Software Design Specifications of **Robot Manipulation (Grasping)**

**CONFIDENTIAL!**

|  |  |
| --- | --- |
| Version number | Version 1.0 |
| Distribution Group | Internal |
| Main responsibleCoordinator | DAI-Labor, TU-Berlin |
| Current State Date | - |

Advanced Agent Competition: Human Robot Collaboration and Teamwork –SS 17

Team Members: MukulBandodkar, M.K. Hasan & ShaonDebnath

# Table ofContents

[Table of Contents 2](#_Toc481682342)

[List of Figures 3](#_Toc481682343)

[List of Tables 4](#_Toc481682344)

[1 Introduction 5](#_Toc481682345)

[1.1 Motivation and Problem Definition 5](#_Toc481682346)

[1.2 Objectives 5](#_Toc481682347)

[1.3 General Constraints 5](#_Toc481682348)

[2 State-of-the-Art 5](#_Toc481682349)

[3 Requirements 5](#_Toc481682350)

[4 Use cases / scenarios 6](#_Toc481682351)

[5 System Architecture and Specifications 6](#_Toc481682352)

[5.1 System Architecture 6](#_Toc481682353)

[5.2 System Specifications 6](#_Toc481682354)

[6 Timeline 7](#_Toc481682355)

[7 Frameworks and Tools 7](#_Toc481682356)

[References 8](#_Toc481682357)

# List of Figures

Figure 1. System Architecture 10.

# **Introduction**

## Motivation and Problem Definition

Industrial robots are on the verge of revolutionizing manufacturing. As they become smarter, faster and cheaper, they’re being called upon to do more. They are taking on more “human” capabilities and traits such as sensing, dexterity, memory and trainability. As a result, they are taking on more jobs - such as picking and packaging, testing or inspecting products, or assembling minute electronics. Also, a new generation of “collaborative” robots ushers in an era of shepherding robots out of their cages and literally hand-in-hand with human workers who train them through physical demonstration. Especially for small and mid-sized manufacturers, a question is arising sooner than most probably expected: “If prices keep declining and capabilities of robotic technologies keep expanding, is now the time to hire some automated help?” 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.

The goal of the project is to demonstrate the basic capabilities already available on the Care-O-Bot platform: perception, navigation and manipulation. Our task 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.

## Objectives

In this project, our objective is, after receiving command to grab any object, care o bot 3 will move to the nearest point of the object, will detect the grasping point of the object, will grab the object and pull it up and will drop it somewhere else. 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 check the code and to check how it works
* Create our own project 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 and arm joint state.
* Object recognition using 3D camera. Detect position and shape of the object.
* Detect the point close to the Object and move the Care-O-Bot 3 to that point.
* Planning for Grasping point of the object.
* Check safety position, if Cob’s position is safe to grab the object using its arm.
* Configuration of real Care-O-Bot to test the project after it works fine in simulator.

## General Constraints

This project makes these assumptions:

* The current position of the robot will be defined.
* The grasping coordinates of the object are specified.
* No obstacles are present in the path from the current position to the grasping position.
* Object shape, size and type will be detected by the “Object Recognition” team.
* Object shape would be such that it would be held by one arm itself.

# **State-of-the-Art**

Robotic grasping is one of the most widely explored areas of manipulation. Teaching robots this skill can be tricky, because there aren’t necessarily obvious connections between sensor data and actions, especially if you have gobs of sensor data coming in all the time (like you do with vision systems). A cleverer way to do it is to just let the robots learn for themselves, instead of trying to teach them at all. Most work in robot manipulation assumes availability of a complete 2-d or 3-d model of the object, and focuses on designing control and planning methods to achieve a successful and stable grasp. Here, we will discuss in detail prior work that uses learning or vision for robotic manipulation. Broadly, grasping methods can be categorized as geometrically driven and data-driven. Geometric methods analyze the shape of a target object and plan a suitable grasp pose, based on criteria such as force closure (Weisz & Allen, 2012) or caging (Rodriguez, 2012). These methods typically need to understand the geometry of the scene, using depth or stereo sensors and matching of previously scanned models to observations (Goldfeder, 2009b).

Data-driven methods take a variety of different forms, including human-supervised methods that predict grasp configurations (Herzog, 2014; Lenz, 2015) and methods that predict finger placement from geometric criteria computed offline (Goldfeder, 2009a). Both types of data-driven grasp selection have recently incorporated deep learning (Kappler, 2015; Lenz et al., 2015; Redmon & Angelova, 2015). Feedback has been incorporated into grasping primarily as a way to achieve the desired forces for force closure and other dynamic grasping criteria (Hudson, 2012), as well as in the form of standard servoing mechanisms, including visual servoing (described below) to servo the gripper to a pre-planned grasp pose (Kragic & Christensen, 2002). The method proposed in this work is entirely data-driven, and does not rely on any human annotation either at training or test time, in contrast to prior methods based on grasp points.

