HERIOT WATT UNIVERSITY

ROBOTICS SYSTEM SCIENCE

B31YS

Husky Project: *Pass the Coffee*

Authors:
Jack ROME H00219766
Shayne SHAW H00214626

December, 2018

CONTENTS

1. INTRODUCTION AND OBJECTIVES

The goals of this project were to implement the topics taught this semester while using the Husky Dual ur5 robot from Clearpath Robotics. As such we have chosen to implement a Hazardous Material Disposal robot which could be used to autonomously locate and dispose of a hazardous substance to reduce the risk to humans. As a proof of concept, a coffee cup located inside the lab will be used to represent the hazardous material which needs to be disposed of within the waste bin in the lab. The project was broken down into sub-tasks, which were then assigned to individual members within the group.

1. Locate a specific object in the environment (In this case a coffee cup)
2. Map the environment if possible
3. Determine the location of this object in the map
4. Navigate the map until the robot is in proximity to the object
5. Manipulate the arms to grip the object
6. Pick up and carry the object
7. Navigate to a second pre-set location and drop the object, ideally in the bin

Fig. 1. Target object: Coffee cup

These seven goals were then classified into 3 fields: Camera

- Locate object – use colour space filtering and/or image recognition.
- Calculate coordinates of the object - using stereo camera binocular disparity.

Control/Navigation

- Map the environment – using gmapping or possibly SVO.
- Navigate the environment, avoiding any obstacles – SLAM, either Monte Carlo localization, custom Dynamic window or custom Potential Field Navigation.

Manipulation (UR5 arms)

- Manipulate the arms – making use of the Universal robotics Moveit package for the UR5.

2. TRIALED SOLUTIONS

A. Locating the target

One of the solutions attempted to locate the object was using the 'find object 2d' package. This package takes images of the target and uses feature-based methods to identify the target from an input stream topic. This works best at close range and with high-resolution input. Best results are also achieved by using numerous input images of the target, at different perspectives.

Subsequently, the 'find object 3d' package could be used alongside this to calculate the target's position in 3D space. It was found however, that this package was designed for cameras similar to the Kinect as it requires a depth topic and so it was not usable with the Bumblebee2 camera, unless depth was pre-calculated using stereo imaging.

B. Control & Navigation

One of the files contained in the Husky packages is the Husky teleop, written in python. This allowed us to develop an understanding of the topics controlling the movement of the UR5 arms, the Husky Base, the grippers and the

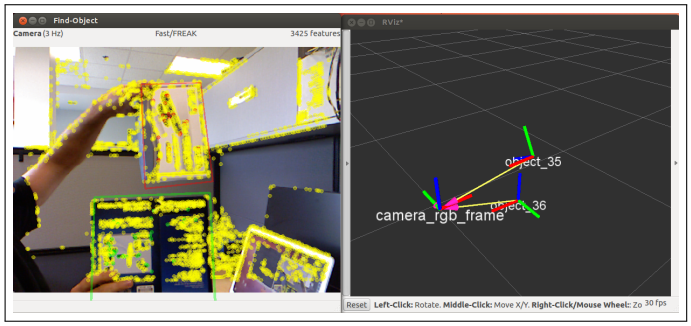

Fig. 2. Find object 2d package in operation

FLIR camera mount. The file is written to be controlled with keyboard commands and so it was simple to edit and test the control different parts of the robot using the keyboard.

The FLIR camera mount can be manipulated through pan and tilt commands. A demonstration was constructed from the Husky teleop file. This demonstration takes pan commands as a float argument (in radians) and publishes the corresponding message to the FLIR topic to pan the camera to the stated position.

C. Manipulation

A demonstration package could be launched on the Husky by using the instructions within the manual. This package activated the planner for the UR5 arms on the Husky and a UI in Rviz. The package allowed the user could visualize the Husky robot with correct arm positions in real-time and set goal states for these arms by dragging them to the desired positions in Rviz. If a successful plan was found by the planner, the UR5 arms could move itself to the desired state without internal or external collisions.

This however, identified a few problems with the Husky. The planner would often fail due to a difference between the ROS time-stamps. When the planner fails to produce a plan, no action is performed. Another problem arises due to the difference between the real-life Husky robot and the URDF model on the Husky computer. One major difference that is the height of the velodyne LIDAR sensor is moved up. This heigh difference caused inaccuracies in the /scan readings and caused the planner to create motion paths that caused internal collisions.

