Tópicos especiais em Inteligência Computacional IV: Sistemas Operacionais de Robôs Inteligentes
Project course:

Rayanne Souza
gsouzarayanne@gmail.com

Franklin Fernández
fracarfer5@aluno.puc-rio.br

December 20, 2018

1 Introduction

This work aims at using ROS to control a robotic arm in order to perform pick and place tasks. In our proposed model, the robot can pick and place different objects of diverse shapes. This was achieved by applying a deep learning algorithm to recognize the target objects. Given the name of the target object, the network must be able to recognize it and then return the bounding box fitting the object. From the acquired information concerning the bounding box of the target, we are able to define the position and the orientation of the object and deliver this to the robotic arm that will have enough intelligence to identify the target's placement.

2 System configuration

2.1 Requirements

Before starting work in the project described in the last section we need to define our hardware and software requirements:

Hardware:

1. Host Computer with nvidia-docker
2. Nvidia Jetson TX2 Board
3. Phantom Arm
4. Base for phantom arm
5. Objects to detection (objects into the list of classes of the neural network)
6. Periphelicals (screen, keyboard, mouse)

Software:

1. Ubuntu 16.04 LTS for Jetson Board
2. Jetpack 3.x or higher
3. Python 2.7
4. Python 3.5
5. ROS Kinetic
6. Tensorflow

Firstly, we need to install all the drivers and machine learning frameworks flashing the Jetson board with the JetPack package in order to be able to use Deep Neural Netwporks (DNN) with it. To do this, we can flash it by two ways, NVIDIA GPU Cloud or nvidia-docker, in this case we are using a Computer host with nvidia-docker running, the steps to flash the Jetson board and install DIGITS server can be found in [1].

After install all the deep learning dependences into the Jetson we can test writing the next commands in the terminal:

```
$ cd jetson-inference/build/aarch64/bin
$ ./imagenet-console orange_0.jpg output_0.jpg
```

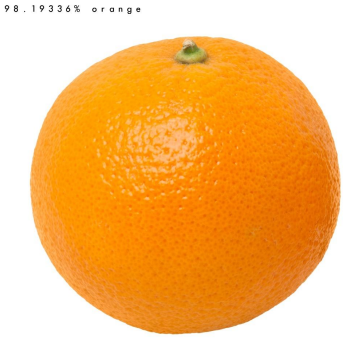

Figure 1: Output image for inference with Imagenet

or

```
$ ./imagenet-console granny_smith_1.jpg output_1.jpg
```

Figure 2: Output image for inference with Imagenet

Next, we need to install all the dependences and configure the object detection neural network. To do this, we can train an object detection DNN from scratch, creating or using an existent dataset, or using a pre-trained object detection DNN. In this case, for time reasons, we are using the Tensorflow Opject Detection API which contains the COCONET, which is a pre-trained object detection DNN which use the Microsoft Common Objects in COntext (MS COCO) dataset and have 91 common objects categories. The steps to install and configure the Tensorflow API can be found in [2].

After install all the Tensorflow Object Detection API into the Jetson we can test it by running the next command in the terminal:

```
$ cd /home/nvidia/tensorflow/models/research/...
    models/research/object_detection
$ python object_detection/builders/model_builder_test.py
```

