Grasping Object using Robot Arm

Vaibhav Tapadiya

WSU ID - J776T432 Department of Computer Science

Wichita State University

Abstract – In today's world robot play's a very crucial role in this world they are used everywhere to solve the problem in medical, technology, car manufacturing etc. In this project we are going to create a robotic arm which can lift the given object for 5 seconds and put it back from it lifted.

Introduction

In this article, a novel, efficient grasp synthesis method is introduced that can be used for closed-loop robotic grasping. Using only a single monocular camera, the proposed approach can detect contour information from an image in real-time and then determine the precise position of an object to be grasped by matching its contour with a given template. This approach is much lighter than the currently prevailing methods, especially vision-based deep-learning techniques, in that it requires no prior training. With the use of the state-of-the-art techniques of edge detection, super pixel segmentation, and shape matching, our visual serving method does not rely on accurate camera calibration or position control and is able to adapt to dynamic environments. Experiments show that the approach provides high levels of compliance, performance, and robustness under diverse experiment environments.

Summary of the project

The main objective of this project is to design an application for robotics in such a way that the robot can detect the given object by using its co-ordinate and real-time image data and move towards it and then it will grasp that object using it's artificial hand and lift it for over 5 seconds and then place that object again to the co-ordinates from where he picked. We use TensorFlow and pybullet to implement it and we pass various images data to our algorithm to train it to detect the object.

Project Description

To implement this project we need various different environments first i install python 3.0 version and PIP to get all its related dependencies. Then we install pybullet to get virtual environments for our robotic arm and TensorFlow to train and test our data objects.
To install TensorFlow we need to first install visual studio which is used to detect object place around the robotic arm. Object Detection is the phenomenon in which the various objects such as knife , scissor , wrench etc to detect it we use Object detection Algorithm

from tensorflow libraries. The different angels and background and saved.
We are given 3-D representation of objects and our first task is to construct dataset of the given 3D images which has RGB-D values and this images are further converted in numpy array to create a dataset and then those are divided into training and testing data set. We then label each and every images of the data set by using label-Img tool. Then in the last we store all the values in the form of XML.

This XML are then furter converted to CSV files and loaded in the system for training and testing purpose. A label-map and a training configuration file is created. A label map is used to match the ID to the name, i.e. ID number of each of the item should match the ID that is specified in the TRF records.

A faster RCNN model is then implemented to get the configuration of training configuration file. After the model is trained an inference graph is generated and this inference graph is used to run the training model.
Then we find the co-ordinates of the object where it is placed on the table and then they are passed as inputs to the robot arm to detect object and lift it.

Discussion of the Results

we wanted our robot was able to detect the object and lift it for 5 seconds. The training dataset was able to recognize the object based on their shape and size also it calculated its coordinate data such as X-axis, Y-axis, -axis which help the robotic arm to find and locate the object and then it moves its arm towards the location specified and tries to touch the object and finally once got touch lifts the object .

images of object are captured from

we wanted our robot was able to detect the object and lift it for 5 seconds. The training dataset was able to recognized the object based on their shape and size also it calculated it's coordinate data such as X-axis,Y-axis,Z-axis which help the robotic arm to find and locate the object and then it moves it's arm towards the location specified and tries to touch the object and finally once got touch lifts the object .

Conclusion

The main goal of this project was to detect the object using object detection technique of tensorflow and train the robotic arm to grasp it from where it find the location of that object. We train all possible images to our program to find and locate the object using co-ordinate method and in the end we tested with some random images which give correct results.

References

Deep Learning Book by Youshua Bengio and Aaron Courville

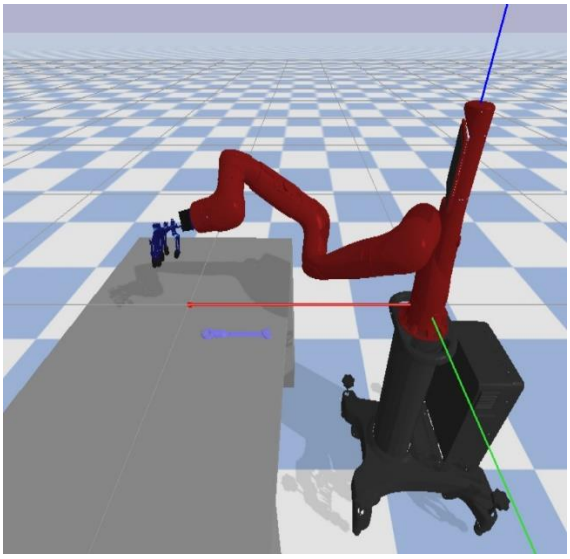
RGB-D Object recognition and Grasp Detection Using Hierarchical Cascaded Forests by Asif, Umar;

