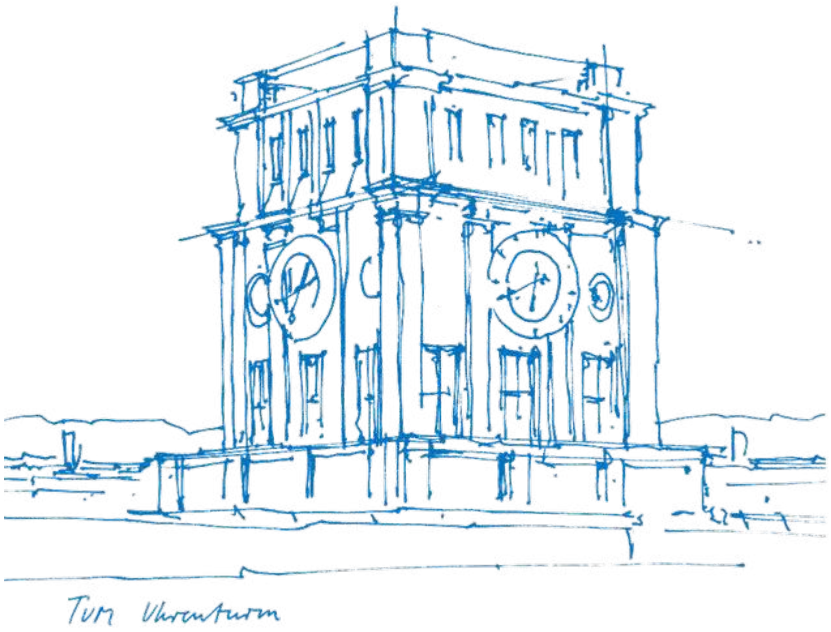
@Adjust name of chair or group (english) in tum-user.sty
@Adjust department name (english) in tum-user.sty
Technical University of Munich

Open the door and take things out

Youth2elders:
Chengjie Yuan
Cong Wang
Helin Cao
Yansong Wu
Fakultät für Elektrotechnik und Informationstechnik

Eingereicht am
München, den 07.02.2020



I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

München, 29.04.2016 Honest Student

Abstract

Considering the further application of the Tiago robot at home, the robot needs to have the ability to implement more specific and practical tasks. In this report, the Tiago robot is used to execute a certain task, in which there are three individual subtasks, namely, navigation, open the door of a cabinet, take one thing out from the cabinet. To achieve the final goal. a laser map is created, the robot can localize based on AMCL and navigate autonomously as well as avoiding obstacles by means of global and local path planning. In the control part, a new library TRAC-IK instead of default IK solver KDL is used to get a better computation of the joint positions so that a proper trajectory could be generated to open the door. Meanwhile, the planning in joint space and cartesian space based on MoveIt is used for the pre-position and grasping part.

Contents

1 Introduction

Nowadays, with the development of the Artificial Intelligence and Robotics, there are more and more scientific competitions based on this research field. With the help of these competitions, a common testbed is provided, in this case, different solutions and it could enable exchanges of research results. One of the classic competitions is RoboCup. RoboCup@Home is a competition where domestic service robots are performing several tasks in a home environment, interacting with people and with the environment in a natural way[1].

In this report, a task called "open the door and take things out" is generated. The whole task process can be seen as the Fig1.1.

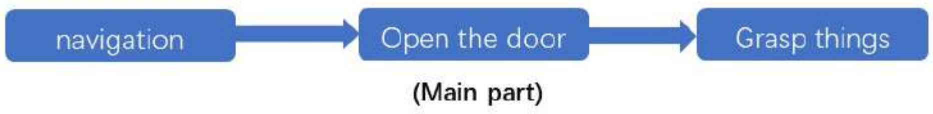


Figure 1.1 Task process

At first, the robot in a certain environment should navigate to a desired position (e.g. in front of the door), then the robot need to recognize the handle on fixed on the door, afterwards it grasps the handle and open the door. When the door is widely opened. The next step for the robot is to grasp a bottle(also can be other things predefined by the commander) and take it out. In this case, the total task can be divided into three subtasks, namely, navigation, data processing and control part. The Fig1.2 shows the Task distribution.