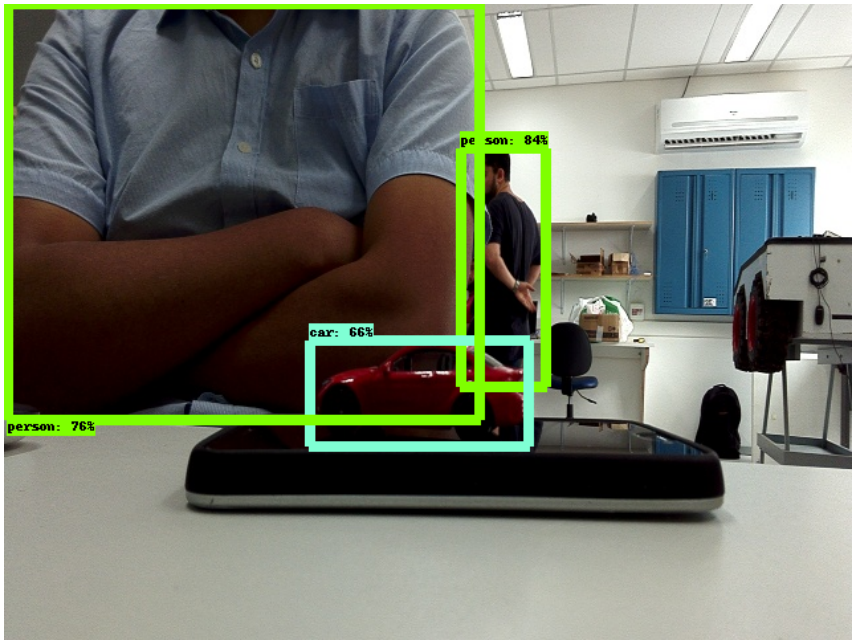

Figure 3: Output image for detection with the Tensorflow DNN

So that, with all the dependences installed we were be able to perform the necesary modifications in order to perform the object localization with the CSI onboard Camera of the Jetson using DNN.

After that, the next step was install ROS Kinetic with all its dependences in the Jetson, to perform this we followed the installation steps of [3]. Finally, with all the programs installed the Jetson was ready to be programmed for our task.

2.2 Camera calibration

The Jetson's camera was calibrated following the monocular camera calibration guidelines found in [4]. After the compiling step, we ran the Calibration Node using the parameters of our checkerboard 15x9 grid.

```
$ rosrun camera\_calibration cameracalibrator.py --size 15x9 --square
    0.0176 image:=/camera/image_raw camera:=/camera
```

The camera calibration parameters are detailed in the Appendix A.

3 System description

3.1 System model

The Jetson board is used for processing the object detection task and for controlling the arm movement. We use the Jetson's camera in this detection. Since the Phantom arm has a short motion range, the best position of the Jetson's camera in respect to the arm is under the Phantom arm. Therefore, we have built a base using MDF wood to serve as support (Figures 4 and 5).

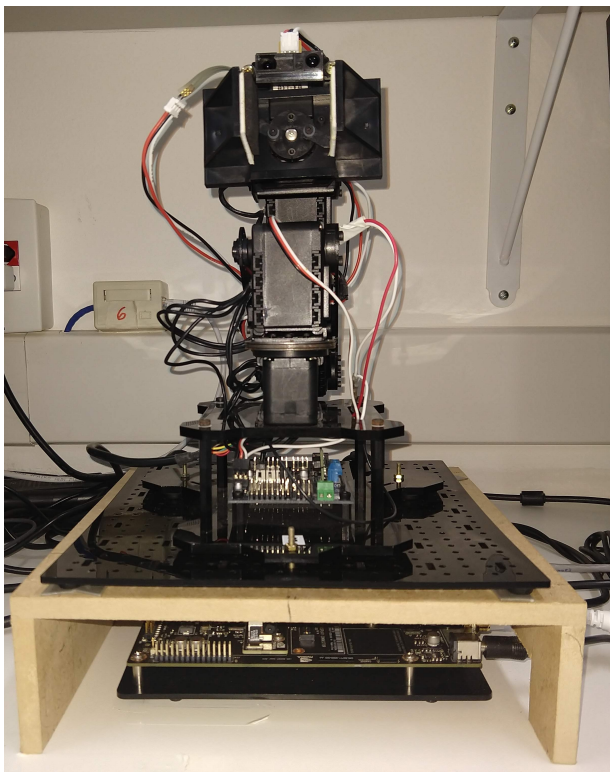

Figure 4: Front view of the arm

Figure 5: Lateral view of the arm

The gripper closing needs to be controlled in order to adapt itself to the different types of objects. For this reason, we put a force sensor in the gripper (Figure 6).

The analog sensor's output shown in Figure 7 was connected to pin A02 of the Phantom board. In order to get this value from a topic, we have changed the file `phantomx_arm.yaml` by adding the sensor to pinout 2.

```
analog_sensors: {
    distance: { pin: 0 },
    force: { pin: 2 }
}
```

To add the Jetson's camera into the robot's description file, we have changed the `phanthon_arm.xacro` by adding the camera link with fixed joint since the camera does not move.