3. RESULTS

A. Locating the target

A.1. Calibration

The camera was calibrated using the ROS stereo calibration package[1]. This required the use of a checked board borrowed from the Oceans Systems Lab. Once calibrated, the various matrices for the stereo camera (such as the rotation matrix and intrinsic matrix) were saved to a temporary file and can be viewed in the topics /bumblebee2/left/calibrations and /bumblebee2/right/calibrations. Matrices used within the programs were copied into the code as using a subscriber and callback would be unnecessary as these parameters are constant.

A.2. Target Identification

The target (i.e., the cup) was found using the HSV colour space. A python program was used to estimate the HSV lower and upper limit of the target and then the input streams from the left and right camera of the bumblebee was masked so that only pixels within this HSV range can be viewed. The mask was eroded and dilated to remove noise in an attempt to work with large chunks of pixels. OpenCV filters were then used to estimate the centre coordinate of what remained in the masked image. There are two methods experimented with to estimate the centre coordinate of the target on the image streams. The first was by using HoughCircles. These are used to calculate circles within the masked image. The largest circle is assumed to be the target and the rest are disregarded as noise. HoughCircles are calculated using OpenCV functions that return the centre coordinates of the circles found, as well as the radii, as a matrix. The largest is stored and used to draw a circle onto the image. The centre coordinates were used for image projection calculation and radius was used to calculate the depth using the pinhole camera model.

Using HoughCircles was not found to be accurate or robust unless the target was well lit and spherical shaped. The alternative was to use moments. Another OpenCV function was used to return the moments of the masked images. These were centre coordinates of a bounded object on the image. This had more consistent results than the HoughCircles and so were used for the rest of the development.

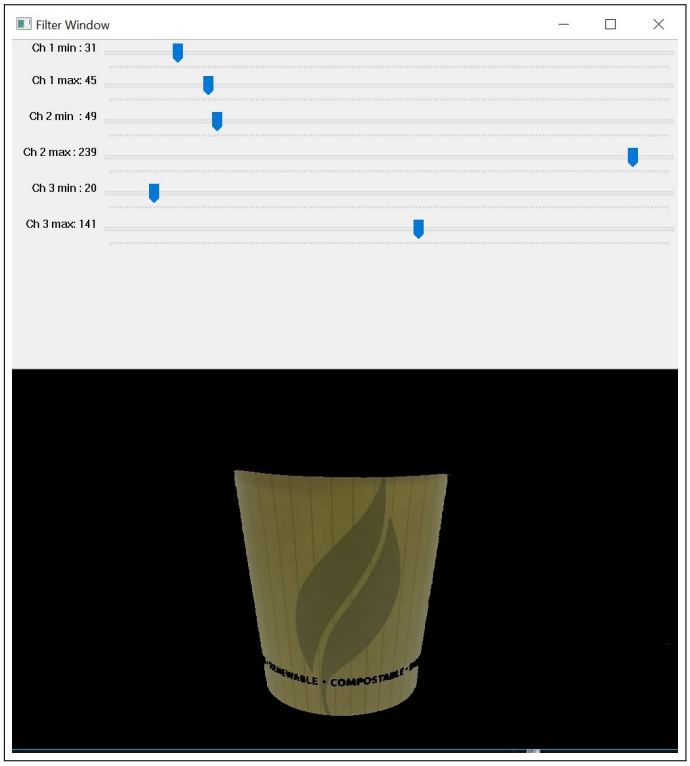

Fig. 3. Screen-shot of program used to gather the HSV range

A.3. Estimating Location

Once the pixel coordinates of the target have been found for the images from the left and right camera stream, an estimate of the location of the cup in 3D space is to be made. This was done using 2 methods. The first method was to estimate the distance from the cameras by using pinhole trigonometry. This uses a known target measurement, such as height in millimetres, as well as the focal length from the calibrated camera.

$$Depth = \frac{FocalLength * Height}{PixelRadius}$$

With the depth of the left and the right camera calculated, the 2D location of the target was estimated using the motion matrix and the camera's rotation relative to the robot. The

