Care-O-Bot 3 Manipulation (Navigation & Grasping)

Mukul Suhas Bandodkar, MK Hasan, Shaon Debnath, *Group3.2*

***Abstract*—Mobile robots for the assistance of people in their home as well as factories and industries has been the need of the hour for quite a long time and will continue to increase tremendously in the mere future. Many big industries are trying to increase efficiency and productivity by trying to automate the production methods by using state-of-the art robots for manipulation, thus reducing the workload on humans. This paper describes a small step towards achieving a smart urban factory with the help of Care-O-Bot 3. This paper depicts the basic capabilities of the Care-O-Bot platform like perception, navigation and manipulation by building a demonstration application in which the robot is placed in the kitchen of an apartment. The robot then tries to fetch an object placed on a table by first navigating to it and then grasping it. The main aim of the project was to simulate the robot and the environment completely using ROS and Gazebo simulator and then test it on the actual robot.**

I. Introduction

Industrial robots are in the process of completely automating the manufacturing and thus revolutionizing it. They are taking on human capabilities and traits such as sensing, intelligence, memory and trainability. As a result, they are taking more jobs away from the humans like picking and packaging, testing or inspecting products or assembling minute electronics. A new generation of collaborative robots is developed in an era of shepherding robots out of their cages and literally hand-in-hand with human workers who train them through physical demonstration.

The most important question of the hour especially for small and mid-sized manufacturers is that if prices keep declining and capabilities of robotic technologies keep expanding, is it the right time to get some robots to solve the problem. Indeed, many have already answered this question. According a PwC survey of manufacturers, 59% of are already currently using some sort of robotics technology [1]. Robots are being deployed beyond traditional automated tasks in the auto industry---and are increasing ranks in sectors such as food and beverage and life sciences and doing work that requires dexterity and precision humans cannot achieve. The emergence of so-called collaborative robots is opening the door to greater human-robot collaboration, with robots leaving their cages and working literally hand-in-hand with their human colleagues.

This paper demonstrates the basic capabilities already available on the Care-O-Bot platform, which are perception, navigation and manipulation. The objective of the project is to build a demonstration application where the robot navigates in an environment and on receiving a request from the user, and fetches the object. The expected outcome is to simulate the application thoroughly and run it on the physical robot successfully.

The Care-O-Bot 3 was developed by Fraunhaofer IPA in the year 2008 [2], equipped with the state of the art technology. Care-O-Bot 3 had a manipulator with 7 DOF and a three-finger gripper which capable of grasping any object. It also has range and image sensors for object detection and 3D mapping of the environment. The Care-O-Bot 3 has many functionalities and has a simple and easy-to-use design.

Fraunhofer IPA has set up different environment and application scenarios on the Care-O-bot 3 implementation such as placing an order for a cold drink bottle, water bottle, any other drinks or object [3]. This robot can be used at home as well as any restaurant or bar. As an example, when a user places an order for getting a bottle of water, the robot will go to the kitchen. There might be many obstacles on the path the robot takes to reach the kitchen or the specified point, like a door, which robot should be able to identify. Then it should be able to detect the grasping points of the door handle and then should be able to open the door and go the kitchen.

After reaching to the kitchen, it will try to detect the requested object (water bottle) and if the bottle is found, the robot will pick it up and will place it on the tray. Then the robot should go back to the user and should serve the bottle. Currently the robot cannot identify the person who placed the order. It can only comprehend the location of user. As a result of which, it will just return to the location where it took the order from.

Many robots have been developed over the decade which are specialized for home and industrial use. The Care-O-Bot 1 and Care-O-Bot 2 which were developed in 1998 and 2002 respectively are examples of a few robots developed for industrial, manufacturing as well as home environments [4]. The first prototype was well known for its mobility functions whereas the second prototype was able to execute manipulation tasks autonomously.

This paper describes the simulation of the kitchen scenario described earlier in this section used for obtaining the required object from the specified location. The location from where Care o bot 3 will take orders from user is assumed to be in the vicinity of the requested object and it can detect the object from the specified location and move to the object without any collision (no door or any other obstacle).

This article is organized as follows. Section II deals with the related works. Section III explains the solution to the problem statement. It briefly gives the overall architecture of the system and also the architecture of the Moveit group detail. Section IV sums up the results of the project. Section V gives the conclusion and finally we end with the references.