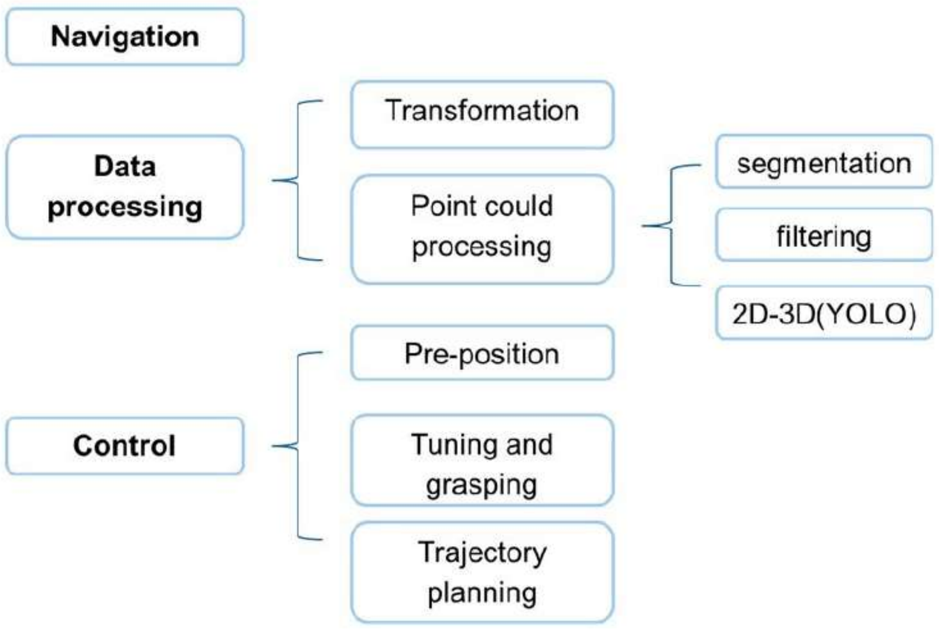


Figure 1.2 Task distribution

In this task, the main part is to open the door. To achieve a desired goal, we separate the part into 4 different steps, which can be seen in the Fig1.3.

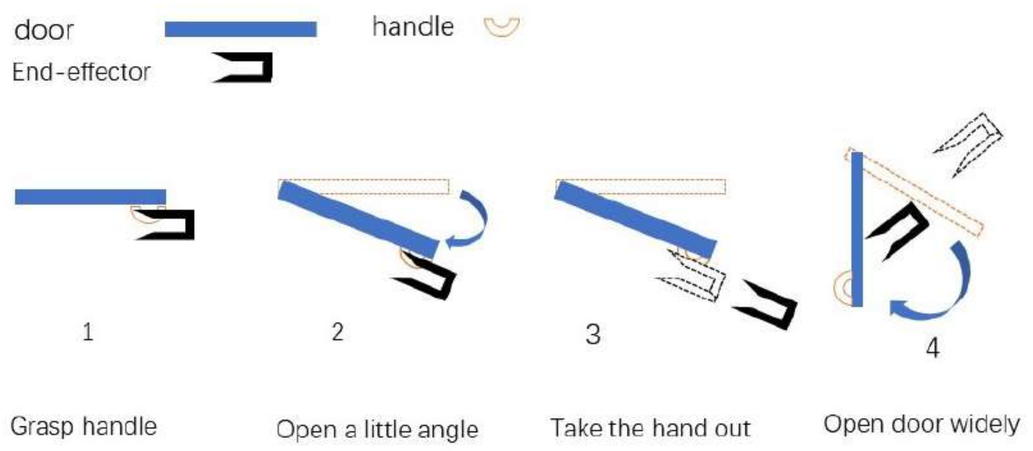


Figure 1.3 Four steps to open the door

2 Navigation

The reference to this part is the following links:
Create a map with gmapping
http://wiki.ros.org/Robots/TIAGo/Tutorials/Navigation/Mapping
Localization and path planning
http://wiki.ros.org/Robots/TIAGo/Tutorials/Navigation/Localization

In this part, we need to first create a map of the environment using "gmapping". Then, run laser-based localization and autonomous navigation avoiding obstacles by means of global and local path planning. In our task, the map is called "Y20_final" When using the map we should to call the service:

$ rosservice call /pal_map_manager/change_map "'Y20_final'"
In order to localize the robot run the following instruction:
$ rosservice call /global_localization "" Then we should rotate the robot to filter the costmap, afterward, we can use the following command to clear the costmap:
$ rosservice call /move_base/clear_costmaps ""

