4.7104, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 1.57, 1.57,
          0.17, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 1.57, 1.57, 0.17, 2.15, 1.5, 1.5]
    # joint ranges for null space
    jr = [0, 0, 0, 0, 0, 0, 0, 0, 1.4, 1.4, 1.4, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4, 1.4,
          0, 1.4, 1.4, 0, 1.3, 1.16, 1.33]
    # restposes for null space
```

```
    rp = [0] * 35
    # joint damping coefficents
    jd = [1.1] * 35


    i = 0
    while 1:
      i += 1
      p.stepSimulation()
      # 0.03 sec/frame
      time.sleep(0.03)
      # increase i to incease the simulation time
      if (i == 1800):
        break
```

Project_IK.py

```
import pybullet as p
import time
import math
import random
import pybullet_data
from datetime import datetime

#######################################
################## Simulation Setup
#######################################
#####################################

clid = p.connect(p.GUI)
if (clid<0):
       p.connect(p.GUI)

p.setGravity(0,0,-9.8)
p.setAdditionalSearchPath(pybullet_data
.getDataPath())
planeId = p.loadURDF("plane.urdf", [0,
0, -1])
p.loadURDF("plane.urdf",[0,0,-.98])
p.configureDebugVisualizer(p.COV_ENABLE
_RENDERING,0)
sawyerId =
p.loadURDF("./sawyer_robot/sawyer_descr
iption/urdf/sawyer.urdf",[0,0,0],
[0,0,0,3])  # load sawyer robot
tableId =
p.loadURDF("table/table.urdf",[1.4,0, -
1],
p.getQuaternionFromEuler([0,0,1.56]))
# load table
#cubeId =
p.loadURDF("cube/marble_cube.urdf",[1.1
,0,0],
p.getQuaternionFromEuler([0,0,1.56]))
# load object, change file name to load
different objects

#######################################
################## Load Object
Here!!!!#############################
#######################################
#######

# load object, change file name to load
different objects
# p.loadURDF(finlename,
position([X,Y,Z]),
orientation([a,b,c,d]))
# center of table is at [1.4,0, -1],
adjust postion of object to put it on
the table
objectId =
p.loadURDF("random_urdfs/108/108.urdf",
[1.20,0.0,0.0],
p.getQuaternionFromEuler([0,0,1.56]))


text_paths4 =
'Data/dtd/images/bubbly/bubbly_0038.jpg
'
text_id4 = p.loadTexture(text_paths4)
p.changeVisualShape(objectId,-
1,textureUniqueId=text_id4)

#######################################
#######################################
#######################################
#####################################

p.configureDebugVisualizer(p.COV_ENABLE
_RENDERING,1)
p.resetBasePositionAndOrientation(sawye
rId,[0,0,0],[0,0,0,1])

#bad, get it from name!
sawyerEndEffectorIndex = 18
sawyerEndEffectorIndex = 16
numJoints = p.getNumJoints(sawyerId)
#65 with ar10 hand
#print(p.getJointInfo(sawyerId, 3))
#useRealTimeSimulation = 0
#p.setRealTimeSimulation(useRealTimeSim
ulation)
#p.stepSimulation()
# all R joints in robot
```

```
js = [3, 4, 8, 9, 10, 11, 13, 16, 21,
22, 23, 26, 27, 28, 30, 31, 32, 35, 36
,37, 39, 40, 41, 44, 45, 46, 48, 49,
50, 53, 54, 55, 58, 61, 64]
#lower limits for null space
ll = [-3.0503, -5.1477, -3.8183, -
3.0514, -3.0514, -2.9842, -2.9842, -
4.7104, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.85, 0.34, 0.17]
#upper limits for null space
ul = [3.0503, 0.9559, 2.2824, 3.0514,
3.0514, 2.9842, 2.9842, 4.7104, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 2.15,
1.5, 1.5]
#joint ranges for null space
jr = [0, 0, 0, 0, 0, 0, 0, 0, 1.4, 1.4,
1.4, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4,
1.4, 0, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4,
1.4, 0, 1.4, 1.4, 0, 1.3, 1.16, 1.33]
#restposes for null space
rp = [0]*35
#joint damping coefficents
jd = [1.1]*35