II. Related Works

Grasping is a wide field in robotics and a lot of research and development is going in the industry to come up with more and more sophisticated robotic systems for varied tasks. Recently MIT developed a robotic hand which could recognize objects by feel [5]. Engineers at MIT built a three-fingered robotic hand that could identify and safely grasp delicate objects. The robotic hand was made of soft silicone with embedded sensors which would be trained to identify different objects. Similarly, a smart grasping system was developed by the Shadow robot company [6]. The grasper has built-in intelligence like torque sensing on each joint. The gripper has an in-built vision system which enables the hand to see what it is about to grasp making it very robust and reliable. Recently the Dex-Net 2.0 bot was developed by the researchers of UC Berkeley [7]. This robot can use Machine learning to grasp any type of unusual object. The robot uses a 3D sensor and a deep-learning neural network to identify and grasp an object.

III. Methodology

A. General Framework

As our project was to work with a service robot and manipulate it, after receiving command from user, care o bot 3 will move to some point where robot will keep safe distance from the object, will detect the grasping point of the object, will grab the object, hold it and keep on the tray deliver the object to user defined position. We can point out all the procedure in some points.

- Setup Robot Operating System (ROS) and install Care-O-Bot 3 libraries (Indigo version)
- Setup a simulating environment of Care-O-Bot using Gazebo and Rviz to visualize Robot movement and plan.
- Create our own package which will be integrated into the ROS environment and depending on the packages of COB Libraries, which will use different ROS NODES and TOPICS that will help us to recognized position and state of care o bot, arm joint state , etc.
- Navigate the Care-O-Bot to user defined position.
- Object recognition using 3D camera. Detect grasping point of the object based on shape of the object.
- Calculating the Inverse Kinematics for Arm Joint based on grasping point and planning for feasible movement of the arm.
- Check safety position, if Cob's position is safe to grab the object using its arm.
- Configuration of real Care-O-Bot to test our package and nodes after it works fine in simulator.

B. Prerequisites

1. Ubuntu 14.04 LTS (Trusty)
2. Ros Operating System, ROS-Indigo
3. Caffe
4. OpenCV
5. GPD (Grasp Pose Detection)
6. Moveit
7. Gazebo
8. Rviz

C. System Architecture and Specifications

The nodes and packages that are shown in figure is discussed in detail below:

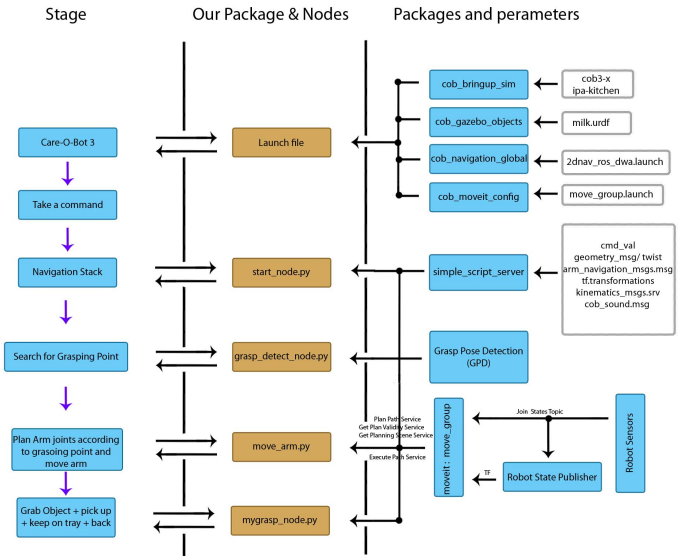

Fig. 1 System Architecture

1. **myrobot_launch_sim.launch:** This launch file creates a simulation with kitchen environment, Care-O-Bot 3 and an object on the table. To test in real Robot myrobot_launch.launch can be used. More over these, this launch file launches all those topics which will be required to manipulate COB properly. The other packages we are launching here are: 2d_nav_ros_dwa.launch and move_group.launch.

2. **start_node:** This node is used to initialize all the components of COB3. i.e torso, head, arm, gripper, base.To initialize or manipulate all component we need to communicate with several rostopic like geomatry_msgs/twist , arm_navigation_msg, tf.transform etc. and we need to pass some rosparam. Start_node communicate with them through simple_script_server. After initializing the components using simple_script_server, arm and tray

is taken to folded position, which are safe before navigating the base around. Then it should ask for it destination. We have set parameter for Table but you also can put any other coordinate in x,y format. After taking location in terminal, this node call move base to navigate COB3.