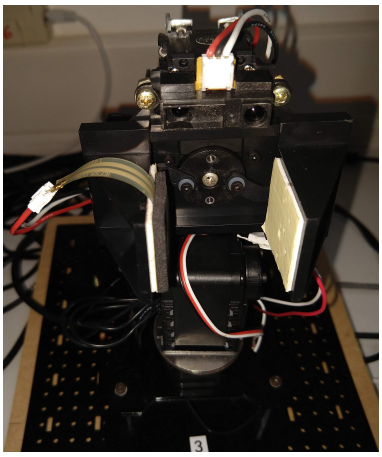

Figure 6: Force sensor on gripper

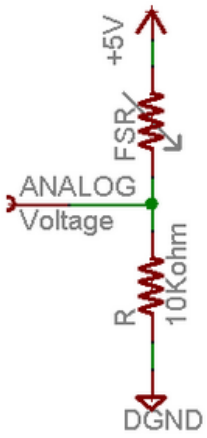

Figure 7: Electrical schematic of the force sensor

```
<link name="camera_link"/>
    <joint name="camera_joint" type="fixed">
      <origin xyz="0.01 0 -0.04" rpy="-1.5707 0 -1.5707"/>
      <parent link="base_link"/>
      <child link="camera_link"/>
    </joint>
```

3.2 Environment

The workspace of a Phanthom arm needs to be limited because of the existence of objects like computers, walls and floating shelves around the arm. The Figure 8 shows the solid objects added to the planning scene to avoid collisions with objects placed in the environment.

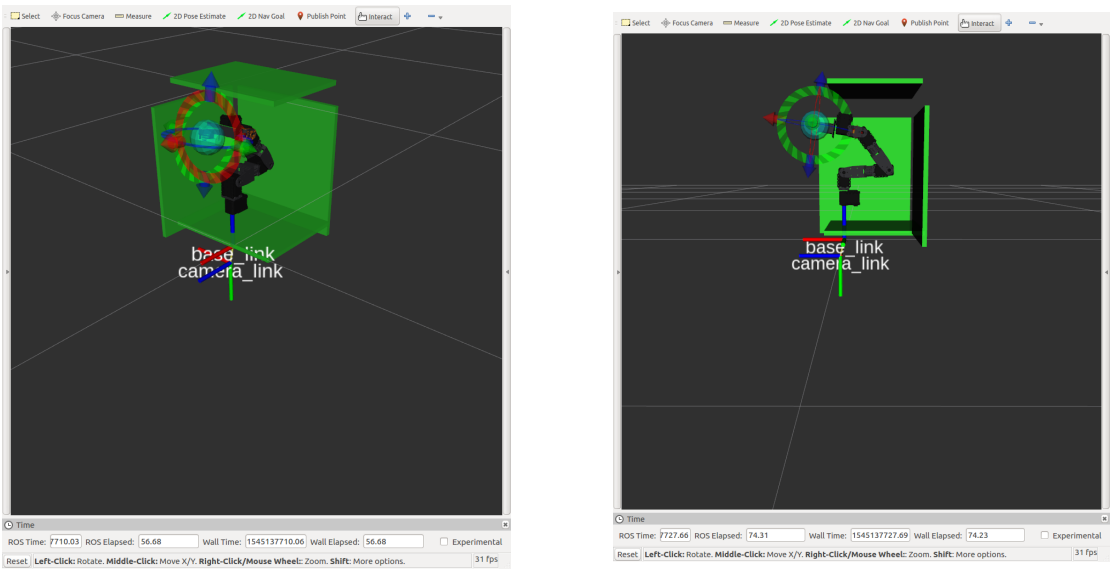

Figure 8: Solid objects added to the planning scene

3.3 The pick and place task

Four steps are needed to initialize the pick and place task, namely (i) running the `object_detection.py` using Python 3.5; (ii) running the `camera.launch` file; (iii) running the `planning_demo.launch` file; (iv) running the `pick_place.py` using Python 2.7.

In the object_detection.py (Appendix B), the object detection is executed and the bounds of the found object are written to the file entitled `data.fcf`.