########################################
################## Inverse Kinematics
Functions
########################################
###################################
#index:51, mid:42, ring: 33, pinky:24,
thumb 62

# move palm (center point) to reach the
target postion and orientation
# input: targetP --> target postion
#        orientation --> target
orientation of the palm
# output: joint positons of all joints
in the robot
#         control joint to move to
correspond joint position
def palmP(targetP, orientation):
       jointP = [0]*65
       jointPoses =
p.calculateInverseKinematics(sawyerId,
20, targetP, targetOrientation =
orientation, jointDamping=jd)
       j=0
       for i in js:
              jointP[i] = jointPoses[j]
              j=j+1

       for i in
range(p.getNumJoints(sawyerId)):


       p.setJointMotorControl2(bodyInde
x=sawyerId,

       jointIndex=i,

       controlMode=p.POSITION_CONTROL,

       targetPosition=jointP[i],

       targetVelocity=0,

       force=500,

       positionGain=0.03,

       velocityGain=1)
       return jointP

# move hand (fingers) to reach the
target postion and orientation
# input: postion and orientation of
each fingers
# output: control joint to move to
correspond joint position

def handIK(thumbPostion, thumbOrien,
indexPostion, indexOrien, midPostion,
midOrien, ringPostion, ringOrien,
pinkyPostion, pinkyOrien,
currentPosition):
       ####### thumb IK
#####################################
#######
       jointP = [0]*65
       joint_thumb = [0]*65
       #lock arm
       for i in [3, 4, 8, 9, 10, 11,
13, 16]:
              jointP[i] =
currentPosition[i]


       jointPoses_thumb =
p.calculateInverseKinematics(sawyerId,
62, thumbPostion, thumbOrien,
jointDamping=jd, lowerLimits = ll,
```

```
         upperLimits = ul,
jointRanges=jr, restPoses =
currentPosition,

         maxNumIterations = 2005,
residualThreshold = 0.01)
       # bulit joint position in all
joints 65
       j=0
       for i in js:
              joint_thumb[i] =
jointPoses_thumb[j]
              j=j+1

       # group thumb mid servo
       if(abs(joint_thumb[61] -
currentPosition[61]) >=
abs(joint_thumb[64] -
currentPosition[64])):
              joint_thumb[64] =
joint_thumb[61]
       else:
              joint_thumb[61] =
joint_thumb[64]

       #build jointP
       for i in [58, 61, 64]:
              jointP[i] =
joint_thumb[i]



       ####### index finger IK
#####################################

       joint_index = [0]*65
       jointPoses_index =
p.calculateInverseKinematics(sawyerId,
51, indexPostion, indexOrien,
jointDamping=jd, lowerLimits = ll,

         upperLimits = ul,
jointRanges=jr, restPoses = jointP,

         maxNumIterations = 2005,
residualThreshold = 0.01)
       # bulit joint position in all
joints 65
       j=0
       for i in js:
              joint_index[i] =
jointPoses_index[j]
              j=j+1

       # group index low servo
       if(abs(joint_index[48] -
joint_thumb[48]) >= abs(joint_index[53]
- joint_thumb[53])):
              joint_index[53] =
joint_index[48]
       else:
              joint_index[48] =
joint_index[53]
       # group index mid servo
       if(abs(joint_index[49] -
joint_thumb[49]) >= abs(joint_index[54]
- joint_thumb[54])):
              joint_index[54] =
joint_index[49]
       else:
              joint_index[49] =
joint_index[54]
       # group index tip servo
       if(abs(joint_index[50] -
joint_thumb[50]) >= abs(joint_index[55]
- joint_thumb[55])):
              joint_index[55] =
joint_index[50]
       else:
              joint_index[50] =
joint_index[55]

