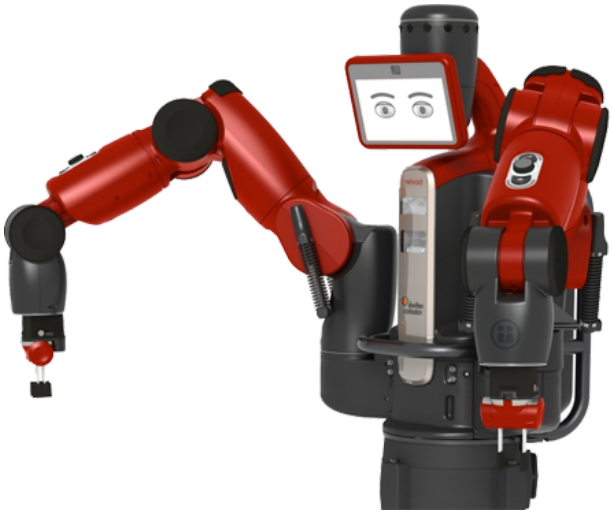
The Un-Paxter

Personal E-Commerce Assistant

https://github.com/fuentesori/HumanoidRobotics2

Philippe Wyder
pmw2125@columbia.edu
github.com/GitWyd

Oriana Fuentes
oif2102@columbia.edu
github.com/fuentesori



Abstract

Humanoid Robots are designed to navigate and interact within environments optimized for humans. While the rise of e-commerce was able to eliminate the need for in-person shopping at local stores, additional assistance is needed in receiving and unpacking the packages, and organizing the shipped products at the consumer's homes. This problem has been solved at an industrial level in the production line by devices such as the ABOT, but has not received much attention in the area of humanoid robotics. In our project we will focus on the opening of amazon boxes with a humanoid robot.

1. Initial project proposal

We will use the Baxter robot to hold a boxcutter or alternatively a customized cutting tool in its gripper that enables it to cut the tape on an amazon box. We will teach baxter to recognize the box, and also to recognize the tape on the box. Then we will have Baxter plan a cutting trajectory, which will be executed with one gripper, while the other is used to hold the box in place. After cutting the tape Baxter will move the box aside so he is ready to receive the next box. The cutting trajectory will be estimated based on the width of the box and the tape will be used as an indicator of the direction in which it is sealed.

Our first tier goals are as follows: to have baxter hold a cutting instrument firmly enough to perform the task, and model it in ROS. From our research we have learned that it can be beneficial to have a multi-blade, spring loaded end-effector or tool which can compensate for inaccuracies when cutting, such as when the opening of the box flaps is not exactly at the center of the tape. Thus, our choice of end-effector is critical to the success of our project. Furthermore, we will use reinforcement learning with kinetic input to have the robot learn from human instruction.

Our second tier goals are to have the robot trained to such an extent that he independently cuts the tape on an amazon box, and is able to hold it down with his other arm while doing so; that the robot is able to move the box aside, once cut open, to make room for the next box.

2. Previous Work

We consulted a variety of papers for this project, some of which focus on moving baxter in effective ways, but also papers that cover the learning aspect of our project. For example the two papers: Learning To Poke by Poking: Experiential Learning of Intuitive Physics, and Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation by the team at Berkeley.

3. Project Timeline

Developing Un-Paxter consists of three phases, first completing the research and refining the project, followed by learning to operate the Baxter robot and finally scripting the behavior of the robot via cartesian control and vision slicing. Research for Un-Paxter is focused on robots currently used commercially in warehouses and related to shipping and packaging. Additionally, we sought research of robots being used to cut objects such as food. Learning to operate Baxter consisted in starting with Rethink Robotics' tutorial and workspace in simulation at Columbia's clic lab. After successfully being able to control Baxter in simulation, we recreated the workspace and basic tutorial with the physical Baxter in the Robotics Lab. Upon having a functional basic demo for Baxter, scripts to cut a box based on cartesian control and on vision were developed. More detail around the proposed timeline and milestones accomplished can be found in the Appendix.

4. Setup and Development

4.1 Initial Setup

Assuming Ubuntu and ROS Indigo are installed, following 'Workstation Setup' tutorial from http://sdk.rethinkrobotics.com/wiki/Workstation_Setup:

Create ROS Workspace

```
$ mkdir -p ~/ros_ws/src
```

Source ROS Setup

```
$ source /opt/ros/indigo/setup.bash
```

Build and Install

```
$ cd ~/ros_ws
$ catkin_make
$ catkin_make install
```

Install SDK Dependencies

```
$ sudo apt-get update
$ sudo apt-get install git-core python-argparse python-wstool python-vcstools
python-rosdep ros-indigo-control-msgs ros-indigo-joystick-drivers
```

Install Baxter SDK

```
$ cd ~/ros_ws/src
$ wstool init .
$ wstool merge
https://raw.githubusercontent.com/RethinkRobotics/baxter/master/baxter_sdk.rosi
nstall
$ wstool update
```

Source ROS Setup

```
$ source /opt/ros/indigo/setup.bash
```

Build and Install

```
$ cd ~/ros_ws
$ catkin_make
$ catkin_make install
```

Download the baxter.sh script

```
$ wget https://github.com/RethinkRobotics/baxter/raw/master/baxter.sh
$ chmod u+x baxter.sh
```