The `camera.launch` file launches the `camera_info_publisher` node to publish the camera calibration data. This data is necessary to project the pixel from `data.fcf` to a 3d ray (line 166 in the Appendix C) for allowing the computation of the object position (line 183 in the Appendix C.

The `planning_demo.launch` file launches the Rviz and nodes associated with the arm controller driver.

In the `pick_place.py` file, we execute the pick and place task. The Appendix C shows the code used for this. The node `pick_place` is subscribed to the topics `/arbotix/force` and `/gpg/camera_info`, respectively to receive the force sensor output and the camera calibration data (lines 122-123). The force control routine (lines 98 - 109) controls the gripper closing. After several tests we defined that 100 for the sensor output value is adequate to avoid damage to the gripper and to the object. The node `pick_place` publishes in the topic `/gripper_joint/command` to control the gripper closing (line 124).

The pick and place routine executes the following steps when given a target (line 150):

i Puts the arm in the initial position (line 147).

ii Reads the bounds of the object from the `data.fcf` file (line 159).

iii Projects the midpoint of the lower horizontal segment to a 3d ray (line 166).

iv Converts the 3d ray from `camera_link` to `base_link` (lines 177 - 181).

v Computes the target position (line 183).

vi Computes the target orientation (lines 193 - 194).

vii Puts the arm in the target position (lines 227-229).

viii Gets the object controlling the gripper closing (line 231).

ix Puts the arm back to into the initial position (line 232).

x Pust the arm in the final position and drop the object to a box (line 233).

xi Puts the arm back in the initial position (line 236).

The initial position corresponds to the arm position shown in Figure 8. After getting the object, the arm comes back to the intial position before finally achieving the final position. This intermediate stage was included because when we send the object directly to the final position, problems involving planning or singularities often occurred.

In the next page, the Figure 9 details the description of all nodes used for the pick and place tasks.

