Introduction to Intelligent to Robotics Project

INTRODUCTION

Background about robotic Object Recognition and Grasping

Over the past 20 years, research in the field of robotic grasping has been conducted with a minor bias toward the creation of the theoretical foundation and the conclusions of the analyses.

There is a huge demand for for the interior robots. Most of them are used in the house hold purposes and offices, are frequently used for grasping activities for finding the object and grasping the object. Moreover it is very difficult to code or write an Algorithm because of the various kind objects and their gripping positions for executing the real time for robots.

In the robots technology many of robotics grasping methods were introduced by the researchers. Recently deep learning algorithms are emerged as popular methods for grasping. Asa a result deep learning methods are being used for indoor robots for viewing ,grasping the objects.

Understanding inverse and forward kinematics and potential applications:

Forward kinematics:

The determination of a robot's end-effector frame's position and orientation from its joint coordinates is known as its forward kinematics.

Inverse kinematics:

Find robot configurations that satisfy a homogeneous transform if one is provided.

Methods of inverse kinematics:

- Analytic Inverse Kinematics
- Inverse Position: findfirst three joint configurations.

- Inverse Orientation: find the last three joint configurations

Numerical Inverse Kinematics:

 In the event that the inverse kinematics equations do not accept analytical solutions, iterative numerical techniques can be used.

 Even in situations when an analytical solution does exist, the precision of these answers is frequently enhanced by numerical techniques.

The goal is to modify the inverse kinematics equations so that they can be solved using current numerical techniques.

- Newton–Raphson Method
- Optimization Methods

Procedure of the project

- First we need to get familiar with 3D robot simulation and programming
- Install pybullet along with Anaconda (or) Install and setup pybullet and python on your system
- Load assigned object in the environment
- Record the tray location in environment
- After loading the object id and tray location we need adjust the hand position by using sliders and record the hand readings , palm position, and orientation values.
- Later we will use inverse kinematics for approaching to the object.
- Then will grasp the object by using the hand control Function

Summary:

This project illustrates the object grasping and moving by a robot in pybullet environment. The robot is pre-trained in this environment to find the things. Here we will study the intelligent programming of the robot and train our own object detector to recognize the designated objects.

Project Descriptions

Procedure:

- Install pybullet with Anaconda
- Install python3
- Setup Pybullet Environment
- Load assigned object in the environment
- Then we will run the sawyer_Env.py for loading and object id and the tray location
- After completing the above step we need to run the tune_grasping.py file for recording the Hand readings , palm position, Orientation values.
- Later we will record the values and load them into the main.py file .
- Will use **Inverse Kinematics Function** in main.py to control the position and orientation of the arm **Hand Direct Control Function** in main.py to control the hand

- After reaching the tray locations , We need to drop the object from the robot hand into the tray.

Functions Used:

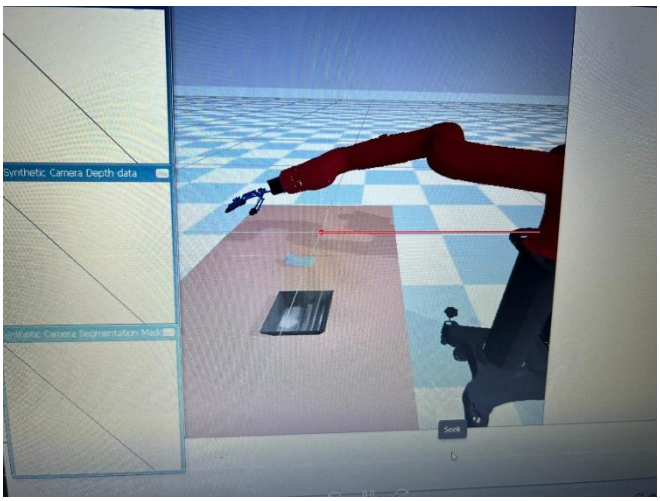
Inverse Kinematics: In robotics kinematics makes use of the kinematics equations to determine the joint parameter that provide position and rotation for robot. By determining the robot movement so that we can move it from the initial position to the desired position is called motion planning. The serial manipulators' inverse kinematics has long been a subject of research. It's essential for managing a manipulator. In real-time manipulator control, the inverse kinematics problem demands a lot of computation and takes a while to solve. Actuators carry out their tasks in joint space, whereas manipulators operate in Cartesian space. The orientation matrix and position vector are enclosed in Cartesian space. Conversely, joint angles serve as a representation of the joint space. Inverse kinematics is needed to convert the end-position and orientation of an effector on a manipulator from Cartesian space to joint space. Geometric and algebraic techniques are employed in order to derive the inverse kinematics solution analytically.