       #build jointP
       for i in [48, 49, 53, 54]:
              jointP[i] =
joint_index[i]


       ####### mid finger IK
#####################################


       joint_mid = [0]*65
       jointPoses_mid =
p.calculateInverseKinematics(sawyerId,
42, midPostion, midOrien,
jointDamping=jd, lowerLimits = ll,

         upperLimits = ul,
jointRanges=jr, restPoses = jointP,

         maxNumIterations = 2005,
residualThreshold = 0.01)
       # bulit joint position in all
joints 65
       j=0
       for i in js:
              joint_mid[i] =
jointPoses_mid[j]
```

```
                j=j+1

        # group mid low servo
        if(abs(joint_mid[39] -
joint_index[39]) >= abs(joint_mid[44] -
joint_index[44])):
                joint_mid[44] =
joint_mid[39]
        else:
                joint_mid[39] =
joint_mid[44]
        # group mid mid servo
        if(abs(joint_mid[40] -
joint_index[40]) >= abs(joint_mid[45] -
joint_index[45])):
                joint_mid[45] =
joint_mid[40]
        else:
                joint_mid[40] =
joint_mid[45]
        # group mid tip servo
        if(abs(joint_mid[41] -
joint_index[41]) >= abs(joint_mid[46] -
joint_index[46])):
                joint_mid[46] =
joint_mid[41]
        else:
                joint_mid[41] =
joint_mid[46]

        #build jointP
        for i in [39, 40, 44, 45]:
                jointP[i] = joint_mid[i]

        ####### ring finger IK
#####################################


        joint_ring = [0]*65
        jointPoses_ring =
p.calculateInverseKinematics(sawyerId,
33, ringPostion, ringOrien,
jointDamping=jd, lowerLimits = ll,

          upperLimits = ul,
jointRanges=jr, restPoses = jointP,

          maxNumIterations = 2005,
residualThreshold = 0.01)
        # bulit joint position in all
joints 65
        j=0
        for i in js:
                joint_ring[i] =
jointPoses_ring[j]
                j=j+1

        # group ring low servo
        if(abs(joint_ring[30] -
joint_mid[30]) >= abs(joint_ring[35] -
joint_mid[35])):
                joint_ring[35] =
joint_ring[30]
        else:
                joint_ring[30] =
joint_ring[35]
        # group ring mid servo
        if(abs(joint_ring[31] -
joint_mid[31]) >= abs(joint_ring[36] -
joint_mid[36])):
                joint_ring[36] =
joint_ring[31]
        else:
                joint_ring[31] =
joint_ring[36]
        # group ring tip servo
        if(abs(joint_ring[32] -
joint_mid[32]) >= abs(joint_ring[37] -
joint_mid[37])):
                joint_ring[37] =
joint_ring[32]
        else:
                joint_ring[32] =
joint_ring[37]

        #build jointP
        for i in [30, 31, 35, 36]:
                jointP[i] = joint_ring[i]


        ####### pinky finger IK
#####################################


        joint_Pinky = [0]*65
        jointPoses_Pinky =
p.calculateInverseKinematics(sawyerId,
24, pinkyPostion, pinkyOrien,
jointDamping=jd, lowerLimits = ll,

          upperLimits = ul,
jointRanges=jr, restPoses = jointP,

          maxNumIterations = 2005,
residualThreshold = 0.01)
        # bulit joint position in all
joints 65
        j=0
        for i in js:
```