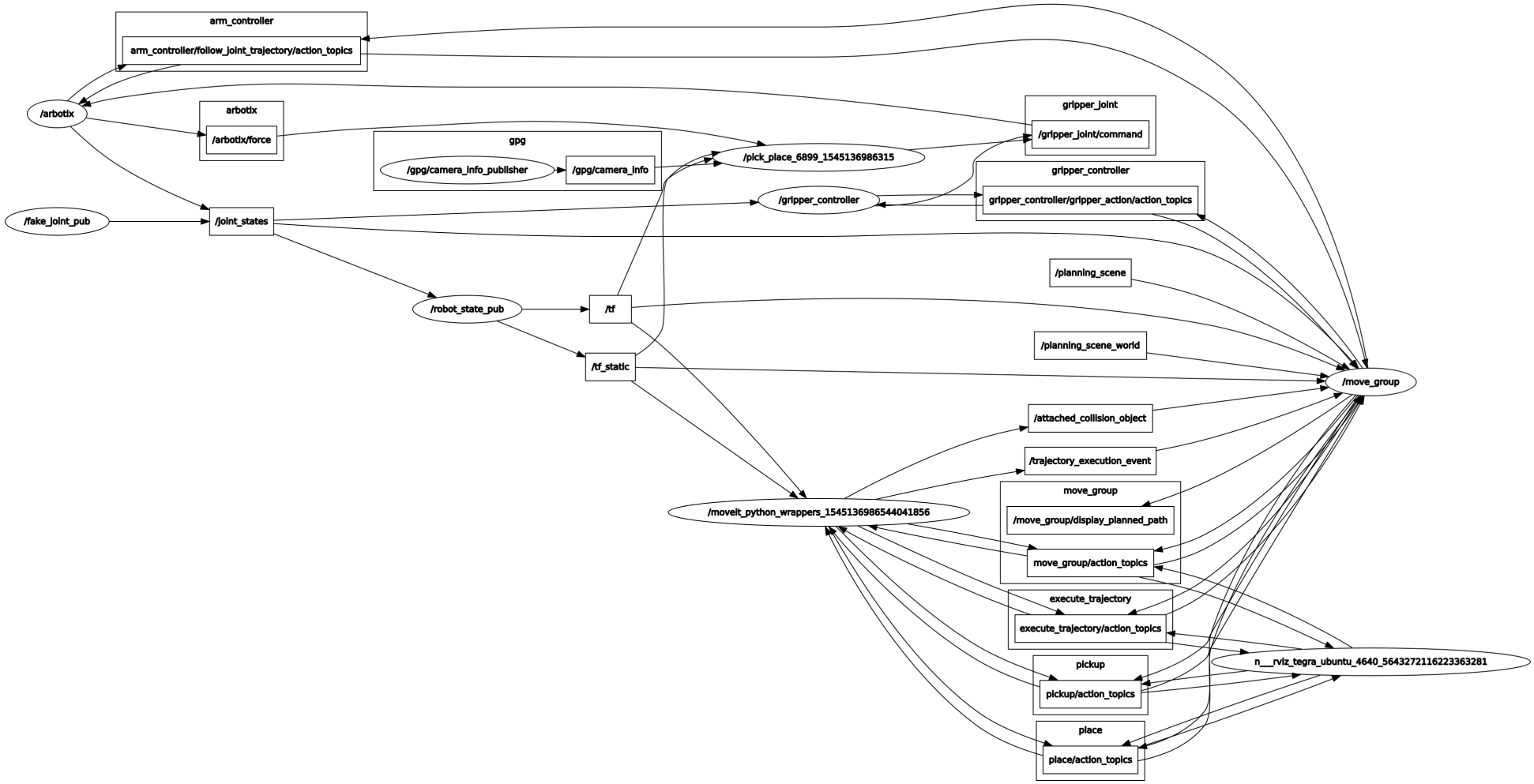

Figure 9: Description of all nodes used for the pick and place tasks

4 Performance

In general, the built project presents good performance since perform the task very well in most of the time. The pre-trained neural network is be able to recognize very well the objects of the class without have the entire image. Besides the phantom arm is be able to pick up the selected objects in the environment. In addition we can say that the accuracy of the system is 80%. This value corresponds to the mean between the hit rates for the car and bottle as reflected in Table 1. Note that the system can always pick up the car. One reason is that the car is less malleable than the bottle.

Object	**N hit**	**N failures**	**rate of hits**	**rate of failures**
Car	15	0	100%	0%
Bottle	9	6	60%	40%

Table 1: Failure and hit rates

5 Discussion

Although the system presents an aceptable performance we saw some troubles which we cannot ignore. Firstly, the neural network used is a pre-trained model with more classes that we need, so we have an heavy neural model that decrease the GPU performance presenting latency in the object detection task. On the other hand, some times the neural network is not be able to recognize some trivial objects like sport balloons or fruits or performs a bad object recognition.

Moreover, we had problems with the motion planning. The main reasons can be summarized as (i) the object is out of the arm range; (ii) the arm achieves singular configurations or (iii) the planning tries to go to the target position by rotating the arm 180°. To reduce this error, we added an upper cover and a wall in the right side of the Phantom, as shown in Figure 8, to reduce the rate of motion planning fails.

Another topic which we can consider as a inconvenient is the Jetson board setting up, since needs a lot of requirements to be configurated: a host computer with a good graphic card, an specific operating system, drivers and a a very large of steps in the configuration process. In addition, in order to perform the neural network training, we need and specific dataset to do this task, in the internet we can find a lot of webpages with interesting dataset for classification, recognition, detection, segmentation, etc, but the principal problem is that those datasets have a very high size of information, most of them considered as garbage. So, we need to be careful with the dataset choosing. For those reasons we considered to use pre-trained object detection neural networks in this project.

6 Constraints and Future work

The object detection is limited to means of transportation, fruits, animals, home appliances and sport balls. We have used a pre-trained network. An improvement could be done by training a new network to detect only the set of required objects or by using

transfer learning techniques in order to improve the accuracy of the network. However, it could reduce the latency on the detection and improve the recognition.

The Phanthom arm has small a range to pick an object. This was reduced even further because of the usage of a support base to place the Jetson under the Phanthom. The use of another camera connected to the Jetson by USB would avoid the need of a base and improve the arm's range.

Another constraint is the dependence of an intermediate stage before placing the arm in the final position. This decreases the time performance of the system. Therefore, a study on how to reduce planning scene erros and how to avoid singular configurations may eliminate this intermediate stage and consequently improve the time performance of the system.