At Google Research, a team of researchers, with help from colleagues at X, tasked a 7-DoF robot arm with picking up objects in clutter using monocular visual servoing, and used a deep convolutional neural network (CNN) to predict the outcome of the grasp. The CNN was continuously retraining itself (starting with a lot of fail but gradually getting better), and to speed the process along, Google threw 14 robots at the problem in parallel. This is completely autonomous: all the humans had to do was fill the bins with stuff and then turn the power on. With 14 robots all working on this problem, a lot of data get collected a lot faster, but at the same time, a lot of unintentional variation gets introduced into the experiment. Cameras are positioned slightly differently, lighting is a bit different for each robot, and each of the compliant, under actuated two-finger grippers exhibits different types of wear, affecting performance. The upside to this is that the robots end up with a tolerance for things like minor hardware variation and camera calibration differences, making the grasping as a whole more robust. Even so, this method can’t be generalized too much, and is unlikely to work on significantly different hardware or in different grasping environments (like trying to pick stuff up off of a shelf). In future work, the researchers plan to explore increasing the diversity of the training setup to see how much more adaptable their technique can get. They’d also like to investigate how this method could be applied to “real world” robots that are “exposed to a wide variety of environments, objects, lighting conditions, and wear and tear.”

One of the student from TU Darmstadt proposed the continuum Gaussian bandits (CGB) policy for learning to grasp. While most grasp learning methods focus on supervised learning approaches, they frame grasping as a continuum-

armed bandits problem and use a reinforcement learning approach. In this manner, the robot actively selects grasps to evaluate and uses the outcomes of these grasps to select better grasps in the future. The robot can thus autonomously improve its grasping abilities over time, and optimize grasps for specific objects. The approach was inspired by the upper confidence bound (UCB) policies used for multi-armed bandits problems. The resulting algorithm is closely related to the UCB policies used in Bayesian optimization. However, rather than performing a global search, the proposed method performs a sample-efficient local search for optimal grasps. This approach is well-suited for learning grasps without relying on explicit information regarding the object’s shape and size. By applying this approach, the robot was able to learn dexterous grasps that would have been difficult to program by hand.

Fraunhofer IPA set up different environment and application scenarios on the Care-O-bot 3 implementation such as placing an order for a Cola Can, water bottle, any other drinks or object. This robot can be used for service in home or any restaurant or bar. As an example, when a user will place an order say a bottle of water, the robot will go to the kitchen. During this path there may have door, which robot should be able to identify and by detecting grasping point of the door handle it 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. But the till now limitation is Robot cannot identify the person who placed the order. It only can understand the location of user. Therefore, when it will go back to the user to serve the bottle, It actually goes back to the location from where it took the order.

In our project we tried to simulate part of this process. We assume that the location from where Care o bot 3 will take order from user, it will be in small distance from the requested object. And robot can detect the object from this location and move to that object without any collision (no door or any other obstacle).

# **Requirements**

* COB will detect the object and locate it using embedded 3D camera. OpenCV and point cloud library will be used for object pose and shape estimation – more specific models of the environment and robot.
* To navigate Care O Bot 3 from the current position to the co-ordinate near to the object, cob\_navigation\_global will be used.
* COB will detect the grasping point using haf\_grasp and agile\_grasp package.
* Using Moveit libraries COB will plan for grasping path and validate it by safety checking using Robot sensor.
* If the plan path is validate, Moveit will be used to move it’s arm properly to grab the object and keep it somewhere else. robot\_state\_publisher and join\_state\_topicto determine the current position of every joint of arm to move the arm properly.
* Robotics Operating System Indigo in Ubuntu 14.04 will be used to setup the environment.
* To check functionality Gazebo and RVIZ will be used as a simulator
* For simulation robot = cob3-2 and robot\_env = ipa-kitchen is used
* To create the object, urdf model of milk box, which can be found in cob\_gazebo\_world is used.

# **Usecases / scenarios**

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 we have 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.

**Misuse cases**

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.