In our task, we integrate them in "test_nav" in the package "robot_nav_motion"

3 Data processing

3.1 3D point cloud

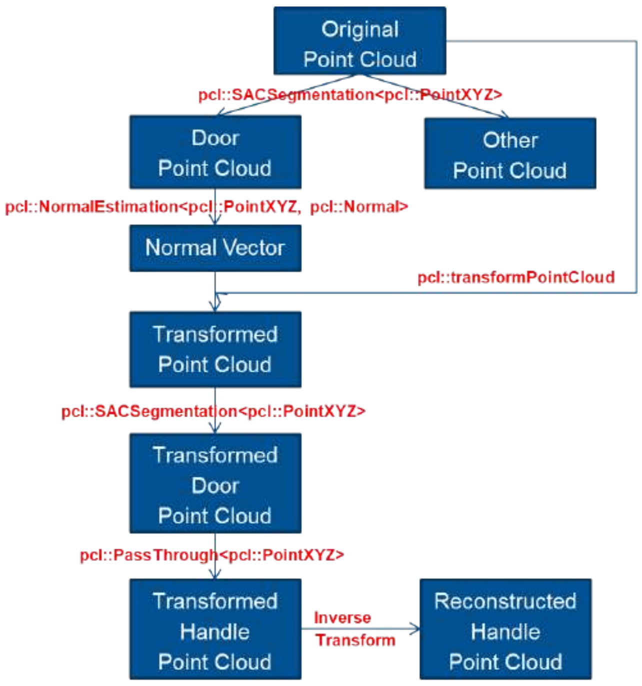
For the point cloud processing, we use the Point Cloud Library (PCL). The PCL is a large scale, open project for 2D/3D image and point cloud processing. The PCL framework contains numerous state-of-the art algorithms including segmentation, filtering, normal estimation, transformation and so on.

3.1.1 Problem Analysis

The target of point cloud processing is to obtain the 3-D position of the handle. Thus, we try to extract the handle point cloud and calculate the position of centroid. The door is the largest plane in the point cloud, so it can be segmented by PCL. And the handle is the only part locates in front of the door. We can get the handle point cloud based on this. But the problem is that the “face” of Tiago is not vertical to the door panel. So we can not use filter directly, the transform plays an important role in point cloud processing.

3.1.2 Processing Detials

The flow chart of point cloud processing can be shown as follows:

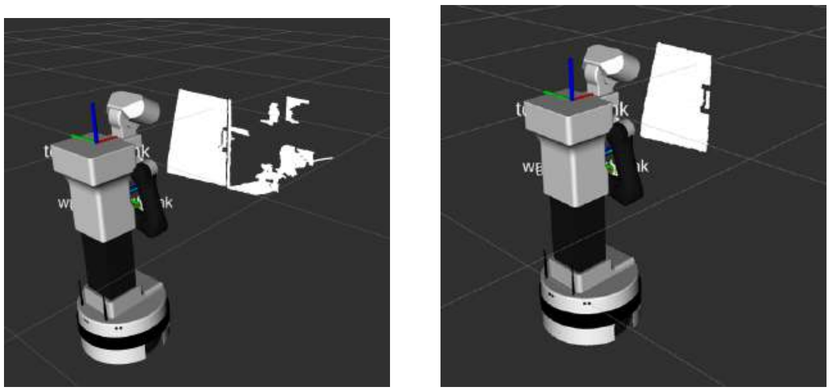


Original Point Cloud
pcl::SACSegmentation<pcl::PointXYZ>
Door Point Cloud
Other Point Cloud
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal>
Normal Vector
pcl::transformPointCloud
Transformed Point Cloud
pcl::SACSegmentation<pcl::PointXYZ>
Transformed Door Point Cloud
pcl::PassThrough<pcl::PointXYZ>
Transformed Handle Point Cloud
Inverse Transform
Reconstructed Handle Point Cloud

Figure 3.1 The flow chart of point cloud processing

Extract the Largest Plane

The first step is to extract the largest plane of the point cloud using *pcl::SACSegmentation<pcl::PointXYZ>*. In most views, the door is the largest plane. But in practice, we found that due to the deviation of the actual position navigated from the preset position, the area of the door in the point cloud is relatively small, so it is possible to regard another plane as the largest plane. For example, the plane on the right side can be regarded as the largest plane, so we utilize the pre-grasp pose of the robot arm. When the robot arm reaches the pre-grasp position, the robot arm will enter the view of Tiago, which just occludes the right part. At this time, the door will be regarded as the largest plane of the point cloud without any ambiguity.