A Calibration parameters in yaml file

```
image_width: 640
image_height: 480
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [472.067658, 0.000000, 323.735012, 0.000000,
         632.969353, 242.976503, 0.000000, 0.000000, 1.000000]

distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.103142, -0.167043, -0.001261, -0.000777, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1.000000, 0.000000, 0.000000, 0.000000, 1.000000,
         0.000000, 0.000000, 0.000000, 1.000000]
projection_matrix:
  rows: 3
  cols: 4
  data: [478.062714, 0.000000, 323.381312, 0.000000, 0.000000,
         643.103394, 242.538970, 0.000000, 0.000000, 0.000000,
         1.000000, 0.000000]
```

B File object_detection.py

```
#!/usr/bin/env python3
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
from pathlib import Path
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

sys.path.remove('/opt/ros/kinetic/lib/python2.7/dist-packages')
import cv2
import subprocess

width = 640
height = 480
WINDOW_NAME = 'Detection'

def open_camera(width, height):
 gst_str = ('nvcamerasrc ! '
                'video/x-raw(memory:NVMM), '
                'width=(int)2592, height=(int)1458, '
                'format=(string)I420, framerate=(fraction)30/1 ! '
                'nvvidconv ! '
                'video/x-raw, width=(int){}, height=(int){}, '
                'format=(string)BGRx ! '
                'videoconvert ! appsink').format(width, height)
 return cv2.VideoCapture(gst_str, cv2.CAP_GSTREAMER)

def open_window(width, height):
 cv2.namedWindow(WINDOW_NAME, cv2.WINDOW_NORMAL)
 cv2.resizeWindow(WINDOW_NAME, width, height)
 cv2.setWindowTitle(WINDOW_NAME, 'Camera Demo for Jetson TX2')


cap = open_camera(width, height)

sys.path.append("..")


# ## Object detection imports
from utils import label_map_util
from utils import visualization_utils as vis_util


# # Model preparation

MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')
```

```
57
58 NUM_CLASSES = 90
59
60 opener = urllib.request.URLopener()
61 opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
62 tar_file = tarfile.open(MODEL_FILE)
63 for file in tar_file.getmembers():
64   file_name = os.path.basename(file.name)
65   if 'frozen_inference_graph.pb' in file_name:
66     tar_file.extract(file, os.getcwd())
67
68
69 # ## Load a (frozen) Tensorflow model into memory.
70
71 detection_graph = tf.Graph()
72 with detection_graph.as_default():
73   od_graph_def = tf.GraphDef()
74   with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
75     serialized_graph = fid.read()
76     od_graph_def.ParseFromString(serialized_graph)
77     tf.import_graph_def(od_graph_def, name='')
78
79
80 # ## Loading label map
81
82 label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
83 categories = label_map_util.convert_label_map_to_categories(label_map,
       max_num_classes=NUM_CLASSES, use_display_name=True)
84 category_index = label_map_util.create_category_index(categories)
85
86 def load_image_into_numpy_array(image):
87   (im_width, im_height) = image.size
88   return np.array(image.getdata()).reshape(
89       (im_height, im_width, 3)).astype(np.uint8)
90
91
92 def main():
93  with detection_graph.as_default():
94    with tf.Session(graph=detection_graph) as sess:
95      while True:
96        ret, image_np = cap.read()
97        image_np_expanded = np.expand_dims(image_np, axis=0)
98        image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
99        boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
100       scores = detection_graph.get_tensor_by_name('detection_scores:0')
101       classes = detection_graph.get_tensor_by_name('detection_classes:0')
102       num_detections = detection_graph.get_tensor_by_name('num_detections
              :0')
103       # Read desired object
104       obj = Path('/home/nvidia/catkin_ws/src/project/src/object.fcf').
              read_text()
105       # Actual detection.
106       (boxes, scores, classes, num_detections) = sess.run(
107           [boxes, scores, classes, num_detections],
108           feed_dict={image_tensor: image_np_expanded})
109       # Visualization of the results of a detection.
110       image, bounds, ident = vis_util.
              visualize_boxes_and_labels_on_image_array(
```

```
111             image_np,
112             np.squeeze(boxes),
113             np.squeeze(classes).astype(np.int32),
114             np.squeeze(scores),
115             category_index,
116             use_normalized_coordinates=True,
117             line_thickness=8,
118             desired_class=obj)
119         print(bounds)
120         print(ident)
121         f= open("/home/nvidia/catkin_ws/src/project/src/data.fcf","w")
122         f.write(str(bounds[0])+","+str(bounds[1])+","+str(bounds[2])+","+str
                (bounds[3]))
123         f.close()
124 
125         cv2.imshow('object detection', image)
126         if cv2.waitKey(25) & 0xFF == ord('q'):
127           cv2.destroyAllWindows()
128           break
129 
130 # Main loop
131 if __name__ == '__main__':
132  try:
133   main()
134  except rospy.ROSInterruptException: pass
```