Simple_script_server: This is the care-o-bot python API provided by robot manufacturer which has all the method for manipulating the components of Care-O-Bot. It can be used with simulated robot or real robot.

Navigation: Navigation can be achieved by using Fraunhofer's global navigation package (cob_navigation_global). The package uses navigation stack's move_base node as well as DWA (Dynamic Window Approach) local path planning (dwa_local_planner). In DWA the controller gives velocity commands to the robot's mobile base. It performs forward simulation from the current state to predict the outcome of different velocity commands and scores the commands by multiple characteristics. The highest scored command is published to the mobile base.

3. **grasp_detect_node:** Traditionally, robotic grasping is understood in terms of two related subproblems: perception and planning. The perceptual component estimates the position and orientation (pose) of the object to be grasped. The planning component reasons about where/how to move the manipulator into a grasp configuration. This node is responsible for first subproblem which detects grasping point of an object. As the default object detection node by Fraunhofer IPA is not available for public use, we had to think about further alternative approach.There are number of grasp detection method that can be used to localize robotic grasp configurations directly from sensor data without estimating object pose. We selected a third party package called Grasp Pose Detector(GDP) . The methods of these package take as input a noisy and partially occluded RGBD image or point cloud and produce as output pose estimates of viable grasps, without assuming a known CAD model of the object. Although these methods generalize grasp knowledge to new objects well, they have not yet been demonstrated to be reliable enough to be used widely. More simply, This package detects 6-DOF grasp poses for a 2-finger grasp (e.g. a parallel jaw gripper) in 3D point clouds. Grasp pose detection consists of three steps: sampling a large number of grasp candidates, classifying these candidates as viable grasps or not, and clustering viable grasps which are geometrically similar. The possibilities of detecting grasping using GPD is mutiple, one is using RBGD camera and another one is with specific Object. You can use this package with a single or with two depth sensors. The package comes with weight files for Caffe for both options. You can find these files in gpd/caffe/15channels. For a single sensor, use single_view15_channels.caffemodel and for two depth sensors, use the channel called 15 channel two_views_15_channels_[angle]. The [angle] is the angle between the two sensor views, as illustrated the picture below. In the two-views setting, you want to register the two point clouds together before sending them to GPD.

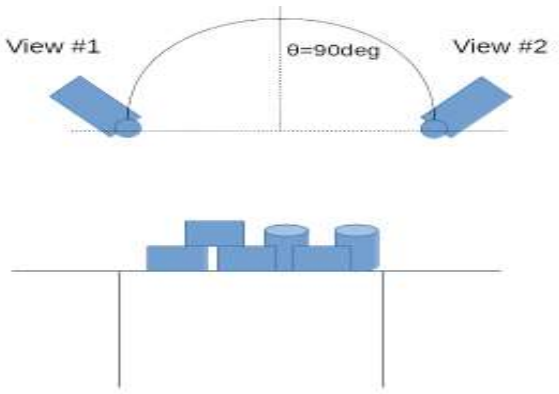

Fig 3: views of GPD [13]

To switch between one and two sensor views, change the parameter trained_file in the launch file launch/caffe/ur5_15channels.launch. For as input channels for neural network the package comes with weight files for two different input representations for the neural network that is used to decide if a grasp is viable or not: 3 or 15 channels. The default is 15 channels. However you can use the 3 channels to achieve better runtime for a loss in grasp quality. As our robot has 7-DOF grasp poses ability and it does not support this arm for implementing the grasping ability, so we only use the grasp pose detection system from this package.

[Note: This node is responsible for detecting grasping point of an object. We have used GPD packages to detect grasping point. As we did not finalized this code, this node is not available in our package. But anyone can start with this node by launching gpd tutorial1.launch , once this package is launched, cob3 will start to give grasping point. As an improvement you can detect the grasping point of a specific object.]

4. **move_arm:** This node is responsible for moving the arm to the grasping point. Before moving the arm, we need to place the arm in pregrasp position, which will make it easier to plan for joint of arm. And also, we need to "cylopen" the gripper. Once arm reached to pregrasp position with opened gripper, we need to plan all the arm joint position by calculating inverse kinematics according to the given grasping point, so that we can move the arm from pregrasp position to

grasping point position. We have used moveit_group package to calculate the value for each joint against grasping point. While it gets plan, COB3 will start to move its arm towards the grasping point and set the gripper to grip the object.