Customize the baxter.sh script

```
$ cd ~/ros_ws
$ gedit baxter.sh
```

Customize

```
**baxter_hostname="baxter_hostname.local"**
**your_ip="192.168.XXX.XXX"**
***ros_version="indigo"***
```

Initialize your SDK environment

```
$ cd ~/ros_ws
$ . baxter.sh
```

Install MoveIt!, following http://sdk.rethinkrobotics.com/wiki/MoveIt_Tutorial

Install MoveIt

```
$ cd ~/ros_ws/src
$ git clone https://github.com/ros-planning/moveit_robots.git
```

Required packages

```
$ sudo apt-get update
$ sudo apt-get install ros-indigo-moveit-full
```

Make new additions

```
$ cd ~/ros_ws/
$ ./baxter.sh
$ catkin_make
```

Plug in and press Baxter power button

Source
```
$ source baxter.sh
```

Enable/disable the robot
```
$ rosrun baxter_tools enable_robot.py -e / -d
```

Untuck/tuck the robot*
```
$ rosrun baxter_tools tuck_arms.py -u / -t
```

***Edit the tuck untuck script neutral positions of choice**

Run joint trajectory server
```
$ rosrun baxter_interface joint_trajectory_action_server.py
```

Run MoveIt
```
$ roslaunch baxter_moveit_config move_group.launch
```

Run Baxter demo
```
$ roslaunch baxter_moveit_config demo_baxter.launch
```

View joint states
```
$ rostopic echo robot/joint_states
```

View on RVIZ
```
$ rosrun rviz rviz
```

4.2 Creating Project Workspace

Steps to run project specific script:

Run joint trajectory server
```
$ rosrun baxter_interface joint_trajectory_action_server.py
```

Run MoveIt
```
$ roslaunch baxter_moveit_config move_group.launch
```

Adding, running and calibrating camera
```
$ roslaunch openni_launch openni.launch
```

To test and view camera images
```
$ rostopic echo  camera/rgb/image_rect_color
$ rosrun image_view image_view image:=/camera/rgb/image_react_color
right clic to save frame
```

Run PC Filter for depth segmentation

```
$ Roslaunch pc_filter pc_filter.launch
```

Camera interactive transform with updated coordinates

```
install lcsr_tf_tools package
$ rosrun lcsr_tf_tools interactive_transform_publisher 0.338245 0.0406369
0.757728  0.0136623 0.60469 0.0184116 0.796131 /base /camera_link 10
```

Running project script

```
$ roslaunch baxter_project baxter_project.launch
```

Adding camera to Pointcloud2 in RVIZ

```
Add pointcloud2
Topic depth_registered/points
Add interactive markers
Update topic, camera_link/update
Update stored w x y z from rviz:
Tf >frames disable all> camera_link
```

Run AR tracking for calibration*

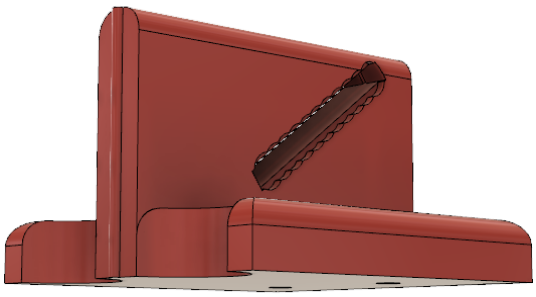
```
$ roslaunch ar_tracking baxter_bundle_curg.launch
```

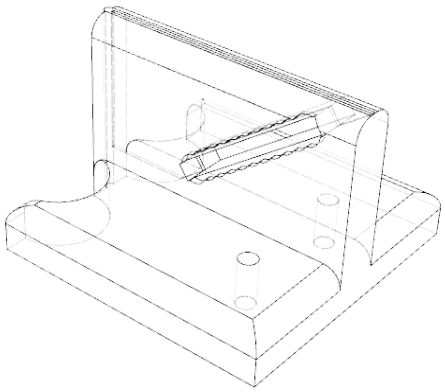
Launch file above was authored by our team, added to CURG Github repository through Iretiayo's account

***decided to not ultimately use this tool**

4.3 Custom Parts

Upon observing that Baxter's original grippers would not be able to hold a blade or sharp object firmly enough to cut a box open, we opted to remove the grippers and create a custom part which we 3D printed. The 3D printed gripper was created based on the dimensions of Baxter's wrist the screw positions so we could firmly mount it to the robot. The gripper can sustain both real and dummy blade as the gap can be adjusted using screws. A rendering of the piece we designed can be seen below:





Upon removing the grippers, the robot URDF file had to be adjusted to run on a robot model that had grippers set to null. Otherwise, ros will seek connected grippers and will also not be able to calculate collisions correctly. In robot model, baxter_description, we set electric_gripper (both right and left) to null_gripper.