C Pick and place code

```
#!/usr/bin/env python

import moveit_commander
from geometry_msgs.msg import Pose, PointStamped, TransformStamped
from std_msgs.msg import Float64, Header
from arbotix_msgs.msg import Analog
from tf2_msgs.msg import TFMessage
from sensor_msgs.msg import CameraInfo
from tf.transformations import quaternion_from_euler
import time, rospy, image_geometry, numpy
import tf2_ros
import tf2_geometry_msgs
import tf,  math


# Global variables

global model

# Initial parameters

gripper = None
cxm = None
cym = None
cam_info = None

# Initialize camera

model = image_geometry.PinholeCameraModel()

# Initial position

pose_target = Pose()


# Callbacks and util functions
def callback_camera(data):
 global model
 model.fromCameraInfo(data)

def callback_force(msg):
 global force
 force = msg.value

def compute_position(p, q):
 z = -0.08
 var_lambda =(z - p.point.z)/(q.point.z-p.point.z)
 x = p.point.x+var_lambda*(q.point.x-p.point.x)
 y = p.point.y+var_lambda*(q.point.y-p.point.y)
 pos = [x,y,z]
 return pos

def callback_goal(bounds):
 global cxm
 global cym
 print bounds
```

```
 cxm = int((bounds[1] + bounds[3])*0.5)
 cym = int((bounds[0] + bounds[2])*0.5)
 print cxm, cym


def callback_gripper(msg):
 global gripper

 gripper = msg.transforms
 if(len(gripper) == 5):
  print(gripper)
# print(gripper)

def go_to_pinitial(move_group):
 pose_initial = Pose()
 pose_initial.position.x = 0.062
 pose_initial.position.y = 0.003
 pose_initial.position.z = 0.274
 pose_initial.orientation.x = 0.001
 pose_initial.orientation.y = -0.067
 pose_initial.orientation.z = 0.020
 pose_initial.orientation.w = 0.997
 move_group.set_pose_target(pose_initial)

 move_group.go(wait=True)
 time.sleep(0.5)

def go_to_pfinal(move_group):
 pose_initial = Pose()
 pose_initial.position.x = 0.300
 pose_initial.position.y = -0.069
 pose_initial.position.z = 0.110
 pose_initial.orientation.x = 0.023
 pose_initial.orientation.y = 0.207
 pose_initial.orientation.z = -0.110
 pose_initial.orientation.w = 0.972
 move_group.set_pose_target(pose_initial)

 move_group.go(wait=True)
 time.sleep(0.5)

def get_object():
 global pub
# Force control routine
 a = 0.5
 delta = 0.1
 while(force < 100) and not rospy.is_shutdown():
  if(a < 1.2):
   pub.publish(a)
   a = a + delta
   time.sleep(0.2)
  else:
   break


# ------------------Main function-------------------------|

def go_target():
```

```
115
116   global cxm
117   global cym
118   global pub
119
120 #Initialize node
121   rospy.init_node('pick_place', anonymous=True)
122   rospy.Subscriber("/arbotix/force", Analog, callback_force)
123   rospy.Subscriber("/gpg/camera_info", CameraInfo, callback_camera)
124   pub = rospy.Publisher("/gripper_joint/command", Float64, queue_size=5)
125
126   tfBuffer = tf2_ros.Buffer()
127   listener = tf2_ros.TransformListener(tfBuffer)
128   direction = PointStamped()
129   direction.header.stamp = rospy.Time()
130   direction.header.frame_id = "camera_link"
131
132   robot = moveit_commander.RobotCommander()
133   group = moveit_commander.MoveGroupCommander("arm")
134   group.set_goal_tolerance(0.01)
135   group.set_planner_id('RRTstarkConfigDefault')
136   group.set_planning_time(0.5)
137
138   r = rospy.Rate(1)
139
140   time.sleep(1)
141   pub.publish(0)
142   time.sleep(1)
143
144
145   while not rospy.is_shutdown():
146 # Go to initial position
147     go_to_pinitial(group)
148
149 #Wait for user input
150     obj = raw_input("Write the object that you want to pick up or write '
