Preface and Acknowledgements

I worked as an intern at the Artificial Intelligence and Robotics Laboratory(AIRLAB) in Istanbul Technical University between 18th of June and 13th of July 2018. Under the supervision of Associate Professor Sanem Sarıel, I worked along 2 MSc students: Abdullah Cihan Ak and Arda İnceoğlu. During my internship, I studied ROS and worked on Baxter, an industrial humanoid robot. My major task was to write an interface for Baxter, which would make the robot perform certain actions based on the user's input from the robot's screen screen aiming to make manual use of the robot easier during experiments. During this project, I worked with a fellow High School intern Emre.

This internship enhanced my interest in computer sciences and robotics and introduced me to new tools and techniques such as ROS. I am sure that I will work on and further develop these skills throughout my high school career and beyond. I had the chance to experience what it is like to work on a research environment in computer engineering. Working alongside and observing graduate students on their researches inspired me deeply and strengthened my thoughts of pursuing a career and education in computers science and engineering.

I would like to thank Associate Professor Sanem Sarıel for giving me the opportunity to work at AIRLAB and making these wonderful 4 weeks possible. I am truly grateful for the help of Cihan Ak and Arda İnceoğlu for being more than willing to help us at any moment and for aiding and supporting us throughout these 4 weeks. Also, I would like to thank all other students and researchers working at AIRLAB for helping us and offering us a very friendly environment.

Baxter User Interface

Overview

Our main project was to write a user interface for Baxter, that would allow the user to make the robot perform certain actions without entering commands from their computers. The idea for this project was born because of the difficulties of working on the robot while entering every single command from a computer.

Our interface displays a menu on Baxter's screen which consists of 4 basic actions: Open or close gripper, move hand on axis x,y or z, rotate on selected axis, and move to home position(Figure 1). The menu is controlled by the wheel, Ok and the back buttons located on Baxter's left arm. The user can navigate by rotating the wheel and can select an action by pressing Ok. While some actions like "Open Gripper" functions directly after Ok is pressed, others like "Move On Axis" asks for further information such as the distance(Figure 2).

Figure 1-Baxter UI Main Menu

Figure 2-Baxter UI Move On Axis z: Select Distance

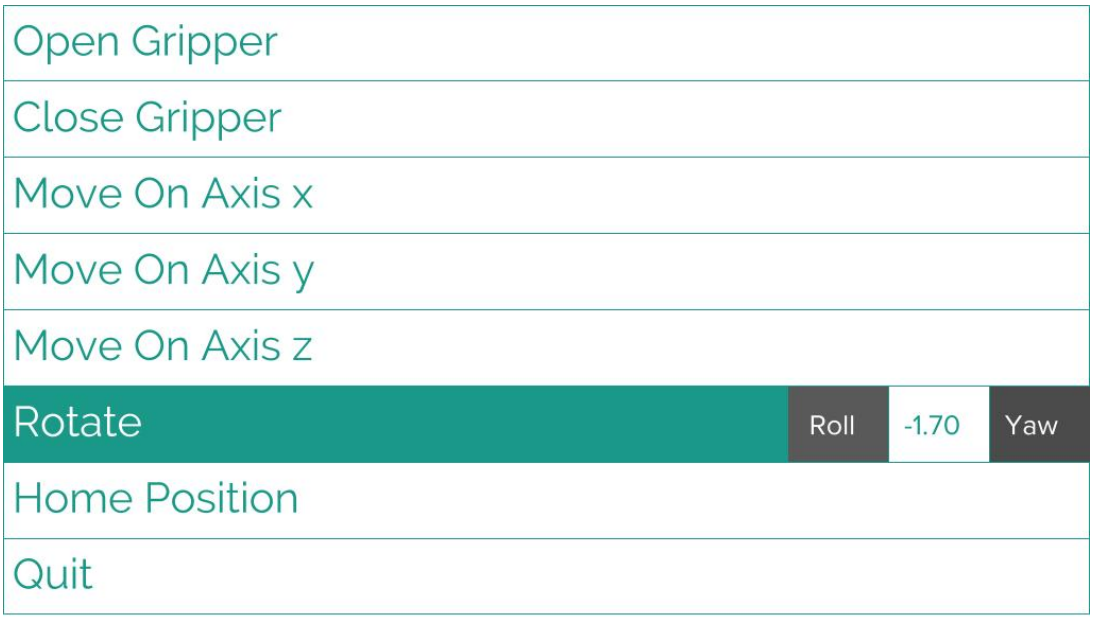

Figure 3-Baxter UI Rotate: Select Pitch Angle

Programming the UI

ROS

ROS(Robot Operating System) is a collection of tools and libraries which make the process of creating complex robot software simpler. Some of the most important concepts in Ros are Node, Publisher, Subscriber, Service Client and topic.

Nodes are executable files in a ROS package; these nodes can communicate with each other by publishing information to topics and listening to certain topics. The information is sent in the form of messages and a topic is just a medium where nodes exchange messages. Nodes that publish messages to topics are called publishers and nodes that listen to topics are called subscribers.

Another way of communication between nodes is the service/client model. In this system, a node provides a service while another node request this service by a message and then waits for a response.

The software for Baxter is based on the Robot Operating System so in our program, we heavily utilized concepts and tools offered by ROS.

Overview

Our code is written in Python and is located in the baxter_sim_io package in a Catkin generated workspace. The code is in the form of a launch file which launches our main program, baxter_gui.py, and some other necessities for the program to be able to run properly.