Specifically, in the following sections in the following files:

```
/home/bo/ros/baxter_new/build/install_manifest.txt:
   91
/home/bo/ros/baxter_new/install/share/rethink_ee_description/urdf/electric_gripper/fingers/sta
ndard_wide.xacro
   92
/home/bo/ros/baxter_new/install/share/rethink_ee_description/urdf/electric_gripper/example_end
_effector.urdf.xacro
   93:
/home/bo/ros/baxter_new/install/share/rethink_ee_description/urdf/electric_gripper/rethink_ele
ctric_gripper.xacro
   94  /home/bo/ros/baxter_new/install/share/baxter_maintenance_msgs/msg/CalibrateArmData.msg
   95
/home/bo/ros/baxter_new/install/share/baxter_maintenance_msgs/msg/CalibrateArmEnable.msg

/home/bo/ros/baxter_new/install/share/baxter_description/urdf/left_end_effector.urdf.xacro:
       1  <?xml version="1.0" ?>
       2  <robot name="left_end_effector" xmlns:xacro="http://www.ros.org/wiki/xacro">
       3:   <xacro:include filename="$(find
rethink_ee_description)/urdf/electric_gripper/rethink_electric_gripper.xacro" />
       4      <xacro:rethink_electric_gripper side="left"
       5                                      l_finger="extended_narrow"

/home/bo/ros/baxter_new/install/share/baxter_description/urdf/right_end_effector.urdf.xacro:
       1  <?xml version="1.0" ?>
       2  <robot name="right_end_effector" xmlns:xacro="http://www.ros.org/wiki/xacro">
       3:   <xacro:include filename="$(find
rethink_ee_description)/urdf/electric_gripper/rethink_electric_gripper.xacro" />
       4      <xacro:rethink_electric_gripper side="right"
       5                                      l_finger="extended_narrow"

/home/bo/ros/baxter_new/install/share/rethink_ee_description/urdf/electric_gripper/example_end
_effector.urdf.xacro:
       1  <?xml version="1.0"?>
       2  <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="left_electric_gripper">
       3:   <xacro:include filename="$(find
rethink_ee_description)/urdf/electric_gripper/rethink_electric_gripper.xacro" />
       4      <xacro:rethink_electric_gripper side="left"
       5                                      l_finger="extended_narrow"

/home/bo/ros/baxter_new/src/baxter_common/baxter_description/urdf/left_end_effector.urdf.xacro
:
       1  <?xml version="1.0" ?>
       2  <robot name="left_end_effector" xmlns:xacro="http://www.ros.org/wiki/xacro">
       3:   <xacro:include filename="$(find
rethink_ee_description)/urdf/electric_gripper/rethink_electric_gripper.xacro" />
       4      <xacro:rethink_electric_gripper side="left"
       5                                      l_finger="extended_narrow"

/home/bo/ros/baxter_new/src/baxter_common/baxter_description/urdf/right_end_effector.urdf.xacr
o:
```

```
       1  <?xml version="1.0" ?>
       2  <robot name="right_end_effector" xmlns:xacro="http://www.ros.org/wiki/xacro">
       3:   <xacro:include filename="$(find
rethink_ee_description)/urdf/electric_gripper/rethink_electric_gripper.xacro" />
       4      <xacro:rethink_electric_gripper side="right"
       5                                      l_finger="extended_narrow"

/home/bo/ros/baxter_new/src/baxter_common/rethink_ee_description/urdf/electric_gripper/example
_end_effector.urdf.xacro:
       1  <?xml version="1.0"?>
       2  <robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="left_electric_gripper">
       3:   <xacro:include filename="$(find
rethink_ee_description)/urdf/electric_gripper/rethink_electric_gripper.xacro" />
       4      <xacro:rethink_electric_gripper side="left"
       5                                      l_finger="extended_narrow"
```

4.4 Testing Environment

5. Robot Control

5.1 Initial Pose

Baxter's initial pose is recorded via listening to the joint states, the baxter_project script will establish a start pose and complete the task and return to the pose the robot was in prior to launching the script. We also integrated the start pose into Baxter's Tuck and Untuck scripts which place the robot's arms in a position better geared for effecting the cutting motion.