http://neuralnetworksanddeeplearning.com/

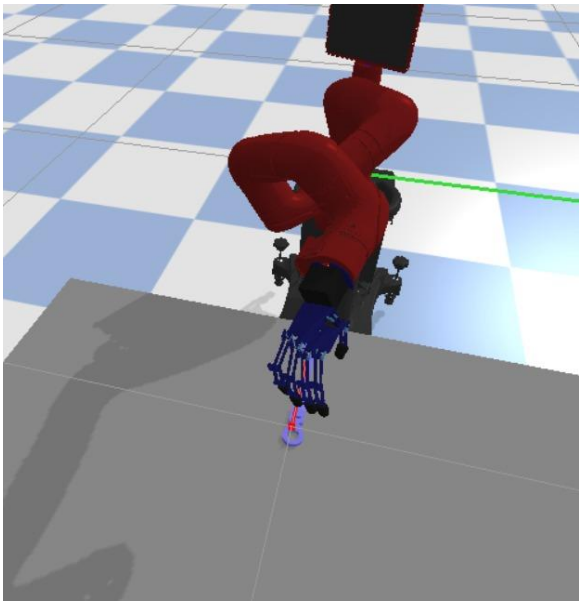
Deep Networks: Modern Practices

https://github.com/tensorflow/models/tree/master/research/object_detection

https://towardsdatascience.com/creating-your-own-object-detector-ad69dda69c85 by gilbert tanner