Forward Kinematics. A manipulator is made up of serial links that are joined together by revolute or prismatic joints from the base frame to the end-effector. Calculating an end-position effector's and orientation in terms of joint variables is known as forward kinematics. To get forward kinematics for a robot mechanism in a methodical manner, a suitable kinematics model should be employed. The Denavit-Hartenberg method, which comprises four parameters, is the most often used technique for determining robot kinematics. Hand Direct Control Activities: For the project, each hand's finger has a unique role. Each function stands for a distinct finger, for example the pinkyF function for the final finger and the indexF function for the second finger. moving each finger of the hand we call respective functions.

Results

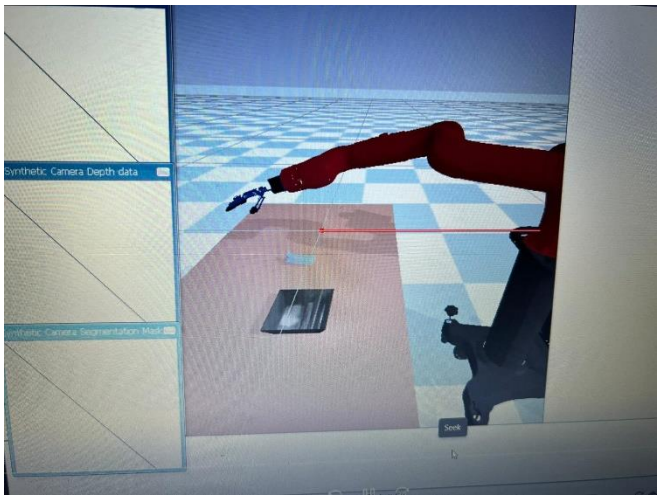
1) **loading object and tray**

   Here will load the object and tray onto the robot table after the we will copy the object and tray values and update it in the codes.



2) Deciding the grasping position

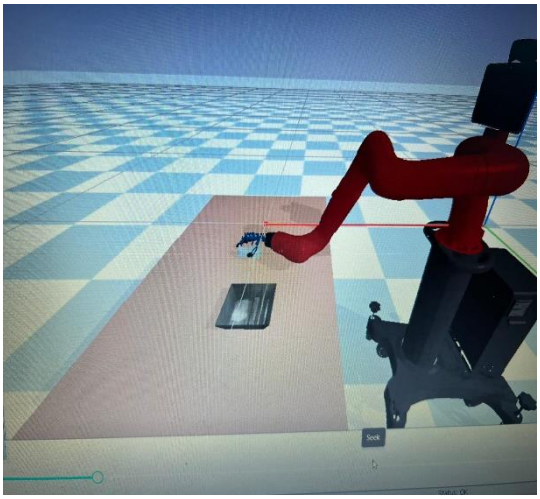
By using the shape of the object we will be deciding the grasping position. We use five fingers to control the grasp. The grasping type is developed by using grasp position and the fingers used.