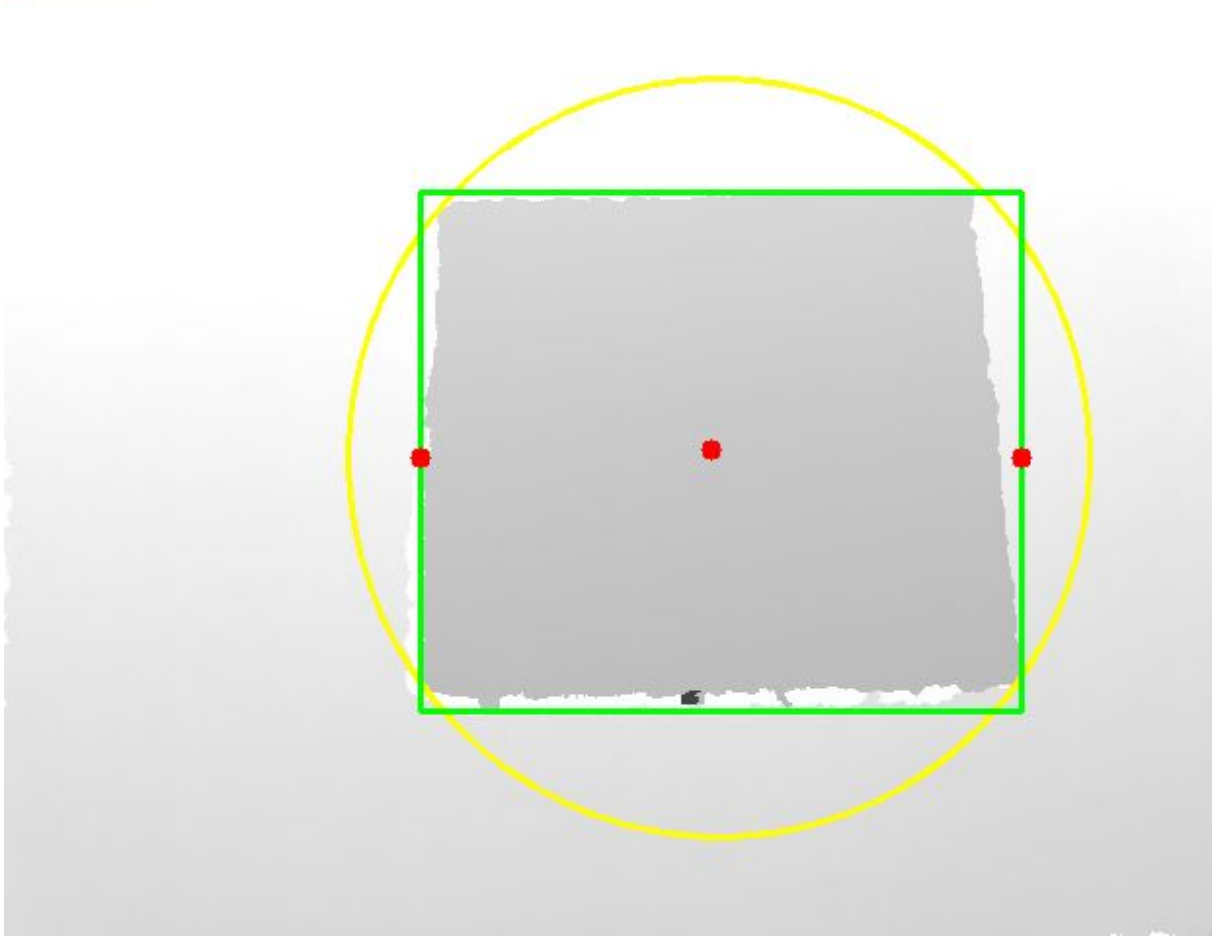
5.2 Cartesian Control

Un-Paxter's cartesian control script is based on a PR2 cartesian control script (source: https://github.com/kunal15595/ros/blob/master/moveit/src/moveit_pr2/pr2_moveit_tutorials/planning/scripts/move_group_python_interface_tutorial.py). The script works with several modules, mainly moveit_commander. The script generates a plan based on a start pose and an end pose, intermediate poses, waypoints, are interpolated in between the two points based on a set distance in between steps.

5.3. Vision

For vision, we have a script image_grabber which subscribes to the camera and stores depth and RGB images. We ultimately chose to use the depth image, transform it to better show the top of the box. We also stored the original depth image data as a csv to be able to retrieve the original depths in meters.


The second part of the script analyzes the image, finds the contours, then selects the largest contour and finds the smallest rectangle around it. From there we found the midpoint and the two endpoints for the box.



With the x,y for each endpoint in the depth image, we indexed the original depth data to obtain depth, z. Having x,y,z for each endpoint we then first transform x,y to meters from the camera frame and then transform x,y,z,0 to x,y,z in the robot frame by using a base to camera_link transform. These points are then used in the cartesian control script as precut and postcut pose orientations. Due to the angle from which the camera perceives

the box, we used the center of mass of the contour to find the middle of the box, which allows us to find the “real” middle without having to geometrically account for the distortion.

6. Results

Demo video:
https://drive.google.com/a/columbia.edu/file/d/0B645yhVX_EwvUmVVRVBCb09DNXc/view?usp=sharing

The vision script is able to obtain x, y, z points for the box top and send them to a planning library, however the transform between the camera_link points and the base isn’t correct and planning fails. The pipeline from vision to transform to planning works, the transform adjustment needed is likely small and may require some better understanding of the tf documentation. All in all we produced a working software that can perform the job with additional calibration.

7. Further Work and team reflections

As further research, we would like to implement this task using reinforcement learning. Ideally, the final pipeline would include being able to recognize the box using vision and then adapting the cutting movement to the dimensions of the box.

We have been working on the physical robot early on the project. To our benefit we got comfortable with the real robot relatively quickly. On the downside, we ran into several technical problems unrelated to our project that required a significant amount of our time to resolve. Overall, we learned how to set-up and operate a robot, to create a workflow pipeline including vision and cartesian control. We learned to interact with ros packages and created our own for the project. We learned to debug and customize existing packages, such as editing starting and neutral poses. We interacted with ros nodes and filtered them, for example by the use of PC_filter to filter out depth in images, and TF to transform the camera_link to the base. We learned that working with integrated systems provides provides a set of unique challenges in debugging, since there are a variety of sources of error from the hardware to the software level. It has been a challenging and exciting semester.