```
                joint_Pinky[i] =
jointPoses_Pinky[j]
                j=j+1

        # group ring low servo
        if(abs(joint_Pinky[21] -
joint_ring[21]) >= abs(joint_Pinky[26]
- joint_ring[26])):
                joint_Pinky[26] =
joint_Pinky[21]
        else:
                joint_Pinky[21] =
joint_Pinky[26]
        # group ring mid servo
        if(abs(joint_Pinky[22] -
joint_ring[22]) >= abs(joint_Pinky[27]
- joint_ring[27])):
                joint_Pinky[27] =
joint_Pinky[22]
        else:
                joint_Pinky[22] =
joint_Pinky[27]
        # group ring tip servo
        if(abs(joint_Pinky[23] -
joint_ring[23]) >= abs(joint_Pinky[28]
- joint_ring[28])):
                joint_Pinky[28] =
joint_Pinky[23]
        else:
                joint_Pinky[23] =
joint_Pinky[28]

        #build jointP
        for i in [21, 22, 26, 27]:
                jointP[i] =
joint_Pinky[i]

        ####### move joints
#####################################


        for i in
range(p.getNumJoints(sawyerId)):


        p.setJointMotorControl2(bodyInde
x=sawyerId,

        jointIndex=i,

        controlMode=p.POSITION_CONTROL,

        targetPosition=jointP[i],

        targetVelocity=0,

        force=500,

        positionGain=0.03,

        velocityGain=1)

#####################################
################## Simulation
#####################################
#################################

# postions and orientations of hand
joints, get from
project_joint_control.py
thumbP = [1.1625325066735959, -
0.10449498868413923, -
0.06201584642636722]
thumbOrien = [0.08481418954301742,
0.7830066745269654, -
0.5522344438557819,
0.27339389151493354]
indexP = [1.2110404422988899, -
0.08285140073825516, -
0.05607174103449016]
indexOrien = [0.22403382942495043,
0.8333010224415348, -
0.37631931063330054,
0.3373455583160427]
midP = [1.2011953241692468, -
0.08000185162493396, -
0.07528736347652193]
midOrien = [0.21594704347783641,
0.8420156613112914, -
0.35639366389448973, 0.342578541148671]
ringP = [1.1760778497594007, -
0.07932664466250612, -
0.0817731348857553]
ringOrien = [0.195674772957788,
0.8613029523494512, -
0.3068616004516512, 0.3545483967955411]
pinkyP = [1.1407944735130955, -
0.08432016557315077, -
0.07860648714769963]
pinkyOrien = [0.1377758313070547,
0.8987142263166167, -
0.16828415544475644,
0.3808031023377046]

currentP = [0]*65
k = 0
while 1:
        k = k + 1

        # move palm to target postion
```

```
	i=0
	while 1:
		i+=1
		p.stepSimulation()
		#currentP = palmP([1.15,
-0.07, 0.25],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
		#currentP = palmP([2.00,
-0.07, 0.25],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
		currentP = palmP([1.20,-
0.10,0.1], p.getQuaternionFromEuler([0,
-math.pi*1.2, 0]))
		time.sleep(0.03)
		if(i==150):
			break

	i=0
	while 1:
		i+=1
		p.stepSimulation()
		#currentP = palmP([1.15,
-0.05, -0.15],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
		currentP = palmP([1.20,-
0.17,-0.3],
p.getQuaternionFromEuler([0, -
math.pi*1.2, 0]))
		time.sleep(0.03)
		if(i==80):
			break
	# grasp object
	i=0
	while 1:
		i+=1
		p.stepSimulation()
		time.sleep(0.03)
		handIK(thumbP,
thumbOrien, indexP, indexOrien, midP,
midOrien, ringP, ringOrien, pinkyP,
pinkyOrien, currentP)
		if(i==80):
			break

	# pick up object
	i=0
	while 1:
		i+=1
		p.stepSimulation()
		#currentP = palmP([1.15,
-0.07, 0.1],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
		currentP = palmP([1.20, -
0.20, 0.2],
p.getQuaternionFromEuler([0, -
math.pi*1.2, 0]))

		time.sleep(0.03)
		if(i==100):
			break
	break

p.disconnect()
print("disconnected")
```

project_control_joint.py

```
import pybullet as p
import time
import math
import random
import pybullet_data
from datetime import datetime