# **System Architecture and Specifications**

## System Architecture

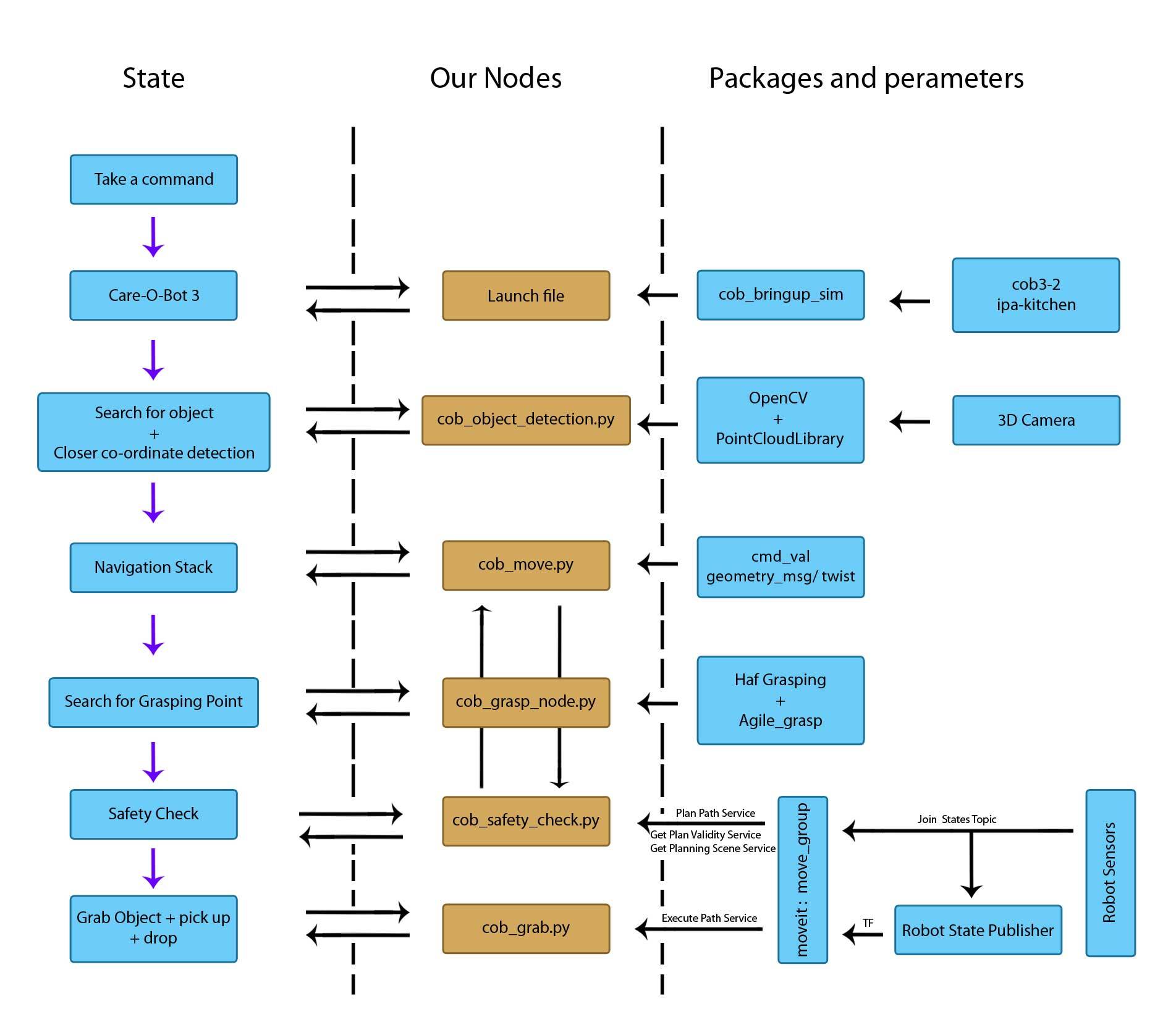


Figure 1: System Architecture / Flow-Chart Diagram

## System Specifications

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

1. **myrobot\_launch\_sim.launch:** This launch file creates a simulation with kitchen environment, Care-O-Bot 3 and an object on the. To test in real Robot myrobot\_launch.launch can be used.
2. **cob\_object\_detection:** This node is used to detect the object using the data received from 3D camera, using the libraries OpenCV and Point Cloud Library. ROS OpenCV, a library of algorithms for image processing is used to detect object surface from an unstructured environment and recognize the object.

Once object is detected, it will detect a co-ordinate, which is close to the Object so that COB can move to that position.

1. **cob\_move:** This node is used to navigate the COB in Omni direction according to the co ordinate received from cob\_object\_detection.

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.