There were a few elements and tools we learned and worked with, but didn’t ultimately use:

- PC_filter - for depth filtering
- AR_tracking - for camera calibration

8. Appendix

8.1 Responsibilities

Below are the original tasks and responsibilities submitted in the proposal

Task	**Notes**	**Responsibility**	**Date**	**Status**
Complete research	Research for overall work in box cutting, warehouses, etc	Philippe, Oriana	February	Complete

Documentation	Ongoing log of bugs, necessary questions, goals attained, changes to plan	Philippe, Oriana	Ongoing	Ongoing
Refine project idea	Refine by goal, work based off of research and support from TAs	Philippe, Oriana	February	Complete
Learn to control Baxxter	Understand basic tasks and movements	Philippe, Oriana	February	Complete
Decide reinforcement and forward kinematics aspects	Evaluate which part of the task will be trained through reinforcement learning, and which part needs to be hardcoded using forward or inverse kinematics.	Philippe, Oriana	February	Further work
Grasping	Evaluate a fitting end-effector: single or multiblade: https://www.amazon.com/Vegetable-Shredder-Kitchen-Scallion-Tools/dp/B01KKLZ4ZO/ref=sr_1_31?ie=UTF8&qid=1487693736&sr=8-31&keywords=multi+blade+cutter	Philippe	March	Switched for custom gripper
Vision	Work on identifying box and tape in order to determine viable cutting patterns	Oriana	March	Learned to integrate camera into ros, have point cloud
Movement	Plan arm movement with necessary paths and force	Oriana	March	Learned basic arm planning and movement
Reinforcement learning	Research reinforcement learning on Baxter, techniques used for arm movements	Philippe	March	Further work
Refine bugs and movements	Ensure to include in documentation	Philippe, Oriana	April 11	Complete
Produce presentation, finalize documentation	Summary of goals completed, changes since proposal and potential further development	Philippe, Oriana	April 18	Complete
Final presentation		Philippe, Oriana	April 25	Complete
Final report		Philippe, Oriana	May 5	Complete

8.2 Development and work log

March 21
- Run Baxter simulation in Gazebo from HumanoidRobotics github
- Able to control arms and gripper
- Contact TAs about working with Kinect

March 29
- Begin work with physical robot, Baxter, in Robotics Lab
- Issues with Camera:
    - In a launch file we need to find the publishing of the transformation from the camera to the topic in rviz
- At startup
    - Initialize source in every terminal window
    - Make sure to enable robot

April 4
- Useful commands:
    - http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf
- Adding images to Baxter's display
    - http://sdk.rethinkrobotics.com/wiki/Display_Image_-_Code_Walkthrough
- Launch Baxter with a 'home' pose so that we limit the need for additional movement

April 5
- Remove baxter grippers
- Design custom gripper for blades
- Load source code to github

April 12
- Add custom gripper to baxter
- Unable to run baxter with null grippers
- Start clean workspace
- Upload new clean workspace to github

April 15
- Fixed null gripper issue
- Work on segmentation code in python
- Work on documentation

April 19
- Migrated to new computer due to issue with python packages for OpenCV
- Implemented cartesian control

April 25
- Create demo video

- Finalize presentation
- Work on documentation

April 26
- Fixed OpenCV
- Added image to Baxter's screen coordinated with cutting task
- Able to run camera correctly
- Used transform publisher to coordinate point cloud with rendering in rviz
    - After finding correct coordinates, integrate into launch file

April 28
- Work on documentation

May 1
- Work on documentation

May 2
- Working on image slicing and obtaining

May 3
- Continue work on camera image analysis
- Build wrapper for pipeline once image analysis is working
- Perform demo

May 5
- Complete vision to planning pipeline
- Complete documentation

8.3 Python code

------------------------- cartesian_controlv2.py ---

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

# Software License Agreement (BSD License)
#
# Copyright (c) 2013, SRI International
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
#  * Redistributions of source code must retain the above copyright
#      notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above
#      copyright notice, this list of conditions and the following
#      disclaimer in the documentation and/or other materials provided
#      with the distribution.
#  * Neither the name of SRI International nor the names of its
#      contributors may be used to endorse or promote products derived
#      from this software without specific prior written permission.
#
```

```
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
#
# Author: Acorn Pooley
# Modified by: Oriana I Fuentes, Philippe Wyder
import os
import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
import cv2
import numpy as np
import cv_bridge
from std_msgs.msg import String
from sensor_msgs.msg import (
       Image,
)
from tf import TransformListener
import tf
import sensor_msgs

KERNEL_SIZE = 11
PATH  = "/home/baxter/ros/baxter_new/src/baxter_project/scripts/"

# changes baxter's expression
def change_screen_image(path):
  img = cv2.imread(path)
  msg = cv_bridge.CvBridge().cv2_to_imgmsg(img, encoding="bgr8")
  pub = rospy.Publisher('/robot/xdisplay', Image, latch=True, queue_size=1)
  pub.publish(msg)
  # Sleep to allow for image to be published.
  rospy.sleep(1)

def getXYZ(image_x, image_y, depth):
  # uses the camera specifictions to turn the x, y values from pixels into meters

  # camera constants
  K= [525.0, 0.0, 319.5, 0.0, 525.0, 239.5, 0.0, 0.0, 1.0]

  fx_inv = 1.0 / K[0]
  fy_inv = 1.0 / K[4]
  cx = K[2]
  cy = K[5]
  # convert x and y values to meters
  out_x = depth * ((image_x-cx) * fx_inv)
```