#######################################
################## Simulation Setup
#######################################
#######################################

clid = p.connect(p.GUI)
if (clid<0):
	p.connect(p.GUI)

p.setGravity(0,0,-9.8)
p.setAdditionalSearchPath(pybullet_data
.getDataPath())
planeId = p.loadURDF("plane.urdf", [0,
0, -1])
#p.loadURDF("plane.urdf",[0,0,-.98])
p.configureDebugVisualizer(p.COV_ENABLE
_RENDERING,0)
sawyerId =
p.loadURDF("./sawyer_robot/sawyer_descr
iption/urdf/sawyer.urdf",[0,0,0],
[0,0,0,3])  # load sawyer robot
tableId =
p.loadURDF("table/table.urdf",[1.4,0, -
1],
p.getQuaternionFromEuler([0,0,1.56]))
# load table
#objectId =
p.loadURDF("cube/marble_cube.urdf",[1.1
,0,0],
p.getQuaternionFromEuler([0,0,1.56]))
```

```
#######################################
################## Load Object
Here!!!!#################################
#######################################
#######

# load object, change file name to load
different objects
# p.loadURDF(finlename,
position([X,Y,Z]),
orientation([a,b,c,d]))
# center of table is at [1.4,0, -1],
adjust postion of object to put it on
the table
#objectId =
p.loadURDF("random_urdfs/001/001.urdf",
[1.1,0,0],
p.getQuaternionFromEuler([0,0,1.56]))
objectId =
p.loadURDF("random_urdfs/108/108.urdf",
[1.20,0,0.0],
p.getQuaternionFromEuler([0,0,1.56]))
#[1.20,0.0,0.15]


text_paths4 =
'Data/dtd/images/bubbly/bubbly_0038.jpg
'
text_id4 = p.loadTexture(text_paths4)
p.changeVisualShape(objectId,-
1,textureUniqueId=text_id4)
#######################################
#######################################
#######################################
#####################################


p.configureDebugVisualizer(p.COV_ENABLE
_RENDERING,1)
p.resetBasePositionAndOrientation(sawye
rId,[0,0,0],[0,0,0,1])
#bad, get it from name!
sawyerEndEffectorIndex = 18
sawyerEndEffectorIndex = 16
numJoints = p.getNumJoints(sawyerId)
#65 with ar10 hand
#print(p.getJointInfo(sawyerId, 3))
#useRealTimeSimulation = 0
#p.setRealTimeSimulation(useRealTimeSim
ulation)
#p.stepSimulation()
# all R joints in robot
js = [3, 4, 8, 9, 10, 11, 13, 16, 21,
22, 23, 26, 27, 28, 30, 31, 32, 35, 36
,37, 39, 40, 41, 44, 45, 46, 48, 49,
50, 53, 54, 55, 58, 61, 64]
#lower limits for null space
ll = [-3.0503, -5.1477, -3.8183, -
3.0514, -3.0514, -2.9842, -2.9842, -
4.7104, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.17, 0.17, 0.17, 0.17, 0.17,
0.17, 0.85, 0.34, 0.17]
#upper limits for null space
ul = [3.0503, 0.9559, 2.2824, 3.0514,
3.0514, 2.9842, 2.9842, 4.7104, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 1.57,
1.57, 0.17, 1.57, 1.57, 0.17, 2.15,
1.5, 1.5]
#joint ranges for null space
jr = [0, 0, 0, 0, 0, 0, 0, 0, 1.4, 1.4,
1.4, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4,
1.4, 0, 1.4, 1.4, 0, 1.4, 1.4, 0, 1.4,
1.4, 0, 1.4, 1.4, 0, 1.3, 1.16, 1.33]
#restposes for null space
rp = [0]*35
#joint damping coefficents
jd = [1.1]*35
#######################################
################## Inverse Kinematics
Function
#######################################
#####################################

