第二题
第一个终端运行
source ~/tensorflow/bin/activate
roslaunch tensorflow_object_detector usb_cam_detector.launch
第二个终端运行
roslaunch mbot_gazebo view_mbot_gazebo_play_ground.launch
第三个终端运行
rosrun test2 test2

思路
在 tensorflow_object_detector 中的 detect_ros.py 中，发布“pose”话题，主要包括杯子的 size_x 横长度，size_y 纵长度，以及杯子中心横坐标 pose_x
在 test2 中的 test2.cpp 中，订阅“pose”话题，接受到数据，进入回调函数
如果 size_x 横长度，size_y 纵长度都增大则发布前进命令，发布“cmd_vel”话题，让机器人前进；
如果 size_x 横长度，size_y 纵长度都减小则发布后退命令，发布“cmd_vel”话题，让机器人后退；
如果 pose_x 中心点横坐标减小则发布左转命令，发布“cmd_vel”话题，让机器人左转；
如果 pose_x 中心点横坐标增大则发布右转命令，发布“cmd_vel”话题，让机器人右转

代码如下：

detect_ros.py

```
#!/usr/bin/env python
## Author: Rohit
## Date: July, 25, 2017
# Purpose: Ros node to detect objects using tensorflow

import os
import sys
import cv2
import numpy as np
try:
  import tensorflow as tf
except ImportError:
  print("unable to import TensorFlow. Is it installed?")
  print("  sudo apt install python-pip")
  print("  sudo pip install tensorflow")
  sys.exit(1)

# ROS related imports
import rospy
from std_msgs.msg import String , Header
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from vision_msgs.msg import Detection2D, Detection2DArray, ObjectHypothesisWithPose

# Object detection module imports
import object_detection
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
```

```
from test2.msg import pose

# SET FRACTION OF GPU YOU WANT TO USE HERE
GPU_FRACTION = 0.4

######### Set model here ############
MODEL_NAME =  'ssd_mobilenet_v1_coco_11_06_2017'
# By default models are stored in data/models/
MODEL_PATH = os.path.join(os.path.dirname(sys.path[0]),'data','models' , MODEL_NAME)
# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_PATH + '/frozen_inference_graph.pb'
######### Set the label map file here ###########
LABEL_NAME = 'mscoco_label_map.pbtxt'
# By default label maps are stored in data/labels/
PATH_TO_LABELS = os.path.join(os.path.dirname(sys.path[0]),'data','labels', LABEL_NAME)
######### Set the number of classes here #########
NUM_CLASSES = 90

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

## Loading label map
# Label maps map indices to category names, so that when our convolution network predicts `5`,
# we know that this corresponds to `airplane`.  Here we use internal utility functions,
# but anything that returns a dictionary mapping integers to appropriate string labels would be fine
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# Setting the GPU options to use fraction of gpu that has been set
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = GPU_FRACTION

# Detection

class Detector:

  def __init__(self):
    self.image_pub = rospy.Publisher("debug_image",Image, queue_size=1)
    self.object_pub = rospy.Publisher("objects", Detection2DArray, queue_size=1)
    self.bridge = CvBridge()
    self.image_sub = rospy.Subscriber("image", Image, self.image_cb, queue_size=1,
buff_size=2**24)
    self.sess = tf.Session(graph=detection_graph,config=config)
```

```
    self.pose_pub = rospy.Publisher("pose", pose, queue_size=1)
    self.pose = pose()

  def image_cb(self, data):
    objArray = Detection2DArray()
    try:
      cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
      print(e)
    image=cv2.cvtColor(cv_image,cv2.COLOR_BGR2RGB)

    # 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 = np.asarray(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)
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    # Each box represents a part of the image where a particular object was detected.
    boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    # Each score represent how level of confidence for each of the objects.
    # Score is shown on the result image, together with the class label.
    scores = detection_graph.get_tensor_by_name('detection_scores:0')
    classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')

    (boxes, scores, classes, num_detections) = self.sess.run([boxes, scores, classes,
num_detections],
      feed_dict={image_tensor: image_np_expanded})

    objects=vis_util.visualize_boxes_and_labels_on_image_array(
      image,
      np.squeeze(boxes),
      np.squeeze(classes).astype(np.int32),
      np.squeeze(scores),
      category_index,
      use_normalized_coordinates=True,
      line_thickness=2)

    objArray.detections =[]
    objArray.header=data.header
    object_count=1

    for i in range(len(objects)):
      object_count+=1
      objArray.detections.append(self.object_predict(objects[i],data.header,image_np,cv_image))

    self.object_pub.publish(objArray)
```

```
        if self.pose.id == "47":
            self.pose_pub.publish(self.pose)

        img=cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB)
        image_out = Image()
        try:
            image_out = self.bridge.cv2_to_imgmsg(img,"bgr8")
        except CvBridgeError as e:
            print(e)
        image_out.header = data.header
        self.image_pub.publish(image_out)

    def object_predict(self,object_data, header, image_np,image):
        image_height,image_width,channels = image.shape
        obj=Detection2D()
        obj_hypothesis= ObjectHypothesisWithPose()

        object_id=object_data[0]
        object_score=object_data[1]
        dimensions=object_data[2]

        obj.header=header
        obj_hypothesis.id = str(object_id)
        obj_hypothesis.score = object_score
        obj.results.append(obj_hypothesis)
        obj.bbox.size_y = int((dimensions[2]-dimensions[0])*image_height)
        obj.bbox.size_x = int((dimensions[3]-dimensions[1] )*image_width)
        obj.bbox.center.x = int((dimensions[1] + dimensions [3])*image_height/2)
        obj.bbox.center.y = int((dimensions[0] + dimensions[2])*image_width/2)

        self.pose.pose_x = int((dimensions[1] + dimensions [3])*image_height/2)
        self.pose.pose_y = int((dimensions[0] + dimensions[2])*image_width/2)
        self.pose.size_x = int((dimensions[3]-dimensions[1] )*image_width)
        self.pose.size_y = int((dimensions[2]-dimensions[0])*image_height)
        self.pose.id = str(object_id)

        return obj

def main(args):
    rospy.init_node('detector_node')
    obj=Detector()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print("ShutDown")
    cv2.destroyAllWindows()
```

```
if __name__=='__main__':
   main(sys.argv)
```