3) Approaching the object:

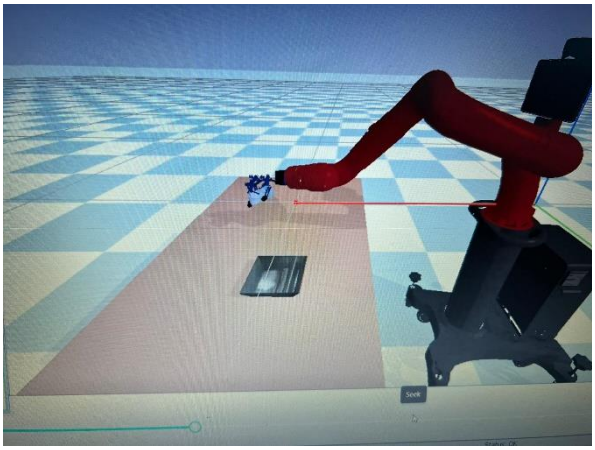
By controlling the palm and orientation near to the object and for grasping the object.

Once the hand reaches to the object try to grasp the object after grasping the object record the readings.



4) Picking up the object

After successful grasping the object by using the arm controls pick the object and place it in the tray.



DISCUSSION OF RESULTS:

1) The below values are the location of the tray.

Tray_X	Tray_y
1.0183010196954019	0.3

2) The below are the values of the robot hand readings of the object.

3) The main motive behind the project is to grasp object by using the inverse kinematics and drop the object in the tray. The output of the project are attached in the above section which shows the grasping of the object and dropping at the tray location.

Conclusion

In this project the the robot position is predefined and by running the tune_grasping.py we will get hand readings and palm position and orientation values. After getting the values we will load the values in main.py file and we use some inverse kinematics algorithms to grasp the object and put it in the tray.

The main motive of the project is to grasp the object and put it in the tray we then we will achieve our targeted goal

Appendix:

Main.py

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

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


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


p.setAdditionalSearchPath(pybullet_data.getDataPath())


planeId = p.loadURDF("plane.urdf", [0, 0, -1])


p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 0)
sawyerId = p.loadURDF("./sawyer_robot/sawyer_description/urdf/sawyer.urdf", [0, 0, 0], [0, 0, 0, 3],
```

```
                useFixedBase=1)     # load sawyer robot

tableId = p.loadURDF("./table/table.urdf", [1.1, 0.000000, -0.3],

p.getQuaternionFromEuler([(math.pi / 2), 0, (math.pi / 2)]), useFixedBase=1, flags=8)


#####################################################################################
# load object, change file name to load different objects
# p.loadURDF(finlename, position([X,Y,Z]), orientation([a,b,c,d]))
# Example:
#objectId = p.loadURDF("random_urdfs/016/016.urdf", [1.25 ,0.25,-0.1], p.getQuaternionFromEuler([0,0,1.56])) # pi*0.5

xpos = 1.1
ypos = 0
ang = 3.14 * 0.5
orn = p.getQuaternionFromEuler([0, 0, ang])

object_path ="random_urdfs/163/163.urdf"
objectId = p.loadURDF(object_path, xpos, ypos, -0.03, orn[0], orn[1], orn[2], orn[3])


#####################################################################################
tray_x = 1.0183010196954019
tray_y = 0.3


trayId = p.loadURDF("./tray/tray.urdf", [tray_x, tray_y, 0], [0, 0, 0, 3])


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

sawyerEndEffectorIndex = 16
numJoints = p.getNumJoints(sawyerId)  # 65 with ar10 hand
```

```
# 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, 0.17, 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, 19, 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(bodyIndex=sawyerId,
                                jointIndex=i,
```

```
controlMode=p.POSITION_CONTROL,
                        targetPosition=jointP[i],
                        targetVelocity=0,
                        force=50000,
                        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]


#hand = [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]
# handReading = [0.2196998776260993, 0.9841056922424084,
0.16991782178342238,
0.21967883521345558,
0.9846229478397389,
0.1699958046620013,
0.5711534611694058,
0.5914229523765463,
0.16999954970542672,
0.573730600144428, 0.5902151809391006,
0.17000660753266578,
0.9359158730554522, 0.265116872922352,
0.170003190706592, 0.9361250259528252,
#                       0.2652466938834658,
0.17003347470289248,
0.9068051254489781,
0.2490975329073341,
0.17008149880963058,
0.9066050389575453,
0.2502858674912193,
0.16999999999999976,
#                       1.5698468053021237,
0.34006621802344955,
0.3400508342876441]


def pinkyF(lower, middle):

p.setJointMotorControlArray(bodyIndex=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(bodyIndex=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(bodyIndex=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(bodyIndex=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(bodyIndex=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])