Our User Interface consists of premade images that change based on the feedbacks from the buttons. To display these images, we used baxter_sentiment_service which is a

service that makes it possible to display a selected image to Baxter's screen. We created a function called faceinterface_client which uses the service provided by baxter_sentiment_service, making us able to display any image for a given time and priority to Baxter's screen. Our program operates around this function, displaying the different menus and selections to the screen to make the UI possible.

```
def faceinterface_client(strg,pri,dur):
	rospy.wait_for_service('/baxter_sentiment_service')
	try :
		facedisplay_str = rospy.ServiceProxy('/baxter_sentiment_service',
Sentiment)
		resp1 = facedisplay_str(strg,pri,dur)
		return resp1.results

	except rospy.ServiceException, e:
		print "Service call failed: %s"%e
```

Obtaining Button States

Our Menu is navigated by using the buttons located on Baxter's left arm. These button states are published to several topics and our node subscribes to these topics to listen to the button states. Then, our node uses this information to display and navigate through the menu. New images are displayed as you scroll down the menu or select an action.

Making the robot move

The robot is able to perform the actions selected on the menu through the commander node, using the baxter_behaviour_server and the MoveIt package. We wrote a client for our node that used baxter_behaviour_server and by this client, we can request basic actions like opening the gripper or moving on an axis.

```
def commander_client(x):
	rospy.wait_for_service("/BehaviourActionServer/BaxterBehaviourActionServer/reque
st_action")
	try:
		client =
rospy.ServiceProxy("/BehaviourActionServer/BaxterBehaviourActionServer/request_action",
Action)
		resp1 = client(x)
		return resp1.success
	except rospy.ServiceException, e:
		print "Service call failed: %s"%e
```

For the rotation of the gripper, we used MoveIt which is a motion planning software. By using Moveit, we planned a trajectory for the robot and made it change its rotation according to the values selected from the menu. We only wanted to change one rotation axis of the robot while keeping the end point the same so we used the tf package to get information about the robots state and position. While planning the path for the robot, we kept every variable the same (using the values we obtained from tf) except the axis that we wanted to change.

```
#getting the left gripper's current position
def get_robot_pos():
	global trans
	global new_rot
	listener = tf.TransformListener()
	rate = rospy.Rate(10.0)
	while not rospy.is_shutdown():
		try:
			(trans,rot) = listener.lookupTransform('/base', '/left_gripper',
rospy.Time(0))
			new_rot = tf.transformations.euler_from_quaternion([rot[0], rot[1],
rot[2], rot[3]])
		except (tf.LookupException, tf.ConnectivityException,
tf.ExtrapolationException):
			pass
		rate.sleep()
```

```
def MoveIt(qx,qy,qz):
	global trans
	print "============ Starting tutorial setup"
	moveit_commander.roscpp_initialize(sys.argv)
	robot = moveit_commander.RobotCommander()
	scene = moveit_commander.PlanningSceneInterface()
	group = moveit_commander.MoveGroupCommander("left_arm")
	display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',moveit_msgs.msg.DisplayTrajectory)
	print "============ Generating plan 1"
	pose_target = geometry_msgs.msg.Pose()
	quaternion =tf.transformations.quaternion_from_euler(qx, qy, qz)
	pose_target.orientation.x = quaternion[0]
	pose_target.orientation.y = quaternion[1]
	pose_target.orientation.z = quaternion[2]
	pose_target.orientation.w = quaternion[3]
	pose_target.position.x = trans[0]
	pose_target.position.y = trans[1]
	pose_target.position.z = trans[2]

	group.set_pose_target(pose_target)
```

```
        plan1 = group.plan()

        print "============ Waiting while RVIZ displays plan1..."
        group.go(wait=True)
```

Progress and Timeline

Week 1

The first week was occupied mostly by installing and configuring ubuntu, ROS and packages required for Baxter. We faced lots of difficulties while trying to install everything however we managed to configure everything towards the end of the week. While trying to install the required programs and packages I completed tutorials about ROS from http://wiki.ros.org/ROS/Tutorials and learned ROS's concepts and mechanics.

Week 2

This week, we were assigned our main project: the Baxter UI. We started from the basics and tried to read the information published by the buttons. When we completed that task, we set one button to open and another one to close the gripper of the robot. By the end of the week, we were able to make the robot open or close its gripper and move on certain axis and we controlled these actions by individual buttons.

Week 3

We designed and made the images for the user interface. We started writing our main program and made it display the images and change according to the button states. Then, we made the robot able to perform the actions selected on the menu.

Week 4

This was our final week so we made final refinements and adjustments to our code. We added the rotate selection this week and therefore learned how to use tf and Rviz. We tested our code several times on the simulation to make sure that it worked perfectly and then tried it out on the actual robot.

Areas for improvement

Displaying the images

Each image of the menu was premade by us. Although this system makes the images more aesthetically pleasing and easier to code, it makes adding more options and changing the menu significantly more difficult. The ideal way to create this user interface would be to make our program create its own images therefore making adding options and features easier.

Navigating the menu

We mainly tested our program on a simulation of baxter and only tested it on the actual robot on our last day. We coded our program according to the values of one full rotation of the wheel on the simulation and one full rotation in the simulation was enough to navigate through all the options. However, when we tried our program on the actual robot, we found out that for one full rotation on the simulation, we needed to complete 9-10 rotations on the actual robot which made it extremely hard to navigate through options. We should have expected differences between the robot and the simulation and started testing our program on the robot earlier.

Conclusion

It is often hard for a researcher, working on a robot, to enter each command from their computers. Our program, Baxter UI, offers a solution to this by giving the option of controlling simple actions of Baxter from its screen. We think that our program will make conducting experiments on Baxter easier by making the researcher able to make the robot perform simple actions from its screen. The program can be improved in future iterations by making the wheel work better and making adding more options to the menu easier.