# Finger tip ID: index:51, mid:42,
ring: 33, pinky:24, thumb 62
# Palm ID: 20
# move palm (center point) to reach the
target postion and orientation
# input: targetP --> target postion
#        orientation --> target
orientation of the palm
# output: joint positons of all joints
in the robot
#         control joint to correspond
joint position
def palmP(targetP, orientation):
       jointP = [0]*65
       jointPoses =
p.calculateInverseKinematics(sawyerId,
20, targetP, targetOrientation =
orientation, jointDamping=jd)
       j=0
       for i in js:
              jointP[i] = jointPoses[j]
              j=j+1
```

```
        for i in
range(p.getNumJoints(sawyerId)):


        p.setJointMotorControl2(bodyInde
x=sawyerId,
        jointIndex=i,
        controlMode=p.POSITION_CONTROL,
        targetPosition=jointP[i],
        targetVelocity=0,
        force=500,
        positionGain=0.03,
        velocityGain=1)
        return jointP

########################################
################## Hand Direct Control
Functions
########################################
################################

#control the lower joint and middle
joint of pinky finger, range both [0.17
- 1.57]
def pinkyF(lower, middle):
        p.setJointMotorControlArray(body
Index=sawyerId,
jointIndices=[21, 26, 22, 27],
controlMode=p.POSITION_CONTROL,
targetPositions=[lower, lower, middle,
middle],
targetVelocities=[0, 0, 0, 0],
forces=[500, 500, 500, 500],
positionGains=[0.03, 0.03, 0.03, 0.03],
velocityGains=[1, 1, 1, 1])

#control the lower joint and middle
joint of ring finger, range both [0.17
- 1.57]
def ringF(lower, middle):
        p.setJointMotorControlArray(body
Index=sawyerId,
jointIndices=[30, 35, 31, 36],
controlMode=p.POSITION_CONTROL,
targetPositions=[lower, lower, middle,
middle],
targetVelocities=[0, 0, 0, 0],
forces=[500, 500, 500, 500],
positionGains=[0.03, 0.03, 0.03, 0.03],
velocityGains=[1, 1, 1, 1])

#control the lower joint and middle
joint of mid finger, range both [0.17 -
1.57]
def midF(lower, middle):
        p.setJointMotorControlArray(body
Index=sawyerId,
jointIndices=[39, 44, 40, 45],
controlMode=p.POSITION_CONTROL,
targetPositions=[lower, lower, middle,
middle],
targetVelocities=[0, 0, 0, 0],
forces=[500, 500, 500, 500],
positionGains=[0.03, 0.03, 0.03, 0.03],
velocityGains=[1, 1, 1, 1])

#control the lower joint and middle
joint of index finger, range both [0.17
- 1.57]
def indexF(lower, middle):
        p.setJointMotorControlArray(body
Index=sawyerId,
jointIndices=[48, 53, 49, 54],
controlMode=p.POSITION_CONTROL,
targetPositions=[lower, lower, middle,
middle],
targetVelocities=[0, 0, 0, 0],
```

```
forces=[500, 500, 500, 500],

positionGains=[0.03, 0.03, 0.03, 0.03],

velocityGains=[1, 1, 1, 1])

#control the lower joint and middle
joint of thumb, range: low [0.17 -
1.57], mid [0.34, 1.5]
def thumb(lower, middle):
	p.setJointMotorControlArray(body
Index=sawyerId,

jointIndices=[58, 61, 64],

controlMode=p.POSITION_CONTROL,

targetPositions=[lower, middle,
middle],

targetVelocities=[0, 0, 0],

forces=[500, 500, 500],

positionGains=[0.03, 0.03, 0.03],

velocityGains=[1, 1, 1])