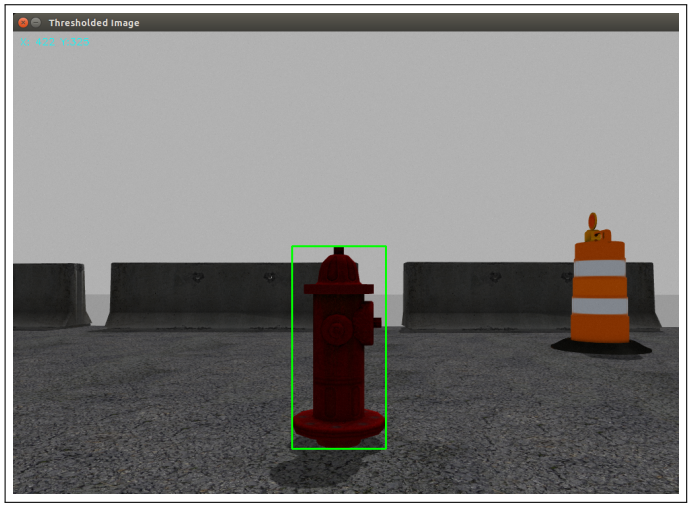

Fig. 4. Screen-shot of image stream within a simulation where the fire hydrant is the target

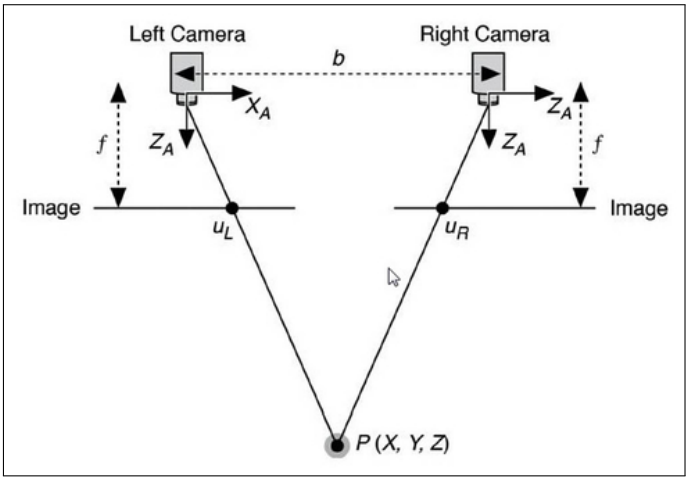

Fig. 5. Stereo trigonometry model used for 3D point estimation

depth is used to calculate the target's local coordinates in reference to the robot and the motion matrix is used to convert this to a global coordinate. For each camera, left and right:

$$x' = d * cos(\theta_t)$$

$$y' = d * sin(\theta_t)$$

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + \begin{bmatrix} -cos(\theta_t) & sin(\theta_t) \\ sin(\theta_t) & cos(\theta_t) \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Where d is the depth calculated previously, x and y are the camera coordinates at state t and θ_t is the FLIR mount pan angle at state t. Another method was using stereo trigonometry to estimate the 3D location of the target. The rotation matrix is used for each camera to obtain the projection points. These are then used to triangulate the 3D displacement of the points, relative to the camera, using OpenCV. These are returned as a homogeneous vector and so an OpenCV function is used to convert it to a Cartesian vector. The OpenCV function is used as it performs the operation efficiently

The 3D positions are transformed into point messages and are published to custom topics so that they can be visualized in RViz. The 2D points are used to publish a point to the nav_goal topic. To summarize, the steps to calculate the 3D location of the target include:

- find the coordinates of the centroid of the target in the left and right camera
- find the projection of these coordinates using the projection matrix
- triangulate these points to a single point using the rotation matrix
- convert to Cartesian point from homogeneous

B. Control & Navigation

B.1. Husky Movement

The robot features a few launch files to run gmapping demos and SLAM explorations. To make use of the gmapping demo, the estimated 3D position of the target was published as a 2D navigation goal through the move_base/goal topic. With the goal in place, the Husky relies upon its readings from the /scan topic to autonomously navigate to the goal whilst building a map of its surroundings. This worked best when the Velodine 3D LIDAR was separated and published to a different topic from the SICK LIDAR. Initially, they were both publishing to the same topic and so this caused faults in the navigation.

B.2. Camera Control

The FLIR mount is used in conjunction with the object tracking script. If the target is not

identified, the FLIR mount can be used to pan or tilt the camera until it is found. It was found that this works better by sending the movement messages at a low increment and frequency for a smoother control. This was tested, however, a robust method of control has not been found and so only a demo of the searching has been developed and so there is not a full integration.