(a) The original point cloud (b) The largest plane

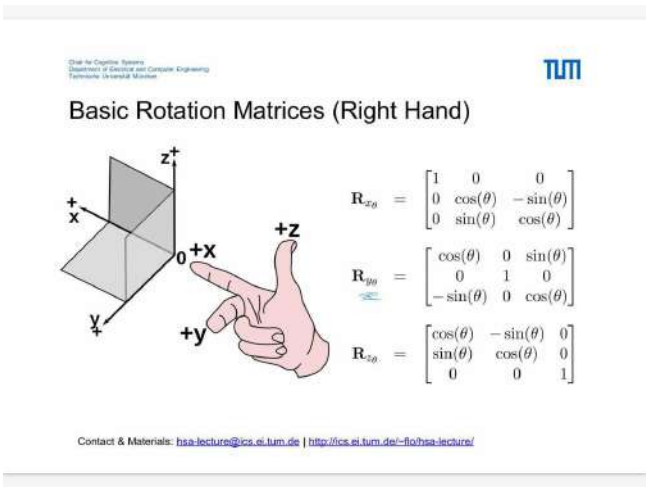
Figure 3.2 Extract the largest plane and get the normal vector

Extract the normal vector

We can easily get the normal vector of each point on the extracted plane by *pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal>*, then we can calculate the average of the normal vector of each point, that is a relatively accurate normal vector

Transform

This step is to calculate the normal vector based on the extracted plane in the first step using *pcl::transformPointCloud*. Because the point cloud is obtained by an RGB-D camera, the 3-D coordinate of the point cloud is based on *xtion_depth_optical_frame*. In order to extract the handle point cloud using the filter, we transform the point cloud which makes a normal vector parallel to the z-axis of the optical frame. That will make it easier to extract the handle point cloud using the filter in the next step. The key problem is to calculate the transform matrix according to the normal vector. Because the normal vector has no position, the translation can be set as a zero vector. The rotation matrix can be calculated as the slide shows:



Chair for Cognitive Systems
Department of Electrical and Computer Engineering
Technische Universität München

Basic Rotation Matrices (Right Hand)

$$\mathbf{R}_{x_\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{y_\theta} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\mathbf{R}_{z_\theta} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Contact & Materials: hsa-lecture@ics.ei.tum.de | http://ics.ei.tum.de/~flo/hsa-lecture/

Figure 3.3 The method of computing rotation matrix

We split the rotation into two-part. The first part is the rotation around the z-axis and the second part is the rotation around the x-axis. It is hard to get expressions that work in all situations, so it is better to discuss case by case according to different octants. The result of transform can be shown as follows:

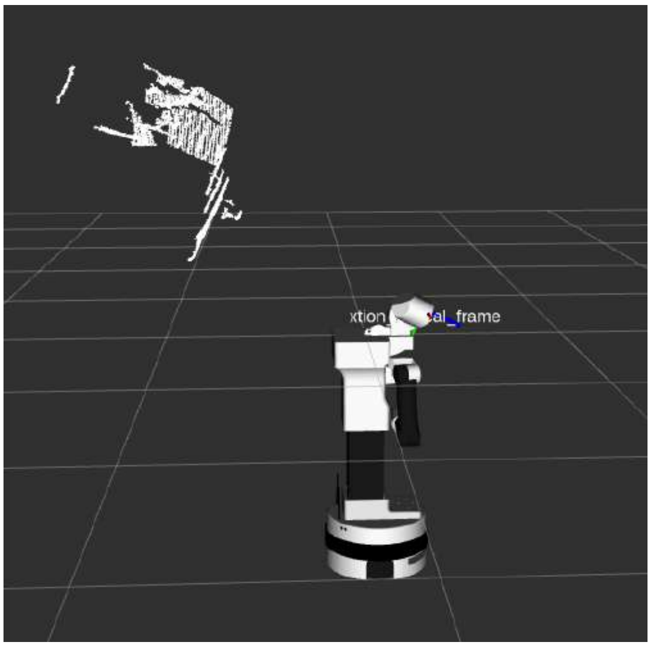
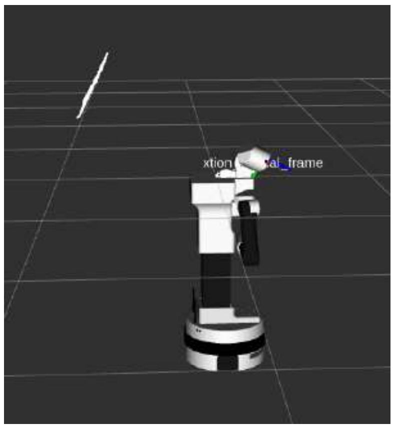