Object and robot arm loaded in system



Robot arm grasping the object

```
import pybullet as p
import time
import math
import random
from numpy.linalg import inv
import numpy as np
import pybullet_data
from datetime import datetime
import numpy
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import matplotlib
import matplotlib.pyplot as plt
from PIL import Image
from object_detection.utils import ops as utils_ops
from utils import label_map_util
from utils import visualization_utils as vsi_util




# Options and parameters


#  You need to update date path
sawyer_robot_path = 'data/sawyer_robot/sawyer_description/urdf/sawyer.urdf'
table_path = 'data/table/table.urdf'
```

```
plane_path = 'data/plane/plane.urdf'



# 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


# Camera
```

```
width = 256

height = 256

fov = 60

aspect = width / height

near = 0.02

far = 1



# 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_path, [0, 0, -1])


p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 0)

sawyerId = p.loadURDF(sawyer_robot_path, [0, 0, 0],

                [0, 0, 0, 3])  # load sawyer robot

tableId = p.loadURDF(table_path, [1.4, 0, -1], p.getQuaternionFromEuler([0, 0, 1.56]))  #
load table
```

```
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



# Set Camera##


p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 1)

p.resetBasePositionAndOrientation(sawyerId, [0, 0, 0], [0, 0, 0, 1])


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


# 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([1.05,-0.05,0.4], [1.1, 0, 0], [1, 0, 0])
```

```
projection_matrix = p.computeProjectionMatrixFOV(fov, aspect, near, far)
# print("projection_matrix",projection_matrix)


######  Inverse Kinematics Function: You don't need to change anything here.


# 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(bodyIndex=sawyerId,
                              jointIndex=i,
                              controlMode=p.POSITION_CONTROL,
```

```
                    targetPosition=jointP[i],
                    targetVelocity=0,
                    force=500,
                    positionGain=0.03,
                    velocityGain=1)
    return jointP

# control the lower joint and middle joint of pinky finger, range both [0.17 - 1.57]
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])


# 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(bodyIndex=sawyerId,
                            jointIndex=i,
                            controlMode=p.POSITION_CONTROL,
                            targetPosition=jointP[i],
                            targetVelocity=0,
                            force=500,
                            positionGain=0.03,
                            velocityGain=1)




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

# Task 1: Load your own objects designed by the instructors. Change the directory to the
file in the directory.


# 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


# object path
```

```
object_path =
'C:/Users/vaibh/Desktop/project_codes22/project_codes/data/test_object/000.urdf'  #
change file name here.

# texture path

texture_path = 'data/dtd/images/banded/banded_0002.jpg' # DON'T change the texture.


# random object position

objectId = p.loadURDF(object_path, [1.1, 0, 0], p.getQuaternionFromEuler([0, 0, 1.56]))

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









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

# Task 2: Write the program to detect and locate the target object.


# 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.

# time.sleep(1)


y = depth_opengl

img_depth = (65535*((y - y.min())/y.ptp())).astype(np.uint16)


y = rgb_opengl

img_rgb = (65535 * ((y - y.min()) / y.ptp()))



# model name of the recognition

MODEL_NAME = "inference_graph"

PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'

PATH_TO_LABELS = 'training/labelmap.phtxt'


############################################################Import the
trained recognition
model################################################

##=============================================================
============================================================

MODEL_NAME = 'inference_graph'

PATH_TO_FROZEN_GRAPH = 'object_detection/'+MODEL_NAME +
'/frozen_inference_graph.pb'

PATH_TO_LABELS = 'training/labelmap.pbtxt'

#=============================================================
============================================================


detection_graph = tf.Graph()
```

```
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    # tf.import_graph_def(od_graph_def, name='')



category_index =
label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)


def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)


##=================================================================
==========================================================


PATH_TO_TEST_DEPTH_DIR = 'test_images/image1.jpg'

##=================================================================
==========================================================

# Size, in inches, of the output images.



IMAGE_SIZE = (12, 8)
```

```
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinates and
fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1, -1])
```

```
      detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
          detection_masks, detection_boxes, image.shape[0], image.shape[1])
      detection_masks_reframed = tf.cast(
          tf.greater(detection_masks_reframed, 0.5), tf.uint8)
      # Follow the convention by adding back the batch dimension
      tensor_dict['detection_masks'] = tf.expand_dims(
          detection_masks_reframed, 0)
    image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

    # Run inference
    output_dict = sess.run(tensor_dict,
                           feed_dict={image_tensor: np.expand_dims(image, 0)})

    # all outputs are float32 numpy arrays, so convert types as appropriate
    output_dict['num_detections'] = int(output_dict['num_detections'][0])
    output_dict['detection_classes'] = output_dict[
        'detection_classes'][0].astype(np.uint8)
    output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
    output_dict['detection_scores'] = output_dict['detection_scores'][0]
    if 'detection_masks' in output_dict:
      output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict


image = Image.open(PATH_TO_TEST_DEPTH_DIR)
# the array based representation of the image will be used later in order to prepare the
```

```
# result image with boxes and labels on it.
image_np = load_image_into_numpy_array(image)
# Expand dimensions since the model expects images to have shape: [1, None, None, 3]
image_np_expanded = np.expand_dims(image_np, axis=0)
# Actual detection.
output_dict = run_inference_for_single_image(image_np, detection_graph)
# Visualization of the results of a detection.
vis_util.visualize_boxes_and_labels_on_image_array(
  image_np,
  output_dict['detection_boxes'],
  output_dict['detection_classes'],
  output_dict['detection_scores'],
  category_index,
  instance_masks=output_dict.get('detection_masks'),
  use_normalized_coordinates=True,
  line_thickness=3)
#line thickness was 8
plt.figure(figsize=IMAGE_SIZE)
plt.imshow(image_np)
plt.show()



boxes = output_dict['detection_boxes']
# get all boxes from an array
max_boxes_to_draw = boxes.shape[0]
# get scores to get a threshold
```

```
scores = output_dict['detection_scores']

# this is set as a default but feel free to adjust it to your needs

min_score_thresh = .5

# iterate over all objects found

for jjj in range(min(max_boxes_to_draw, boxes.shape[0])):

  #

  if scores is None or scores[jjj] > min_score_thresh:

     #the number in the next if statement tells which object you wanna get cordinate of; in this
case 2 asin obj 2

     if output_dict['detection_classes'][jjj] == target_object:

        cordinates = 256 * boxes[jjj]

        print('xmin:' + str(cordinates[0]))

        print('ymin:' + str(cordinates[1]))

        print('xmax:' + str(cordinates[2]))

        print('ymax:' + str(cordinates[3]))


xmin = cordinates[0]

ymin = cordinates[1]

xmax = cordinates[2]

ymax = cordinates[3]


print(xmin,ymin,xmax,ymax)

# return u,v

def locate_object(img_depth,img_rgb,view_matrix,projection_matrix):

    pass
```

```
# return d
def get_depth(img_depth,img_rgb,view_matrix,projection_matrix):
    pass


def get_coord(u,v,d):
    """
    xmax, xmin, ymax, and ymin you sould be able to from the detected depth region of the object in the scene.
    u = (xmax-xmin)/2
    y = (ymax-ymin)/2
    :param u: image coordination of the point of the object in x-axis
    :param v: image coordination of the point of the object in y-axis
    :param d: depth information of the point
    :return: world coordinate of the object.
    """
    # K is the intrinsic matrix
    fov = 60
    fx = 1/(math.tan(60/2))
    fy = 1/(math.tan(60/2))
    px = 0
    py = 0
    K = [[fx,0,px],[0,600.688,py],[0,0,1]]
    C = [u, v, d]
    cam = [C[0]*C[2],C[1]*C[2],C[2]]
    kinv = inv(K)
    coordinate = np.matmul(kinv,cam)
```