MoveIt Architecture

MoveIt! is chosen over the conventional arm_navigation stack which is used to integrate manipulators in ROS. MoveIt! Provides functions for collision-free motion planning like forward kinematics, inverse kinematics and collision monitoring. MoveIt! is chosen over the conventional arm_navigation stack which is used to integrate manipulators in ROS. MoveIt! Provides functions for collision-free motion planning like forward kinematics, inverse kinematics and collision monitoring.

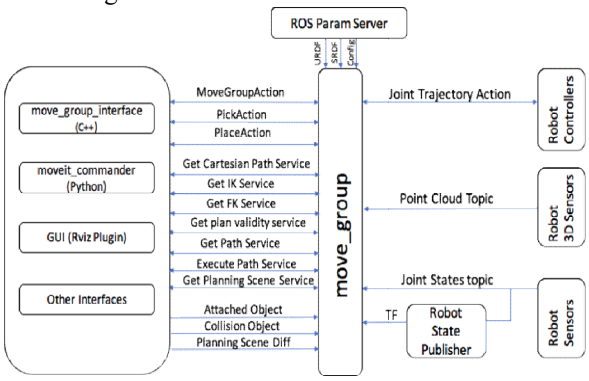

Fig. 2 MoveIt Architecture [12]

To use MoveIt! on a new robot, it must be in the Unified-Robot-Description Format (URDF). This is an XML format, in which a robot is modeled by using so-called links and joints. The initialization of the module is done by the MoveIt set up assistance. The MoveIt! Setup Assistance supports the definition of planning groups and creates the configuration files for the robot, such as the Semantic Robot Description Format (SRDF). It contains planning groups, end-effectors and others Information, such as the self-collision matrix.

The main component of MoveIt! is the move_group node. This block connects all the components and provides actions and services to the user. The access the move_group either in C ++ through the move_group_interface or in Python through the Moveit_commander to access the functionality of the node. The user can also use the graphical rviz motion planning plug-in for MoveIt. The ROS parameter sends information like the SRDF, URDF and the other configuration files to the move_group.

Communication between move_group and the robot takes place via ROS topics and actions. With the help of the joint_states topic, the controllers of the bot continuously transmit the current positions of the joints. The data of the 3D camera of the bot can be used to create live 3D maps of the surroundings, A so-called octomap. In this, the environment is represented by cubes of adjustable edge length and used to collide the robot with the environment avoid. If the movement of a component has been successfully planned its position is sent as FollowJointTrajectoryAction to the controller of the component. This is implemented by the ActionServer that supports the calculated trajectory on the robot

Various services are offered by the MoveIt group like forward or backward Inverse kinematics or a path defined by waypoints in the work area of the manipulator. Furthermore, there is the MoveGroupAction, to provide a collision-free trajectory from a start to a target configuration of the manipulator, as well as a pick and place action around items.

5. **mygrasp_node:** This node is used for Grasping the object, hold it, put it back on the tray and navigate to initial position. To grasp the object gripper needed to be “cylclosed” and arm need to be position “home” at first, then it can move toward tray. Gripper gets the method and parameter for gripping from simple_script_server. Once object is placed on the tray, arm should be folded before navigating to initial position.

D. Use and Misuse cases

The project makes use of OpenCV headers for object detection and shape estimation. Assuming the current position of the robot and the 3D localization of the object to be fetched, on receiving an input from the user, the robot base will navigate itself to the coordinates of interest. Since it is assumed that there will be no obstacles on the navigation path, the movement of the base from the current position to the point where the object is defined to be, won’t be very complicated. Once the robot base is approaching the object to be grasped it will have to define a safe are for grasping the object which will be a challenging task. Once the safe zone is decided and fixed the robot arm is all set to grasp the object under consideration. The arm of the bot has to now define a trajectory to be able to fetch the object, hold it and deliver to the position of interest without the object being broken or fallen on its way.

Determining a safe and correct position for the arm where it will be able to easily detect and grasp the object. A condition may arise wherein the bot has moved too close to the object and hence is not able to find a safe position to grasp the object. In such situations, the bot base will move using the move_it package to a position where it will be able to safely detect and grasp the object. The force with which the object has to be grasped so that it does not damage or break the object on its way to the desired location.