0.6579502400620709,
0.17077031419079114,
1.1386768699572838,
1.0894212302872914,
0.19751315819872878,
1.1428490439060197,
1.0377958048053253,
0.1714625969253612,
1.5689115835924374,
0.34084727936496095,
0.3406543690380435]

grasp_orientation = [1.571832769393921, 2.837588395277343, 0.8169429206848145]

grasp_palmPosition = [1.1, -0.135, -0.02]

handClose = [0.9518602488025184,
0.9525470182946459,
1.5474074983539889,
0.9360509968123255,
0.9350156258164676,
0.22392050064941072,
0.5944632666102456,
0.5793631323758753,
1.16989640098739248,
0.5854447548559536,
0.5821846276001464,
0.17047160298302724,
0.8805923372517218,
0.3838697556762429,
0.22621806265375474,
0.9384265748909743,
0.2984993223313543,
0.17173215163837467,
1.2230835819966675,
0.22276218388972487,
0.17234180343293869,
1.2271037262924913,
0.24140473736653117,
0.17141295627521993,
1.5777646827634443,
0.5456018336264689,
0.5480414005648172]

##
Input Value Here##

handInitial = [0.26121505230261155,
0.17934456879275282,
1.1146831961129855,
0.26493107969000046,
0.17907812933399364,
0.3545015109713303,
0.22064881672076855,
0.5670891755018503,
1.2468723794210015,
0.18128529903142357,
0.5851992296711084,
0.17014392502411396,
0.5872846661538109,
0.7724645929203886,
0.22290265600560122,
0.6284191039153812,

pu_palmPosition = [0.94, 0.0, 0.25]

pu_orientation = [1.2892775535683597, 2.827588395376452, 1.2237756253388918]

final_palmPosition = [0.9110235933478783, 0.17, 0.25]

final_orientation = [0.727097749711193, 2.2984782021237885, 0.4631795893178812]

handOpen = [0.23791776350578354, 0.1757544910936571, 0.29764745486837116, 0.23754362279773994, 0.16999597089686969, 0.3775939322409635, 0.18168474927512803, 0.1758926369336866, 0.1845598545607197, 0.17030196791446582, 0.17011452285795407, 0.1700102825769791, 0.1717451657212141, 0.17208485272719378, 0.17001139107791998, 0.17009669284461805, 0.17393682045571487, 0.16999999988999997, 0.170107095758437, 0.26215783072686536, 0.16999960893597742, 0.17000115527206383, 0.25230788515373766, 0.1700390287977374, 0.8499999370367525, 0.3401734466845466, 0.3400368734419835]

##

initial_palmPosition = [0.94, 0.0, 0.2]

initial_orientation = [1.2892775535583496, 2.827588395276342, 1.2237756252288818]

initial_palmPosition = [initial_palmPosition[0]-0.1,initial_palmPosition[1]-0.05,initial_palmPosition[2]]

##

p.resetDebugVisualizerCamera(cameraDistance=0.5, cameraYaw=0, cameraPitch=-150, cameraTargetPosition=[0.8,0.5,0.2])

p.setGravity(0, 0, -10)

write the code for step 9-12

''' Step 1: move robot to the initial position '''

Parameters:

arm: initial_palmPosition, initial_orientation

hand: handInitial

''' Step 2: move robot to the grasping position '''