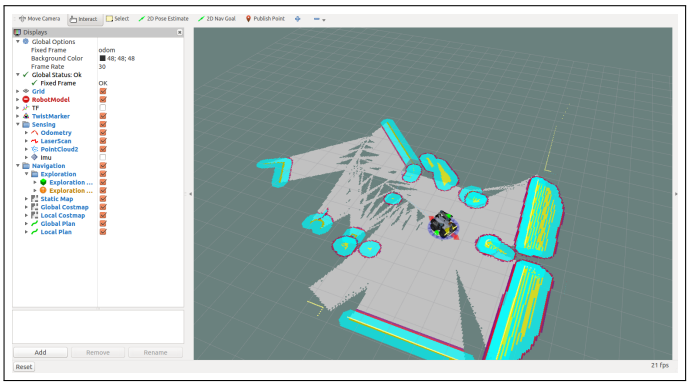

Fig. 6. Husky SLAM in operation

C. Manipulation

A custom Moveit package was created following the procedure outlined in the ROS Ignite manipulation course. Five individual robot poses were created for the right arm to assist with the goals of the project and the end effector was specified to be the gripper. The order of states for picking up an object were as follows:

- Start - robot's rest position
- Reach - reach down to a rough are of the object with the gripper open
- Close - adjust for accuracy and close the gripper on the target object
- Carry - Close the gripper and lift the object
- Open - Reach towards the goal position of the target in the gripper and release

C.1. Object interaction

As the arms were not able to be successfully moved by scripting, object interaction was not possible outside of simulation. The arm was able to be moved by suing Moveit Rviz and the relevant planner.

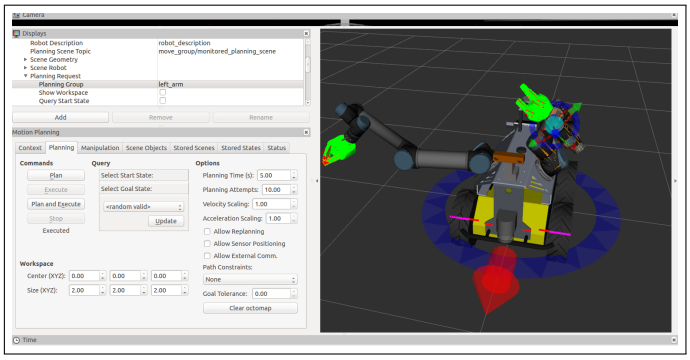

Fig. 7. Screenshot of Rviz MoveIt planner control

4. DISCUSSION

A. Performance Review

A.1. Identifying the target

It was found that colour isolation was not a robust method for locating the cup, especially in a busy environment, such as the labs. A better method would be through using feature-based object tracking using methods like FAST, BRIEF, SURF or SIFT. This could be achieved through the use of the find_object_3d package and using the stereo_image_proc to provide the depth or by using the newly installed depth camera to provide the depth stream. Depth could also be calculated using OpenCV functions. This would also allow for more than one object to be the target of the algorithm and for different goal states to be planned.

A.2. Control & Navigation

Within the ROS ignite 'Robot Perception' tutorials, a demonstration can be found where the Tiago Robot performs surface detection. The robot will navigate to the table/surface that it has perceived and use the find_object_2d package to identify the objects. This is useful as feature-based object detection performs best when the objects are at high resolution and so far away objects are not as easily perceived. This is a reason why it was decided to use colour-based detection to initially estimate the location of the object as we planned for the robot to navigate to a rough area before using the more advanced methods.

A.3. Manipulation

By using the Velodyne 3D LIDAR, the height of the surface that the target cup rests upon can be estimated. This, combined with the ROS ignite surface detection, will make the 3D state of arm to be in prior to gripping the target easier to estimate. The Velodyne scan has been seperated from the SICK laser scan to a different topic, however, adjustments to the URDF files and other such inaccurate configuration files need to be made to eliminate the differences between the virtual model and the real life robot for the Velodyne to operate correctly.