E. Errors and Solutions

During manipulation of gripper, some rospy errors may arise. To solve it simple_script_server.py is needed to be modified.

simple_script_server.py can be updated in following way (find it at /opt/ros/indigo/lib/python2.7/dist-packages/simple_script_server/) and inside the class calculate_point_time

Updated code[6]:

```
def calculate_point_time(self, component_name, start_pos, end_pos, default_vel):
    try:
        d_max = max(list(abs(numpy.array(start_pos) - numpy.array(end_pos))))
        point_time = d_max / default_vel
    except ValueError as e:
        print "Value Error", e
        print "Likely due to mimic joints. Using default point_time: 3.0 [sec]"
        point_time = 3.0    # use default point_time
        return point_time
```

IV. RESULTS

To successfully work and manipulate with Care-O-Bot we had to go through with some phases. We will discuss about our accomplishment in each phases.

Initialization of components: To manipulate different components i.e. torso, head, arm, gripper, base, tray of Care-O-bot 3, we need to initialize all of them. All of them were successfully initialized using simple_script_server.

Success rate: 100%

Manipulate components: We were able to manipulate all the components i.e. torso, head, arm, gripper, base, tray of Care-O-bot 3 using simple_script_server. Some common manipulation of predefined position is

Gripper move to cylclosed or cycopen
Arm move to folded or pregrasp
Head move to home
Torso move to home
Tray move to down

Success rate 100%

Navigation: We were able to navigate COB3 to any coordinate of generated map using simple_script_server. To plan the path to destination coordinate from current ordinate dwa planner has been used.

Success rate: 100%

Grasping point detection: Fraunhofer IPA has already developed package for object and grasping point detection named "cob_object_detection", which is not available for public but can be purchased.

So we have used third party package called GPD (Grasp Pose Detection).

With GPD we were able to detect grasping position of different objects using care-o-bot camera. COB3 detects grasping point of all available object through the robot Camera. Now we need to identify the grasping point of specific object among the grasping points of all objects.

We believe that we have scope to improve a lot here. We are suggesting to invest more time and effort.

Success rate: 50% [not complete yet]

Note: we put the grasping point manually

Planning arm move against grasping point: We were successful to calculate the 7 joint positions of arm by inverse kinematics using moveit_group package.

Success rate 100%

Grasp the object: After detecting the object and get grasping point, COB3 should try to grasp the object. But the object slips from the gripper.

To solve this issue, we took different type of approach

Approach 1:

We modified object urdf, dae file, where we tried by changing mass, rigidbody parameter, body_box to base_link.

Approach 2:

We tried to receive pressure parameter so that we can abort gripper cylclosed action. But pressure parameter not found for COB3

Approach 3:

We stopped gripper cylclosed action after seconds. But in this way, when next command run to take the object from table to hold position, Object slips again.

So we believe here we can put more time to solve this issue, so that COB3 can grasp the object successfully

Result Summary:

- Torso, gripper, head, tray manipulation works fine in Gazebo Simulator but Gripper was failed to initialize in real COB3-8 in Dai Lab
-
- 100% accuracy in navigation
- Grasping point detection using GDP is successful but not for the specific object.
- Arm Manipulation to grasping point using moveit succeed
- Improvement required in grasping point detection for specific object
- Improvement required to grasp the object

RQT Graph

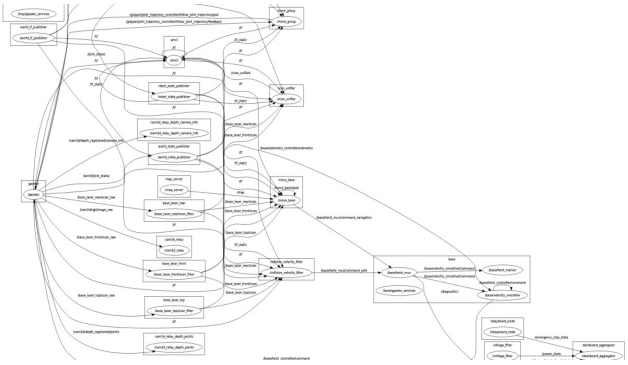

Fig. 3 RQT Graph

The RQT graph describes interconnection between the different ROS nodes/ topics used in the project as shown in Fig. 3.A high resolution RQT graph can be found at the following link:
https://gitlab.tubit.tu-berlin.de/aac-hrc/robot-grasping/blob/master/doc/rosgraph.svg.