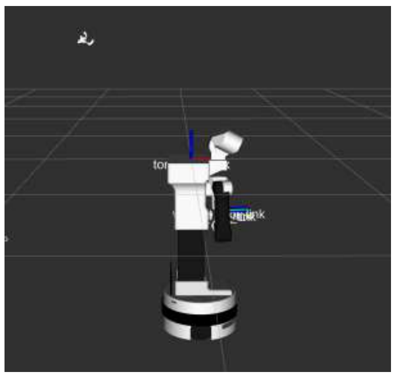
Figure 3.4 The result of transform

Filter

The filter in PCL is*pcl::PassThrough<pcl::PointXYZ>*. The boundary of filter can be set as the z-coordinate of the door, so the point cloud behind the door can be filtered.The result of filter can be shown as follows:



(a) The transformed largest plane



(b) The transformed handle

Figure 3.5 The result of filter

Reconstruction

The last step is reconstructing the handle point cloud by inverse transform.The result can be shown as follows:

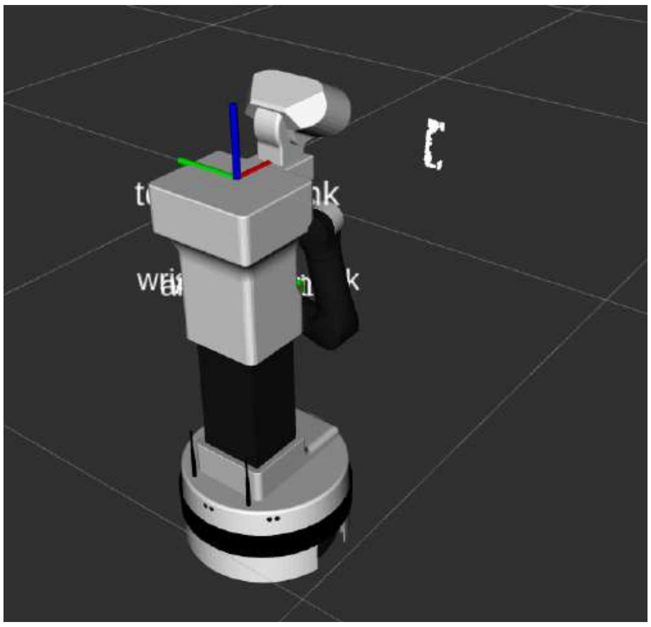


Figure 3.6 The result of reconstruction

3.2 Transformation

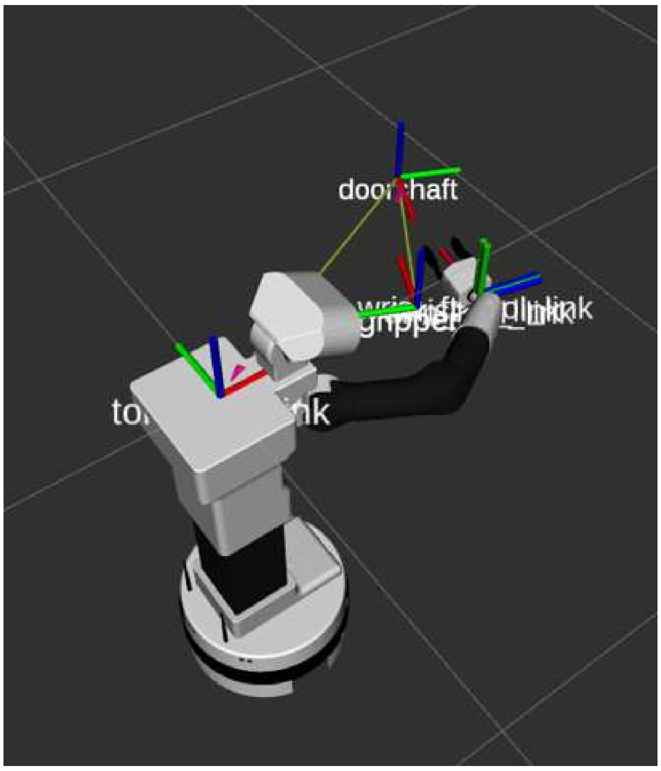


Figure 3.7 Basic frames illustration

We use many frames to control the Tiago in our project, as shown in 3.7. The frame “xtion_optical_-frame” is the frame of the robot camera. We use this frame to get the position of door handle as long as the position of the object. The frame “gripper_link” is the frame of the gripper. This frame almost coincides with the handle so that the robot can grasp the handle. The frame “doorshaft” and the frame “gripper1” are frames we made to make a better control. When the robot opens the door, we use the “torso_lift_link” as base link to control the arm. On the condition of grasping object, the frame “base_link” is used as the base link to control.

As shown below, the frame “doorshaft” is almost in the same plane with the handle and “gripper1”. The “doorshaft” is the frame of the shaft of the door. The direction of the x-axis of this frame is always against the x-axis of the gripper. When the robot opens the door, this “doorshaft” frame turns together with the door and the gripper of the robot. The frame “gripper1” is the planned frame of the gripper. It is the interpolated desired gripper frame. It shows the trajectory of the gripper as the robot opens the door. The open angle in our project is between 20 to 30 degrees, depends on the accuracy of the navigation part.

In door opening and the object grasping part, we have the 3D coordinate according to the camera frame “xtion_optical_frame”. We use the “tf_lookup::lookupTransform” of the Tiago package to get the transform of the frames in Tiago. And we use the “tf::TransformBroadcaster” to get the frame and transform of the frames designed by ourselves.

3.2.1 Object Grasping

In object grasping part, we use Yolo to detect the object, as shown in 3.11. After we get the detected coordinate of the object from Yolo, we can get the 2D coordinate from the Yolo topic “/darknet_ros/bounding_boxes”. In this topic, we can get the minimum and maximum of the x-coordinate and y-coordinate of the object in pixel coordinate. These coordinates are transformed and published into a new topic, with a massage type “Rect”. Together with the 2D coordinate in pixel, and the point cloud from topic “/x-