B. Suggestions moving forward

1. Update the URDF file on the husky to take into account the additional height of the Velodyne sensor.
2. Add a USB port to allow fast transfer of large files without the need to do this through SSH.
3. Running code directly off the husky as opposed to through SSH might solve the bandwidth clogging and slowing of Rviz when the Velodyne or Bumblebee camera are being streamed.
4. Fix the clock synchronization issue.
5. Create a Github or similar repository and save code that is known to work from each group with a short description of what it does and how to implement it.
6. Add to the repository a list of known errors and any solutions which have been discovered.
7. Create a few small demos for each major component or function on the husky to aid in familiarizing new users with their functionality and basic operation. Movement, arm manipulation, camera movement...etc.
8. If multiple groups are to work on the husky simultaneously, create allotted time slots when that group is able to have sole use of the robot for testing purposes to prevent monopolizing of robot time.

5. REFERENCES

- The Construct. (2018). ROS Tutorials: ROS Perception in 5 Days | Robot Ignite Academy. [online] Available at: http://www.theconstructsim.com/ros-tutorials-unit-0-intro-ros-perception-5-days/ [Accessed Dec. 2018].
- Wiki.ros.org. (2018). find_object_2d - ROS Wiki. [online] Available at:

http://wiki.ros.org/find_object_2d [Accessed Dec. 2018].

- Docs.opencv.org. (2018). Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation. [online] Available at: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html [Accessed Dec. 2018].
- Wiki.ros.org. (2018). stereo_image_proc - ROS Wiki. [online] Available at: http://wiki.ros.org/stereo_image_proc [Accessed Dec. 2018].
- Clearpathrobotics.com. (2018). Husky Dual UR5 Mobile Manipulation Demo (Simulation) — Husky UGV Tutorials 0.5.1 documentation. [online] Available at: http://www.clearpathrobotics.com/assets/guides/husky/HuskyDualManip.html [Accessed Dec. 2018].

6. CODE AND VIDEOS

The code created during this project can be found on this page:

https://github.com/jrome5/Husky-Code.git

Code for the camera streams was adapted from:

https://github.com/Elucidation/StereoColorTracking

Code for the HSV range finder can be found on:

https://github.com/alkasm/cspaceFilterPython

7. APPENDIX

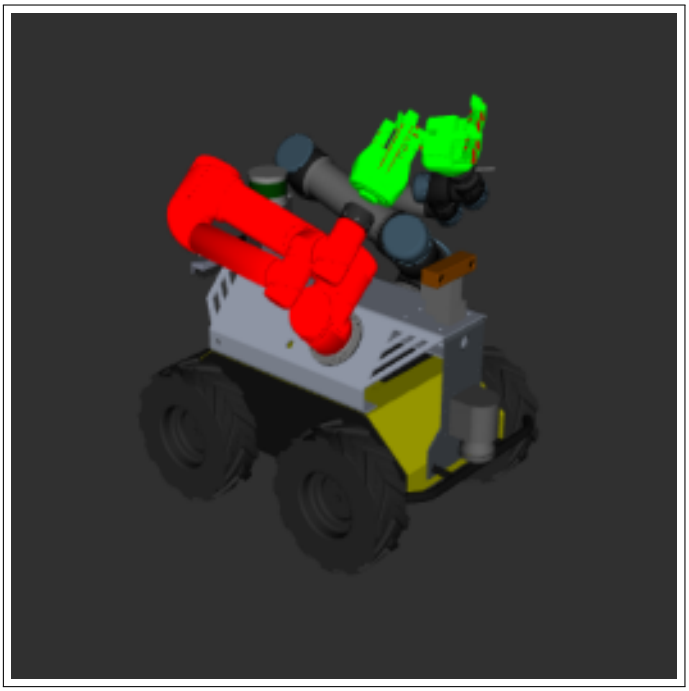

Fig. 8. Start - to be used as a default start location until the robot is in proximity to the object and use of the arm is required