V. CONCLUSION

Our group has tried to implement a working simulation that demonstrates the key features of Care-O-Bot 3, what this robot is capable of doing. The initial goal was to successfully run and manipulate the object using the robot in a simulation environment and also in real robot machine. Due to some unavailability of resources, we had to change the goal at the end of the project deadline. That means the workload has lessened quite a bit, which was really beneficial as some missing functionality of the manufacturer's software resulted in more work than expected. Due to lack of proper documentation of cob 3, we had to struggle to find out the missing package and changes which have been made by manufacturers. As the support for this robot is almost came to an end, it was hugely challenging to work with existed library and packages. The initialization of all the component was difficult as the name of this component was changed and not compatible with existed library, but we were able to do with the great amount of background research we have done. Some problems have been identified with the navigation of robot in simulation environment due to navigation planner and that was solved with finding out the missing package. The motion planning for arm and calculation of inverse kinematics has successfully implemented with a moveit package, though we had some initial problem of defining the Inverse kinematics plugin. Object detection had not been implemented fully by the manufacturer as there was some deprecated sample script available. Programming this missing functionality from scratch was deemed outside the scope of this project, as the capability of object detection did not exist for the robot. A dummy implementation has implemented where the robot tried to grasping from a known location with the predefined grasping point. We tried to use some third party Objection detection and grasping point detection library and packages like OpenCV, OpenRave, Caffe, and GPD. We had some success with GPD for finding the grasping point of the object by RGBD camera of a robot but the constraints of time have blocked our further achievement. Due to the grasp plugin bugs in simulation environment Gazebo, the robot gripper was not able to grasp the object properly in spite of getting the grasping point. the grasping point was provided manually and Inverse kinematics calculation was perfect for the arm with given grasping point. That was gazebo physics bug which we tried to solve but could not actually. It could be the problem with URDF file as well but it is still vague. The gazebo environment was too slow and one solution to make it faster was by lowering the resolution of gazebo and rviz. Some research has been made but we could not found the good solution.
After able to completion of successful simulation we tried to work with main cob machine, we were very able to successfully run our packages on the main machine. We navigate the robot and initialize every component except the gripper. As there was some problem exists with robot gripper.

Finally, the simulation environment was the kitchen of the office and where the Care-O-Bot acts as a service robot. As we have worked and implemented most of the features, this robot is capable of doing. The Object perception and manipulation were left for future projects to solve when we will be able to get the proper packages from the manufacturer or we will build our own package instead. For proper Human-Robot Interactions protocols, these above-mentioned problem is really a big hindrance and the solution is must to have. We will continue to work with this problem and we will integrate all of these to main Care-O-Bot in near future.

REFERENCES

[1] PWC: Pulse on robotics https://www.pwc.com/
[2] "Robotic Home Assistant Care-O-bot - Fraunhofer AAL." https://www.aal.fraunhofer.de/content/dam/aal/en/doc/2009_robotic_home_assistant_careobot_%203_product_vision_and_innovation_platform.pdf.
[3]"Care-O-bot® 3." https://www.care-o-bot.de/content/dam/careobot/en/documents/productsheets/Product%20Sheet_Care-O-bot%203.pdf.

[4] "Care-O-bot II—Development of a Next Generation Robotic ...." https://link.springer.com/content/pdf/10.1023/B:AURO.0000016865.35796.e9.pdf.
[5] Soft robotic hand can pick up and identify a wide array of objects | MIT ...." 30 Sep. 2015, http://news.mit.edu/2015/soft-robotic-hand-can-pick-and-identify-wide-array-of-objects-0930.
[6] "The Shadow Smart Grasping System - Shadow Robot Company." 2 Nov. 2016, https://www.shadowrobot.com/shadow-smart-grasping-system/.
[7] “Meet the most nimble-fingered robot ever built | Berkeley News." 26 May. 2017, http://news.berkeley.edu/story_jump/meet-the-most-nimble-robot-ever-built/.
[8] ROS Indigo:http://wiki.ros.org/indigo/Installation/Ubuntu
[9] ROS Environment Check: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
[10]Create WorkSpace: http://wiki.ros.org/catkin/Tutorials/create_a_workspace
[11] COB Indigo setup: http://wiki.ros.org/Robots/Care-O-bot/indigo
[12] Moveit Install: http://moveit.ros.org/install/
[13] GPD (Grasping Pose Detection): https://github.com/atenpas/gpd
[14] OpenCV: http://opencv.org/