########################################
################## Simulation
########################################
###################################

currentP = [0]*65
k = 0
while 1:
	k = k + 1

	# move palm to target postion
	i=0
	while 1:
		i+=1
		p.stepSimulation()
		#currentP = palmP([1.15,
-0.07, 0.25],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
		currentP = palmP([1.20,-
0.10,0.1], p.getQuaternionFromEuler([0,
-math.pi*1.2, 0]))
		time.sleep(0.03)
		if(i==150):
			break
	i=0
	while 1:
		i+=1

		p.stepSimulation()
		#currentP = palmP([1.15,
-0.05, -0.15],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
		currentP = palmP([1.20,-
0.17,-0.3],
p.getQuaternionFromEuler([0, -
math.pi*1.2, 0]))
		if(i==80):
			break
	# grasp object and print
postions and orientations of finger
tips
	# record the postions and
orentations for inverse kinematics
	i=0
	while 1:
		i+=1
		p.stepSimulation()
		time.sleep(0.03)
		thumb(2.5, 1.57)
		if(i==10):
#angel for each
finger
			#indexF(1.4, 2.47)
			#midF(1.4, 1.47)
			#ringF(1.4, 1.47)
			#pinkyF(1.5, 1.57)
			indexF(1.25, 2.37)
			midF(1.25, 1.37)
			ringF(1.25, 1.37)
			pinkyF(1.5, 1.47)


		if(i==50):
			#index:51, mid:42,
ring: 33, pinky:24, thumb 62
			print("Thumb
Position: ", p.getLinkState(sawyerId,
62)[0]) # get position of thumb tip
			print("Thumb
Orientation: ",
p.getLinkState(sawyerId, 62)[1]) # get
orientation of thumb tip
			print("Index
Position: ", p.getLinkState(sawyerId,
51)[0])
			print("Index
Orientation: ",
p.getLinkState(sawyerId, 51)[1])
```

```
                        print("Mid:
Position: ", p.getLinkState(sawyerId,
42)[0])
                        print("Mid:
Orientation: ",
p.getLinkState(sawyerId, 42)[1])
                        print("Ring
Position: ", p.getLinkState(sawyerId,
33)[0])
                        print("Ring
Orientation: ",
p.getLinkState(sawyerId, 33)[1])
                        print("Pinky
Position: ", p.getLinkState(sawyerId,
24)[0])
                        print("Pinky
Orientation: ",
p.getLinkState(sawyerId, 24)[1])
                        print("Palm
Position: ", p.getLinkState(sawyerId,
20)[0])
                        print("Palm
Orientation: ",
p.getLinkState(sawyerId, 20)[1])
                        break