Parameters:

arm: grasp_orientation, grasp_palmPosition

```
        i += 1
        # p.stepSimulation()
        currentP = palmP([initial_palmPosition[0], initial_palmPosition[1], initial_palmPosition[2]],
p.getQuaternionFromEuler([initial_orientation[0], initial_orientation[1], initial_orientation[2]]))
        time.sleep(0.03)
        p.stepSimulation()

        if (i == 100):
            print('robot reached initial position')
            break
        # moving palm to grasp position.
''' Step 3: grasp object '''

# Parameters:
  # hand: handClose

''' Step 4: pick up  the object '''

# Parameters:
  # arm: pu_palmPosition, pu_orientation

''' Step 4: move object to the tray '''

# Parameters:
  # arm: final_palmPosition, final_orientation


''' Step 5: put the object in the tray '''

# Parameters:
  # hand: handOpen

#inital position
k = 0
while 1:
    k = k + 1

    # move palm to target postion
    i = 0
    while 1:
        i += 1
        #p.stepSimulation()
        currentP = palmP([1.0, -0.005, -0.12],
p.getQuaternionFromEuler([1.4292775535583496, grasp_orientation[1], grasp_orientation[2]]))
        #low [0.17 - 1.57], mid [0.34, 1.5]
        #thumb(-1.57,0.34)
```

```
        time.sleep(0.03)
        p.stepSimulation()

        if (i == 100):
            print(' Robot grasped object position')
            break
        #handclose
    i = 0
    while 1:
        i+=1
        thumb(1.5, 0.5)
        indexF(1.5, 1.57)
        midF(0.5, 1.57)
        ringF(0.5,1.57)
        pinkyF(0.9, 1.57)
        time.sleep(0.03)
        p.stepSimulation()

        if (i == 100):
            print('Object position closed suceefully')
            break
        # pickup
    i = 0
    while 1:
        i += 1
        # p.stepSimulation()
        currentP = palmP([pu_palmPosition[0], pu_palmPosition[1], pu_palmPosition[2]],
p.getQuaternionFromEuler([pu_orientation[0], pu_orientation[1], pu_orientation[2]]))
        time.sleep(0.03)
        p.stepSimulation()

        if (i == 100):
            print('picking up object')
            break
        # object to tray
    i = 0
    while 1:
        i += 1
        p.stepSimulation()
        currentP = palmP([0.7110235933378673, 0.30, -0.25],
p.getQuaternionFromEuler([1.0546246528625488, 2.827698495276342, 1.2238756252288818]))
        #thumb(-1.57, 1.5)
        #indexF(handOpen[2], handOpen[3])
        thumb(-1.57, -1.57)
        indexF(-1.57, -1.57)
        midF(-1.57, -1.57)
        ringF(-1.57, -1.57)
        pinkyF(-1.57, -1.57)
        time.sleep(0.03)

        time.sleep(0.03)
        p.stepSimulation()

        if (i == 100):
```

```
        print('move object to tray')
        break
    #handopen
i = 0
'''while 1:
    i += 1
    thumb(handOpen[0], handOpen[1])
    indexF(handOpen[2], handOpen[3])
    midF(handOpen[4], handOpen[5])
    ringF(handOpen[6], handOpen[7])
    pinkyF(handOpen[8], handOpen[9])
    time.sleep(0.03)
    p.stepSimulation()



    if (i == 100):
        print('object left in tray successfully')
        break
    '''




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

REFERENCES:

- Göngör F, Tutsoy Ö. Design and Implementation of a Facial Character Analysis Algorithm for Humanoid Robots. *Robotica* 2019; 37(11): 1850–1866.
- Bicchi A, Kumar V. Robotic grasping and contact: a review. In: Proceedings 2000 ICRA millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065), 24–28 April 2000, pp. 348–353. San Francisco, California: IEEE.
- Johns E, Leutenegger S, Davison AJ. Deep learning a grasp function for grasping under gripper pose uncertainty. In: Intelligent robots and systems (IROS), 2016 IEEE/rsj international conference on Intelligent Robots and Systems (IROS), 9 October 2016, pp. 4461–4468. IEEE.
- Kumra S, Kanan C.Robotic grasp detection using deep convolutional neural networks. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), Vancouver, BC, Canada, 24–28 Septeber 2017, pp. 769–776. IEEE.

Video link :

https://1drv.ms/v/s!AmWJSiSlv6W3qjCNW-0MG0BzLhjH?e=XUkmbO