          quit' to exit: ")
151     if obj == 'quit':
152       break
153     print obj
154 #Read center of mass
155     f= open("/home/nvidia/catkin_ws/src/project/src/object.fcf","w")
156     f.write(obj)
157     f.close()
158     f = open("data.fcf",'r')
159     args = f.readline()
160     args = args.split(',')
161     args = numpy.array(args, dtype=numpy.float64)
162 #Calculate object position
163     cxm = int((args[1] + args[3])*0.5)
164     cym = int(args[2])
165     position = None
166     line = model.projectPixelTo3dRay((cxm,cym))
167
168     if cxm == 0 and cym == 0:
169       print "zero"
170       pass
171
```

```
172     direction.point.x = line[0]
173     direction.point.y = line[1]
174     direction.point.z = line[2]
175 
176     try:
177      q = tfBuffer.transform(direction, "base_link")
178      direction.point.x = 0
179      direction.point.y = 0
180      direction.point.z = 0
181      p = tfBuffer.transform(direction, "base_link")
182 
183      position = compute_position(p,q)
184      pose_target.position.x = position[0] + 0.01
185      pose_target.position.y = position[1]
186      if obj == 'bottle':
187       pose_target.position.z = position[2] + 0.065
188      else:
189       pose_target.position.z = position[2] + 0.03
190      if pose_target.position.z < -0.052:
191        pose_target.position.z = -0.052
192 
193      angle = math.atan2(pose_target.position.y,pose_target.position.x)
194      quart = tf.transformations.quaternion_from_euler(0, 1.5707, angle)
195 
196      pose_target.orientation.x = quart[0]
197      pose_target.orientation.y = quart[1]
198      pose_target.orientation.z = quart[2]
199      pose_target.orientation.w = quart[3]
200 
201     except (tf2_ros.LookupException, tf2_ros.ConnectivityException, tf2_ros.
           ExtrapolationException) as e:
202       print e
203       r.sleep()
204 
205 
206 
207 # Send the coordinates to Rviz
208     broadcaster = tf2_ros.TransformBroadcaster()
209     t = TransformStamped()
210     t.header.stamp = rospy.Time.now()
211     t.header.frame_id = "base_link"
212     t.child_frame_id = "object"
213     t.transform.translation.x = position[0]
214     t.transform.translation.y = position[1]
215     t.transform.translation.z = position[2]
216 
217     t.transform.rotation.x = 0
218     t.transform.rotation.y = 0
219     t.transform.rotation.z = 0
220     t.transform.rotation.w = 1
221     broadcaster.sendTransform(t)
222 
223 # Publish the object position
224 
225     print "pose_target"
226     print pose_target
227     group.set_pose_target(pose_target)
228 # Pick up the object and come back to initial position
```

```
229     group.go(wait=True)
230     time.sleep(1)
231     get_object()
232     go_to_pinitial(group)
233     go_to_pfinal(group)
234     pub.publish(0)
235     time.sleep(0.5)
236     go_to_pinitial(group)
237
238 # Main loop
239 if __name__ == '__main__':
240   try:
241     go_target()
242   except rospy.ROSInterruptException: pass
```

References

[1] https://github.com/dusty-nv/jetson-inference#system-setup, accessed: Nov-2018.

[2] "Tensorflow object detection api," https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md, accessed on Nov-2018.

[3] https://www.jetsonhacks.com/2017/03/27/robot-operating-system-ros-nvidia-jetson-tx2/, accessed: Nov-2018.

[4] "Monocular camera calibration guide," http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration, accessed on Nov-2018.