return coordinate

u,v = locate_object(img_depth,img_rgb,view_matrix,projection_matrix)

d = get_depth(img_depth,img_rgb,view_matrix,projection_matrix)

palmP_coordination = get_coord(u,v,d)

[1.15, -0.15, 0.15] # you need to update the palmP_coordination from depth region from the recognized objects

##

Task 3: Adjust the parameters for a successful grasp.

Run the simulation to grasp the object. Please record a video.

postions and orientations of hand joints, get from project_joint_control.py

thumbP = [1.0558935917169188, 0.027851175810643107, 0.1073322098156722]

thumbOrien = [0.39812982280279957, 0.690809337445068, -0.395742134500974, 0.45570085195699445]

indexP = [1.0795482197897535, 0.05518673811763744, 0.11565267334129664]

indexOrien = [0.31200454208076017, 0.6307833715960751, -0.316721080466879, 0.6359663992906824]

midP = [1.1753641477801184, -0.00021219911214629197, 0.09550815888884753]

midOrien = [0.6837067170243077, 0.17836048088911896, -0.6829256500902726, 0.18532463517197023]

ringP = [1.1650410995377067, 0.0007302504476453704, 0.0737612239747884]

```
ringOrien = [0.681825980091062, 0.185157084155091 88, -0.6811143508115239,
0.19212867609662887]

pinkyP = [1.1548277580687578, 0.0011463710029330151, 0.052919588278547966]

pinkyOrien = [0.6797792129433045, 0.1922707302814458, -0.6791405384537569,
0.19924938382831286]



currentP = [0] * 65

k = 0

while 1:

    k = k + 1


    # move palm to target postion

    i = 0

    while 1:

        # initial hand position

        i += 1

        p.stepSimulation()

        currentP = palmP(palmP_coordination, p.getQuaternionFromEuler([0, -math.pi * 1.5,
0]))

        time.sleep(0.03)

        if (i == 150):

            break


    i = 0

    while 1:

        # second hand position
```

```
        i += 1
        p.stepSimulation()
        currentP = palmP([1.1, -0.1, 0.05], p.getQuaternionFromEuler([0, -math.pi * 1.5, 0]))
        time.sleep(0.03)
        if (i == 80):
            break


    i = 0
    while 1:
        # third hand position
        i += 1
        p.stepSimulation()
        currentP = palmP([1.05, -0.025, 0.055], p.getQuaternionFromEuler([0, -math.pi * 1.5,
0]))
        time.sleep(0.05)
        if (i == 50):
            break


    # i = 0
    # while 1:
    #     i += 1
    #     p.stepSimulation()
    #     currentP = palmP([1.15, -0.05, -0.15],
    #                      p.getQuaternionFromEuler([-1.5708068687188497, -0.1745263268106108, -
1.5707944404751424]))
    #     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]))
        time.sleep(0.03)
        if(i==5000):
            break
    break


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