test2.cpp

```
#include "ros/ros.h"
#include "std_msgs/Int64.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
#include "test2/pose.h"

int flag = 0;
std_msgs::Int64 pose_width, pose_high, pose_x;

class SubscribeAndPublish
{
public:
 SubscribeAndPublish()
 {
  pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);
  sub = n.subscribe("pose", 1, &SubscribeAndPublish::callback, this);
 }

 void callback(const test2::pose::ConstPtr& pose)
 {
      if(flag==0){
        pose_width.data = pose->size_x;
        pose_high.data = pose->size_y;
        pose_x.data = pose->pose_x;
        flag = 1;
      }
      else{
        if((pose->size_x>pose_width.data+20)&&(pose->size_y>pose_high.data+20)){
          ROS_INFO("go forward");
          pose_width.data = pose->size_x;
          pose_high.data = pose->size_y;
          pose_x.data = pose->pose_x;
          geometry_msgs::Twist msg;
          msg.linear.x = 0.3;
          msg.angular.z = 0;
          pub.publish(msg);
        }
        else if((pose->size_x<pose_width.data-20)&&(pose->size_y<pose_high.data-20)){
          ROS_INFO("go back");
          pose_width.data = pose->size_x;
          pose_high.data = pose->size_y;
          pose_x.data = pose->pose_x;
          geometry_msgs::Twist msg;
          msg.linear.x = -0.3;
```

```
            msg.angular.z = 0;
            pub.publish(msg);
          }
          else if( (pose->pose_x + pose->size_x/2) < (pose_x.data + pose_width.data/2 - 30) ){
            ROS_INFO("turn left");
            pose_width.data = pose->size_x;
            pose_high.data = pose->size_y;
            pose_x.data = pose->pose_x;
            geometry_msgs::Twist msg;
            msg.linear.x = 0;
            msg.angular.z = 0.3;
            pub.publish(msg);
          }
          else if( (pose->pose_x + pose->size_x/2) > (pose_x.data + pose_width.data/2 + 30) ){
            ROS_INFO("turn right");
            pose_width.data = pose->size_x;
            pose_high.data = pose->size_y;
            pose_x.data = pose->pose_x;
            geometry_msgs::Twist msg;
            msg.linear.x = 0;
            msg.angular.z = -0.3;
            pub.publish(msg);
          }
        }
    }

private:
  ros::NodeHandle n;
  ros::Publisher pub;
  ros::Subscriber sub;

};

int main(int argc, char** argv){
    ros::init(argc, argv, "test2");
    SubscribeAndPublish SAPObject;
    ros::spin();

    return 0;
}
```