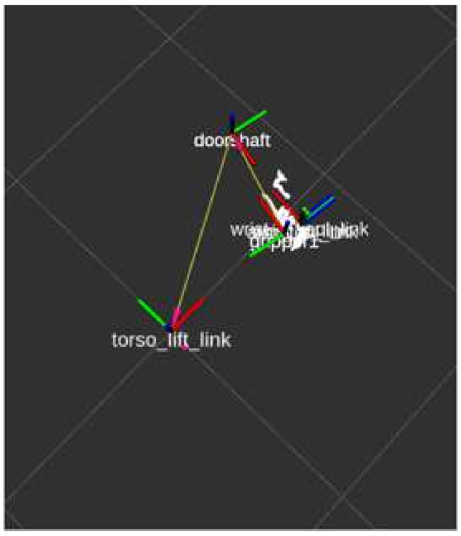


Figure 3.8 Frames illustration when opening door, step 1

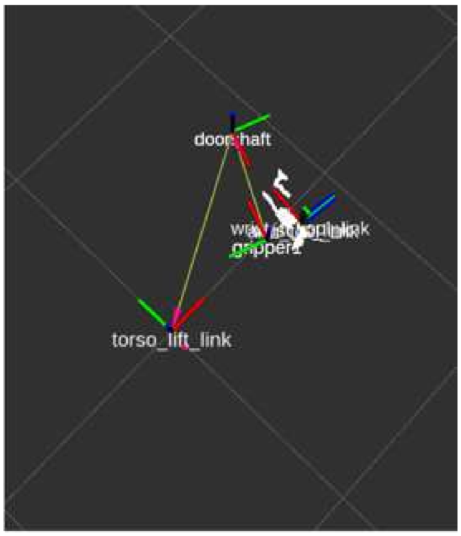


Figure 3.9 Frames illustration when opening door, step 2

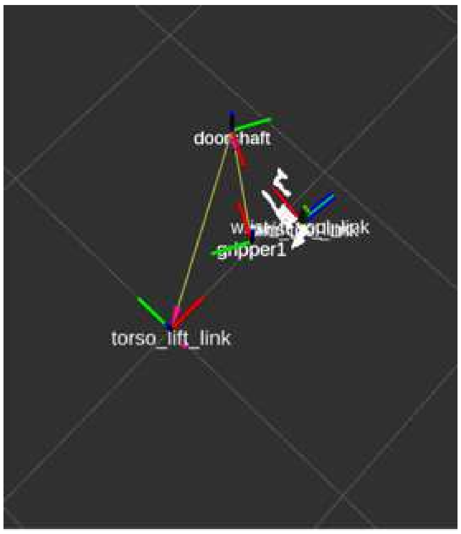


Figure 3.10 Frames illustration when opening door, step 3

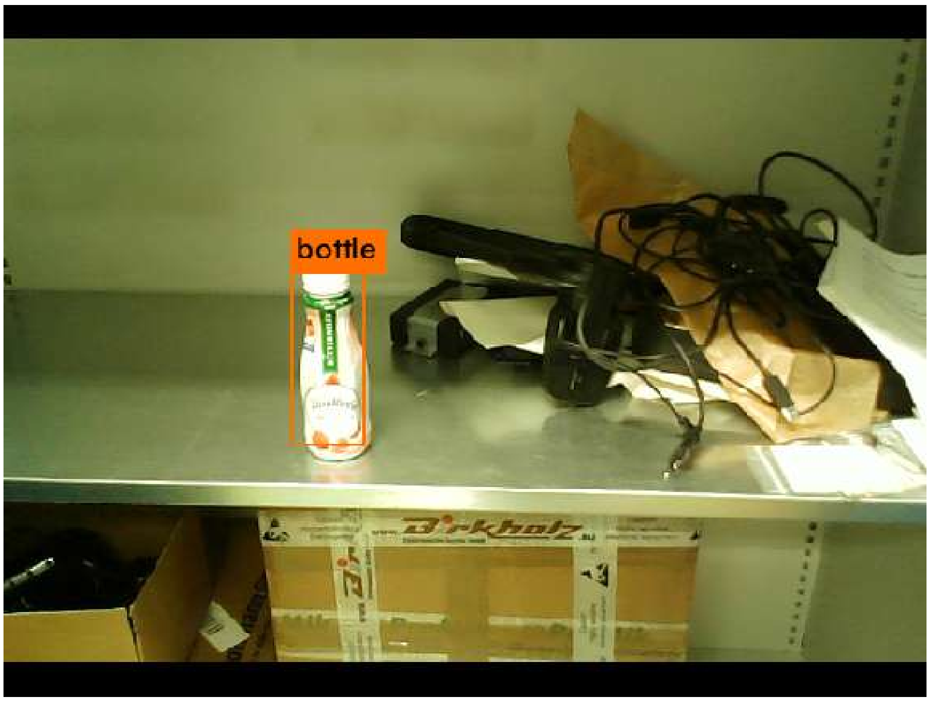


Figure 3.11 Object detection with Yolo

tion/depth_registered/points", we can get the transformed 3D coordinate through the method mentioned in former chapter.

After we get the 3D coordinate of the desired object according to the "base_link" frame, we use the Cartesian Space control method to let the Tiago grasp it. A pre-grasp post is designed, in order to avoid collision. When grasping, the y-axis and the z-axis are firstly set to be consistent with the object's coordinate. Then the gripper moves in the x-axis direction to get into the grasp position and make the end-effector execute the grasp motion. After grasping, robot turns back into home position

4 Control part

4.1 trajectory planning

The key point of opening a door of cabinet is controlling the end effector to move alone a predefined trajectory. To achieve this, generating an exact trajectory is of the essence. Then, a list of waypoints will be set on the trajectory. Through controlling the end effector to reach these waypoints in turn, we can approximately assume that, the end effector moves along the predefined trajectory. The match degree between the actual tracks and the predetermined trajectory depends the complexity of the trajectory, the number of the inserted waypoints and the property of the robot arm. The waypoints are calculated in cartesian space. However, the change of end effector's position is implemented by adjust the rotation angle of each joint. Therefore, a transformation from the cartesian space to joint states through inverse kinematic is of vital importance.