```
  out_y = depth * ((image_y-cy) * fy_inv)
  out_z = depth

  return out_x, out_y, out_z


def vision_coordinates(out_x1, out_y1, out_z1, out_x2, out_y2, out_z2):
  # loads camera coordinate transformation frame from the csv stored by cv_transform.py
  base_camera_transformation = np.loadtxt(PATH + "tf_base_camera.csv", delimiter=',')
  pre_cut   = np.array([out_x1, out_y1, out_z1, 0])
  post_cut  = np.array([out_x2, out_y2, out_z2, 0])

  out_x1, out_y1, out_z1, _ = np.dot(base_camera_transformation, pre_cut)
  out_x2, out_y2, out_z2, _ = np.dot(base_camera_transformation, post_cut)

  return out_x1, out_y1, out_z1, out_x2, out_y2, out_z2

def get_coordinates():
  image = cv2.imread(PATH + "image_rect.png")
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  mask = cv2.inRange(gray, 185, 223)

  # dilate and erode image to ensure distinct contours
  kernel = np.ones((KERNEL_SIZE, KERNEL_SIZE),np.uint8) * 255 # 255 - value for white
  erosion = cv2.erode(mask, kernel, iterations = 2)
  dilation = cv2.dilate(erosion, kernel, iterations = 2)

  cnts = cv2.findContours(dilation.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]
  center = None
  area = None
  # assumes that the box is the largest contour in its vield of view
  if len(cnts) > 0:
        print "Inside len cnts"
        # maximum value of cnts given the key contourArea
        c = max(cnts, key=cv2.contourArea)
        # enclosing circle value extracted
        ((i,j), radius) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)
        # centroid calculation
        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
        area = M["m00"]
        #print "Area is ", M["m00"]

        x,y,w,h = cv2.boundingRect(c)

        depth_map = np.loadtxt(PATH +"image_rect.csv", delimiter=",")

        # Get Pre-cut location & Post-cut location
        # pre-cut out_x1, out_y1, out_z1,
        out_x1 = x
        out_y1 = int(j)
        out_z1 = depth_map[x+int(w*.25)][int(j)]
        # post-cut out_x2, out_y2, out_z2
        out_x2 = x+w
        out_y2 = int(j)
        out_z2 = depth_map[x+int(w*.75)][int(j)]
```

```
        # Transform x and y values of pre-cut & post-cut location to Meters
        out_x1, out_y1, out_z1 = getXYZ(out_x1, out_y1, out_z1)
        out_x2, out_y2, out_z2 = getXYZ(out_x2, out_y2, out_z2)

        # Transform pre-cut & post-cut pose to base frame
        out_x1, out_y1, out_z1, out_x2, out_y2, out_z2 = vision_coordinates(out_x1, out_y1,
out_z1, out_x2, out_y2, out_z2)

        return out_x1, out_y1, out_z1, out_x2, out_y2, out_z2

def move_group_python_interface_tutorial():

  ## First initialize moveit_commander and rospy.
  sys.argv.append('joint_states:=/robot/joint_states')
  print "============ Starting tutorial setup"
  moveit_commander.roscpp_initialize(sys.argv)
  rospy.init_node('move_group_python_interface_tutorial',
              anonymous=True)

  ## Instantiate a RobotCommander object.  This object is an interface to
  ## the robot as a whole.
  robot = moveit_commander.RobotCommander()

  ## Instantiate a PlanningSceneInterface object.  This object is an interface
  ## to the world surrounding the robot.
  scene = moveit_commander.PlanningSceneInterface()

  ## Instantiate a MoveGroupCommander object.  This object is an interface
  ## to one group of joints.  In this case the group is the joints in the left
  ## arm.  This interface can be used to plan and execute motions on the left
  ## arm.
  group = moveit_commander.MoveGroupCommander("left_arm")

  # CHANGE IMAGE

change_screen_image("/home/baxter/ros/baxter_new/src/baxter_examples/share/images/futureisnow.
png")
  ## We create this DisplayTrajectory publisher which is used below to publish
  ## trajectories for RVIZ to visualize.
  display_trajectory_publisher = rospy.Publisher(
                                      '/move_group/display_planned_path',
                                      moveit_msgs.msg.DisplayTrajectory, queue_size=10)

  ## Wait for RVIZ to initialize. This sleep is ONLY to allow Rviz to come up.
  print "============ Waiting for RVIZ..."
  rospy.sleep(10)

  ## Getting Basic Information
  ##
  ## We can get the name of the reference frame for this robot
  print "============ Reference frame: %s" % group.get_planning_frame()

  ## We can also print the name of the end-effector link for this group
  print "============ Reference frame: %s" % group.get_end_effector_link()

  ## We can get a list of all the groups in the robot
  print "============ Robot Groups:"
```

```
  print robot.get_group_names()

  ## Sometimes for debugging it is useful to print the entire state of the
  ## robot.
  print "============ Printing robot state"
  print robot.get_current_state()
  print "============"

  # Change Image

change_screen_image("/home/baxter/ros/baxter_new/src/baxter_examples/share/images/wolverine01.
png")

  # Create list of waypoints for cutting trajectory
  waypoints = []

  # start with the current pose
  waypoints.append(group.get_current_pose().pose)

  out_x1, out_y1, out_z1, out_x2, out_y2, out_z2 = get_coordinates()
  print "coordinates", out_x1, out_y1, out_z1, out_x2, out_y2, out_z2