The correct launch commands after starting roscore and bringing up the robot and environment were:

* -  roslaunchcob\_navigation\_global 2dnav\_ros\_dwa.launch
* -  roslaunchcob\_navigation\_globalrviz.launch

Using the packages required installation of the dwa\_local\_plannerpackage:

- sudo apt-get install ros-indigo-dwa-local-planner

To use the tutorial packages, the cob\_tutorials-indigo\_devpackage should be downloaded from Github, placed in the src folder of the catkin workspace and then built using catkin\_make. The script server can be used to easily give commands to the move\_baseas well as the arms, torso and grippers of the robot. To use the python API the target locations need to be defined in navigation\_goals.yaml, which can be found in

/opt/ros/indigo/share/cob\_default\_env\_config/ipa-kitchen

The scripts should then be placed in

.../catkin\_ws/src/cob\_tutorials-indigo\_dev/cob\_script\_server\_tutorial/scripts

When the script server node is launched, it reads the navigation goals from file and uploads them as parameters so they can be used by the python scripts. So to run a self-made script the following commands are required:

* -  roslaunchcob\_script\_serverscript\_server.launch
* -  rosruncob\_script\_server script\_name.py

1. **cob\_grasp\_node:** This node is used for Grasping and Manipulation

Grasping packages in general which could be used are as follows:

1. agile\_grasp
2. haf\_grasping
3. handle\_detector
4. mini\_max
5. atenpas/grasp\_selection

The best way to implement the grasping of the bot would be to use the package cob\_grasping.

We could also use a third-party software called OpenRAVE to calculate grasps that achieve force closure on different items. On a basic level, OpenRAVE takes as parameters the item to grasp and the gripper that does the grasping, then attempts to find stable grasps and saves the joint configurations of the gripper for said grasps which can then be published to ROS.

The lack of operational object detection results in that the robot would have been able to only to grasp items that have pre-defined stable grasps or models compatible with OpenRAVE for which grasps with force closure could be calculated.

1. **cob\_safety\_check:** This node is used to check if the position of COB is safe enough to grab the object in it’s grasping point. If it detects that from the current position COB cannot grab the object, it uses cob\_move to relocate its position. For this safety checking moveit is used. In Moveitmove\_group node is main component which combines all the required components and provide the required services. Using this we can plan for the arm path, can check the validity if it the path is safe. For this we can use moveit\_graspsindigo-devel which can be downloaded from

<https://github.com/davetcoleman/moveit_grasps/tree/indigo-devel>

1. **cob\_grab:** This node is used to grab the object through it’s grasping point and drop it on other location. This node is responsible for moving the arm to proper position. Moveit is also used in this node to execute the validate plan path which approved in safety check node. Communication between robot and move\_group is done by ROS Topics and Actions. Move\_group uses robot\_state\_publisher and join\_state\_topicto determine the current position of every joint of arm to move the arm properly.

# **Timeline**

Tentative timeline for this project is given below:

|  |  |
| --- | --- |
| **Date** | **Milestone** |
| 17.05 – 30.05 | Move the Care O Bot to a fixed location  Node: cob\_move.py  Using: geometry\_msg/twist |
| 31.05 – 13.06 | Milestone 2 :Searching for Grasping point and move to that location  Nodes: cob\_grasp\_node.py and cob\_move.py collaborate  Using: Haf\_grasping, Agile\_grasp |
| 14.06 – 28.06 | (Intermediate milestone)  Grab the object and pick it up  Node: cob\_grab.py  Using: Moveit\_grasps indigo-devel |
| 28.06 –11.07 | Milestone3: drop it , (Safety check, fine tuning) and Check in Real Robot. |

# **Frameworks and Tools**

For this project we are using Robot Operating System (ROS) in Ubuntu 14.04 Trusty.

Other Framework:

* Catkin Project
* Care-O-Bot library indigo
* To move the Care O Bot 3 in Omni direction we are using:
  + Cmd\_val, cob\_2dnav
  + Geometry\_msg/twist
* To search for grasping point:
  + Haf\_grasping
  + Agile\_grasp
  + Moveit\_grasps indigo-devel
* OpenCV will be used for image processing tools
* Gazebo and Rviz will be used as simulator

# 

# **References**

[1] PWC: Pulse on robotics <https://www.pwc.com/>

[2] ROS Indigo Install: <http://wiki.ros.org/indigo/Installation/Ubuntu>

[3] Care-O-Bot 3 library install: <http://wiki.ros.org/Robots/Care-O-bot/indigo>

[4] Haf Grasping: <http://wiki.ros.org/haf_grasping>

[5] Agile\_Grasp: <http://wiki.ros.org/agile_grasp>

[6] Moveit\_grasp indigo devel: <https://github.com/davetcoleman/moveit_grasps/tree/indigo-devel>