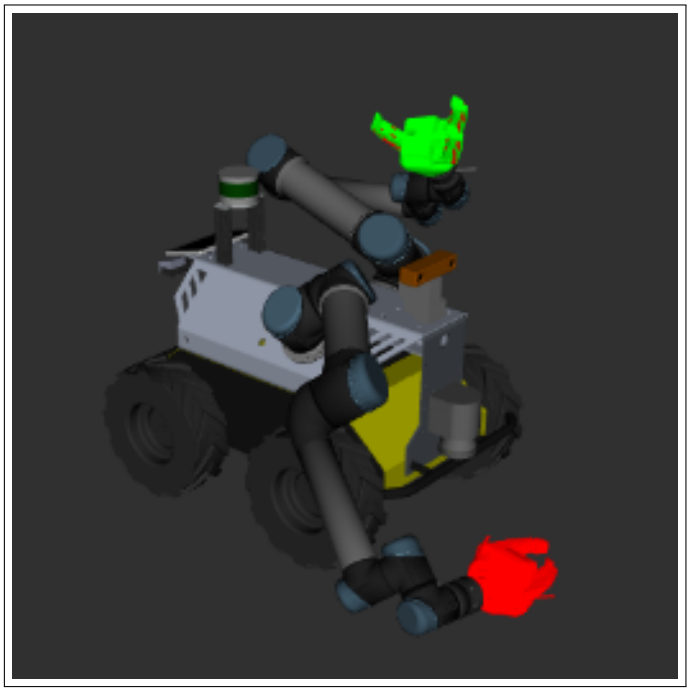

Fig. 9. Reach – A rough pose to lower the arm to a position close to where the object is anticipated to be located. Small adjustments would then be made to maneuver the arm into a position to pick up the object

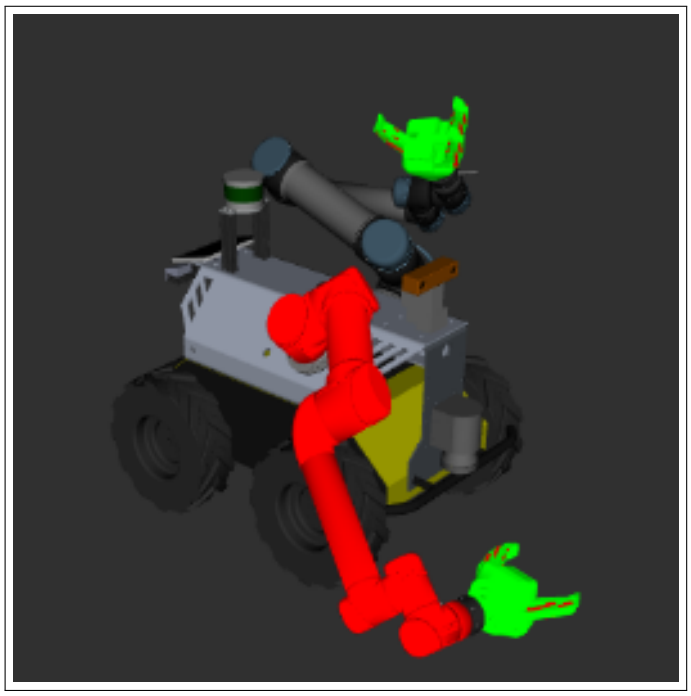

Fig. 10. Close – This pose only affects the end effector of the arm, the gripper, allowing it to grip the object without crushing it.

Fig. 11. Carry - This would be used to lift the object without spilling any contents.

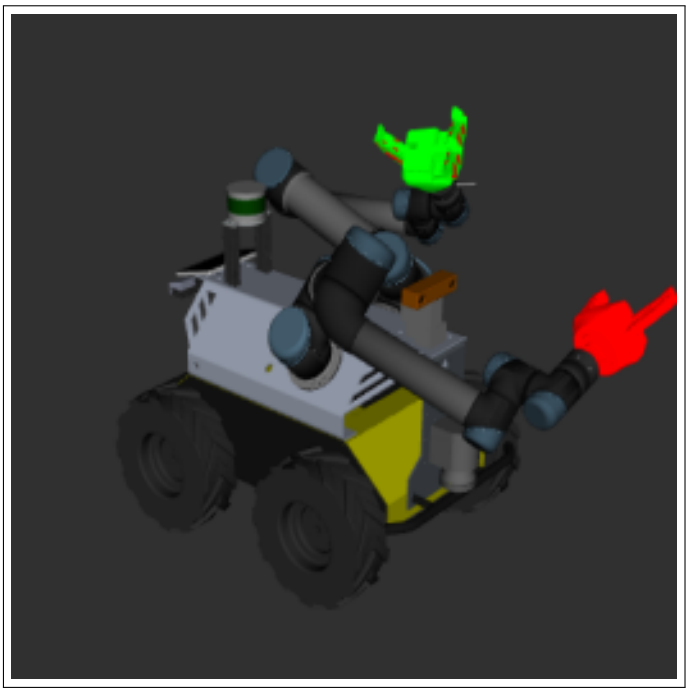

Fig. 12. Open – An end effector pose which would allow the robot to release the object.