  # Set our neutral_pose (assuming baxter cuts from right to left)
  neutral_pose = geometry_msgs.msg.Pose()
  neutral_pose.position.x = 0.58041
  neutral_pose.position.y = 0.3062
  neutral_pose.position.z = 0.3805
  neutral_pose.orientation.x = .7183
  neutral_pose.orientation.y = .6895
  neutral_pose.orientation.z = -0.0581
  neutral_pose.orientation.w = 0.0724
  # Set our precut_pose (assuming baxter cuts from right to left)
  precut_pose = geometry_msgs.msg.Pose()
  precut_pose.position.x = out_x1
  precut_pose.position.y = out_y1
  precut_pose.position.z = out_z1
  precut_pose.orientation.x = .6729
  precut_pose.orientation.y = .7388
  precut_pose.orientation.z = -0.0264
  precut_pose.orientation.w = 0.0264

  # Set our precut_pose (assuming baxter cuts from right to left)
  postcut_pose = geometry_msgs.msg.Pose()
  postcut_pose.position.x = out_x2
  postcut_pose.position.y = out_y2
  postcut_pose.position.z = out_z2
  postcut_pose.orientation.x = 0.7268
  postcut_pose.orientation.y = 0.6858
  postcut_pose.orientation.z = -0.0252
  postcut_pose.orientation.w = 0.029

  # append all waypoints to list
  waypoints.append(neutral_pose)
  waypoints.append(precut_pose)
  waypoints.append(postcut_pose)
  waypoints.append(group.get_current_pose().pose)
  # start cutting job along waypoints
```

```
  cutting_job(waypoints, group)
  # CHANGE IMAGE

change_screen_image("/home/baxter/ros/baxter_new/src/baxter_examples/share/images/wolverine03.
png")

def cutting_job(waypoints, group):

  # CHANGE IMAGE

change_screen_image("/home/baxter/ros/baxter_new/src/baxter_examples/share/images/wolverine02.
png")
  ## We want the cartesian path to be interpolated at a resolution of 1 cm
  ## which is why we will specify 0.01 as the eef_step in cartesian
  ## translation.  We will specify the jump threshold as 0.0, effectively
  ## disabling it.
  (plan3, fraction) = group.compute_cartesian_path(
                              waypoints,   # waypoints to follow
                              0.01,          # eef_step
                              0.0)           # jump_threshold

  print "============ Waiting while RVIZ displays plan3..."
  rospy.sleep(5)
  if fraction < 1:
       print('Plan failed')
  else:
       group.execute(plan3)

  print "============ STOPPING"


if __name__=='__main__':
  try:
       move_group_python_interface_tutorial()
  except rospy.ROSInterruptException:
       pass
```

------------------------------------ cv_transform.py --

```
import rospy
from tf import TransformListener
import tf
import cv_bridge
import rospy
import sensor_msgs
import matplotlib.pyplot as plt
import cv2
import numpy as np

#
#      cv_transform.py
#      @author: Oriana I. Fuentes
#      Script to store the transformation frame from the camera to the base in a csv_file
#      so it can be imported by the rosnode
#      this form of saving makes sense since the robot is stationary and so is the camera
#

FILENAME = "/home/baxter/ros/baxter_new/src/baxter_project/scripts/tf_base_camera.csv"
```

```
if __name__ == '__main__':
        rospy.init_node('tf_listener')
        listener = tf.TransformListener()
        listener.waitForTransform('/base', '/camera_link', rospy.Time(), rospy.Duration(4.0))
        (trans,rot) = listener.lookupTransform('/base', '/camera_link', rospy.Time(0))
        print trans,rot
        base_camera_transformation = listener.fromTranslationRotation(trans, rot)
        np.savetxt(FILENAME, base_camera_transformation, delimiter=',')
```

------------------------------ image_grabber.py -------------------------

```
# coding: utf-8
import cv_bridge
import rospy
import sensor_msgs
import matplotlib.pyplot as plt
import cv2
import numpy as np
import sys

# image_graber.py
# @author Oriana I. Fuentes & Philippe Wyder
# Runs a ros node that grabs an image every ten seconds and stores it
# either as a png file or a csv file.
# The script can be run in two modes, as a DepthGrabber and as a ImageGrabber:
# python image_grabber depth
# will load the depth map and store it once as a csv file and once as a png file,
# these files will later be used by the cartesian_controlv2.py to retrieve the
# cutting location.

# python image_graber image
# Saves the rgb image as png for further processing, currently not needed.

PATH = "/home/baxter/ros/baxter_new/src/baxter_project/scripts/"

class DepthGrabber:
        def __init__(self, image_topic="/camera/depth/image_rect"):
                self.currentImage = None
                self.image_topic1 = image_topic
                self.bridge = cv_bridge.CvBridge()

        def getcurrentImage(self):
                return self.getcurrentImage

        def startImageGrab(self):
                rospy.init_node('listener', anonymous=True)
                rospy.Subscriber(self.image_topic1,sensor_msgs.msg.Image, self.callback)


                rospy.spin()

        def callback(self, data):
                self.currentImage = self.bridge.imgmsg_to_cv2(data, desired_encoding =
"passthrough")
                rospy.loginfo("received new image")
                FILENAME = PATH + self.image_topic1.split('/')[-1]
                np.savetxt(FILENAME+".csv", self.currentImage, delimiter=",")
```