        # pick up obejct
        i=0
        while 1:
                i+=1
                p.stepSimulation()
                #currentP = palmP([1.15,
-0.07, 0.1],
p.getQuaternionFromEuler([0, -
math.pi*1.5, 0]))
                currentP = palmP([1.20, -
0.20, 0.2],
p.getQuaternionFromEuler([0, -
math.pi*1.2, 0]))
                time.sleep(0.03)
                if(i==200):
                        break
        break
p.disconnect()
print("disconnected")
```

VII. REFERENCE

Robotics Industries Association, 1999.

[1] B. Luitel and G. K. Venayagamoorthy, "Decentralized Asynchronous Learning in Cellular Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 11, pp. 1755-1766, Nov. 2012, doi: 10.1109/TNNLS.2012.2216900.

[2] J. Burgner-Kahrs, D. C. Rucker and H. Choset, "Continuum Robots for Medical Applications: A Survey," in IEEE Transactions on Robotics, vol. 31, no. 6, pp. 1261-1280, Dec. 2015, doi: 10.1109/TRO.2015.2489500.

[3] L. Marques et al., "Automation of humanitarian demining: The 2016 humanitarian robotics and automation technology challenge," 2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA), Kollam, 2016, pp. 1-7, doi: 10.1109/RAHA.2016.7931893.

[4] D. P. Gravel, "Flexible robotic assembly efforts at Ford Motor Company IEEE/RSJ international conference on intelligent robots and systems," Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), Las Vegas, NV, USA, 2003, pp. 2522-2527 vol.3, doi: 10.1109/IROS.2003.1249249.

[5] David Gravel and Wyatt S. Newmann, "Force Controlled Robotic Assembly Efforts at Ford Motor Company", NIST Performance Merrics for Inrelligent Systems, 1999.

[6] Jing Wei"Inte1ligent Robotic Learning using Guided Evolutionary Simulated Annealing" Case Westem Reserve University Masters Thesis 2002

[7] ICBL, Landmine Monitor 2015, International Campaign to Ban Landmines - Cluster Munition Coalition (ICBL-CMC), 2015.

[8] L. Marques, G. Muscato, Y. Yvinec, M. Peichl, G. Alli, and G. A. Turnbull, The EOD Future in the Light of NATO EOD Demonstrations and Trials 2012. NATO Centre of Excellence for Explosive Ordnance Disposal, 2013, ch. Sensors for close-in detection of explosive devices in the framework of FP7-TIRAMISU project, pp. 63–74.

[9] L. Marques, A. de Almeida, M. Armada, R. Fernandez, H. Montes, ´ P. G. de Santos, and Y. Baudoin, “State of the art review on mobile robots and manipulators for humanitarian demining,” in Proceedings of the 10th International IARP Workshop (HUDEM 2012). IARP, 2012.

[10] D. Mikulic, Design of demining machines. Springer, 2013

[11] G. Hirzinger, “Robots in space—A survey,” Adv. Robot., vol. 9, pp. 625–651, 1994.

[12] I. Leite, C. Martinho, and A. Paiva, “Social robots for long-term interaction: A survey,” Int. J. Soc. Robot., vol. 5, no. 2, pp. 291–308, 2013.

[13] R. A. Beasley, “Medical Robots: Current systems and research directions,” J. Robot., vol. 2012, pp. 1–14, 2012.

[14] S. Maeso, M. Reza, J. Mayol, J. B. M. Guerra, E. Andradas, and M. Plana, “Efficacy of the da vinci surgical system in abdominal surgery compared with that of laparoscopy: A systematic review and meta-analysis,” Ann. Surgery, vol. 252, no. 2, pp. 254–262, 2010.

[15] G. Turchetti, I. Palla, F. Pierotti, and A. Cuschieri, “Economic evaluation of da vinci-assisted robotic surgery: A

systematic review," Surgical Endoscopy, vol. 26, no. 3, pp. 589–606, 2012.

[16] L. Chua and L. Yang, "Cellular neural networks," in Proc. IEEE Int. Symp. Circuits Syst., vol. 2. Mar. 1988, pp. 985–988.

[17] P. Tzionas, A. Thanailakis, and P. Tsalides, "A hybrid cellular automaton/neural network classifier for multi-valued patterns and its VLSI implementation," Integr., VLSI J., vol. 20, no. 2, pp. 211–237, 1996.

[18] T. Su, Y. Du, Y. Cheng, and Y. Su, "A fingerprint recognition system using cellular neural networks," in Proc. Int. Workshop Cellular Neural Netw. Appl., 2005, pp. 170–173.

[19] L. Chua and L. Yang, "Cellular neural networks: Theory," IEEE Trans. Circuits Syst., vol. 35, no. 10, pp. 1257–1272, Oct. 1988.

[20] I. Aizenberg, N. Aizenberg, J. Hiltner, C. Moraga, and E. M. zu Bexten, "Cellular neural networks and computational intelligence in medical image processing," Image Vis. Comput., vol. 19, no. 4, pp. 177–183, 2001.

[21] Z. Zeng and W. X. Zheng, "Multistability of neural networks with timevarying delays and concave-convex characteristics," IEEE Trans. Neural Netw. Learn. Syst., vol. 23, no. 2, pp. 293–305, Feb. 2012