```
                depth_array = np.array(self.currentImage, dtype=np.float32)
                cv2.normalize(depth_array, depth_array,0,1, cv2.NORM_MINMAX)
                rospy.loginfo("received new image")
                IMAGE = self.image_topic1.split('/')[-1]+".png"
                cv2.imwrite(FILENAME+".png", self.currentImage*255)

                rospy.sleep(10)




class ImageGrabber:
        def __init__(self, image_topic="/camera/rgb/image_rect_color"):
                self.currentImage = None
                self.image_topic1 = image_topic
                self.bridge = cv_bridge.CvBridge()

        def getcurrentImage(self):
                return self.getcurrentImage

        def startImageGrab(self):
                rospy.init_node('listener', anonymous=True)
                rospy.Subscriber(self.image_topic1,sensor_msgs.msg.Image, self.callback)
                rospy.spin()

        def callback(self, data):
                self.currentImage = self.bridge.imgmsg_to_cv2(data, desired_encoding =
"passthrough")
                rospy.loginfo("received new image")
                IMAGE = self.image_topic1.split('/')[-1]+".png"
                cv2.imwrite(IMAGE, self.currentImage)
                rospy.sleep(10)


if __name__ == '__main__':
        topic = ""
        if (len (sys.argv) == 2):
                im_type = sys.argv[1]
                print(im_type)
                if im_type == "depth":
                                dm_grabber = DepthGrabber(image_topic="/camera/depth/image_rect")
                                dm_grabber.startImageGrab()
                elif im_type == "image":
                                im_grabber =
ImageGrabber(image_topic="/camera/rgb/image_rect_color")
                                im_grabber.startImageGrab()
                else:
                        print("incorrect, pick depth or image")

        else:
                print("need arguments for topic")
```

--------------------------- baxter_project_vision.launch ------------------------

```
<?xml version="1.0"?>
```

```
<launch>
  <!-- <include file="$(find baxter_moveit_config)/launch/demo_baxter.launch"/> -->
  <include file="$(find baxter_moveit_config)/launch/demo_baxter.launch"/>

  <node name="move_group_python_interface_tutorial" pkg="baxter_project"
type="cartesian_controlv2.py" respawn="false" output="screen">
  </node>

</launch>
```

9. References

CURG Github
https://github.com/CURG

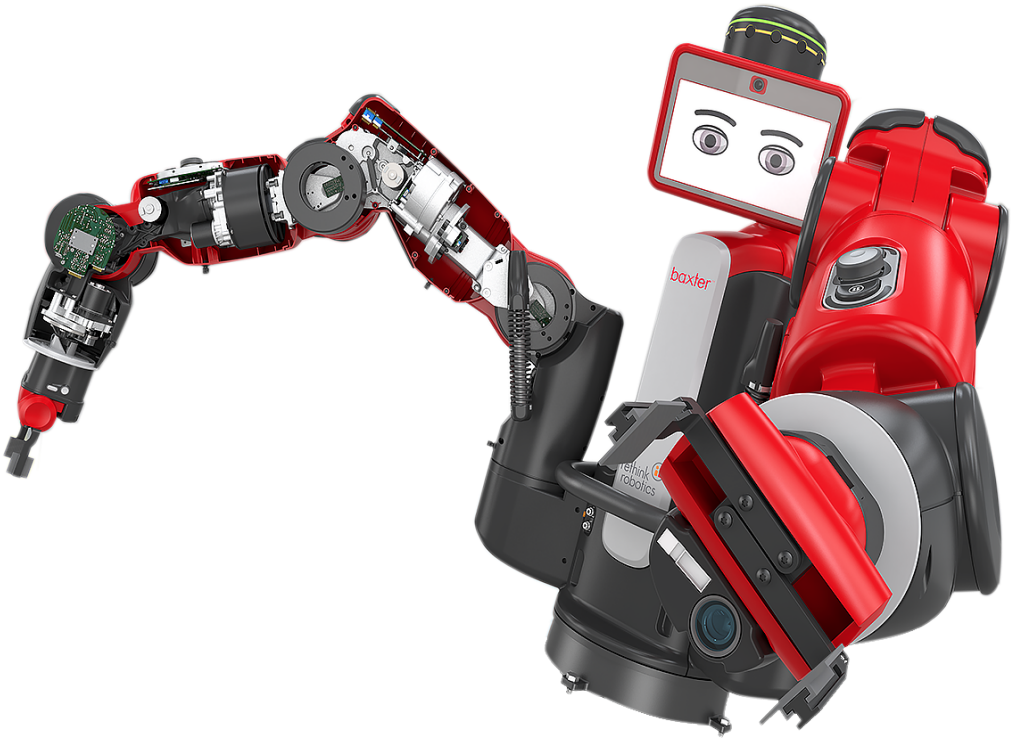
Rethink Robotics SDK wiki
http://sdk.rethinkrobotics.com/wiki/Baxter_Setup
http://sdk.rethinkrobotics.com/wiki/Display_Image_-_Code_Walkthrough
http://sdk.rethinkrobotics.com/wiki/MoveIt_Tutorial

ROS docs
http://docs.ros.org/hydro/api/pr2_moveit_tutorials/html/planning/scripts/doc/move_group_python_interface_tutorial.html

Kunal Khandelwal Github
https://github.com/kunal15595/ros/blob/master/moveit/src/moveit_pr2/pr2_moveit_tutorials/planning/scripts/move_group_python_interface_tutorial.py

Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation
Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, Sergey Levine
https://ropemanipulation.github.io/

Learning to Poke by Poking: Experiential Learning of Intuitive Physics
Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, Sergey Levine
http://ashvin.me/pokebot-website/


baxter
rethink robotics