4.1.1 Inverse kinematic

Inverse Kinematics, when given the configuration of a robot, provides a set of joint values that reach a desired Cartesian pose for the robot's end effectors. Nowadays, the most widely-used generic inverse kinematic (IK) solver worldwide is the inverse kinematic algorithm implemented in the open-source Orocos Kinematics and Dynamics Library (KDL). This solver is based on the joint-limit constrained pseudoinverse Jacobian to find joint solutions that come "close enough" to the desired Cartesian solution.

$$q_{next} = q_{prev} + J^{-1} p_{err} \tag{4.1}$$

However, in practice, KDL always encounter false-negative failures. There are two main reasons for the failure:

1) frequent convergence failures for robots with joint limits
2) no actions taken when the search becomes "stuck" in local minima
To improve these problems, there are some approaches.

The KDL-RR Algorithm

The KDL-RR Algorithm consists of two parts. On the one hand, it runs the KDL with a number of iterations. On the other hand, local minima are easily detectable when

$$q_{next} \approx q_{prev} \tag{4.2}$$

. If a local minimum is detected, a random seed is generated for q to "unstick" the iterative algorithm. KDL-RR solve the problem of local minima effectively. However, it is still confused by the convergence failures for robots with joint limits.

The SQP IK Algorithm

Inverse kinematic as a nonlinear optimization problem can be solved locally using sequential quadratic programming (SQP). The SQP problem can be evaluated using the NLopt library. Using the NLopt C++

API, it was possible to create a “drop in” replacement for KDL’s IK solver that uses the SLSQP [2] algorithm. Because SQP is focusing on minimizing the overall amount of joint movement and is only considering Cartesian error as a constraint, it is biasing the search around the joint space of the seed value, rather than making constant progress towards minimizing the overall Cartesian error.

The SQP-SS and SQP-L2 IK Algorithms

In SQP-SS and SQP-L2 IK algorithms the nonlinear optimization formulation is changed to minimize Cartesian pose error directly (Sum of Squares error for the 6-element error vector and L2-Norm of that Cartesian error vector). In this case, only the joint limits continue to be constraints to the SQP formulation.

The TRAC-IK Algorithm[3]

The TRAC-IK works as a combination of KDL-RR and SQP-SS algorithms. It creates two thread and run the two algorithms synchronously. If one algorithm finds the solution, the other algorithm will be shut downed. Compared with KDL, it runs significantly faster and more stable.

In this project, we use the TRAC-IK to take the place of the default IK solver of KDL. To install the TRAC-IK library,

```
$ sudo apt-get install ros-kinetic-trac-ik
```

For each inverse kinematic, a “seed” joint state is necessary to generate the Jacobian matrix. The result of the last inverse kinematic is used as “seed” for the next inverse kinematic.

4.1.2 Moving through waypoints

After getting the corresponding joint states, just like the “Joint Trajectory Controller” call the corresponding action to control the end effector move along the determined trajectory.

```
$ /arm_controller/follow_joint_trajectory (control_msgs/FollowJointTrajectoryAction)
```

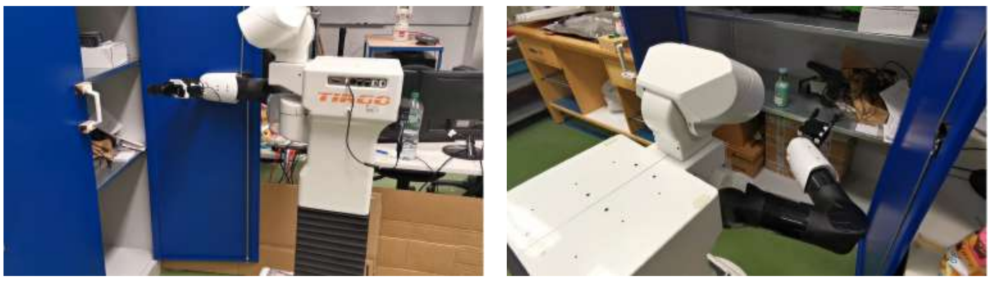
4.2 pre-position, grasping and tuning

The basic methods used in our task are the following based on MoveIt.
Planning in joint space
http://wiki.ros.org/Robots/TIAGo/Tutorials/MoveIt/Planning_joint_space
Planning in cartesian space
http://wiki.ros.org/Robots/TIAGo/Tutorials/MoveIt/Planning_cartesian_space

To make it easier for further computation of the robot arm to grasp things, at first we should move the robot arm to a pre-position. Depending on two different situations, pre-position should be different as well. To grasp the handle or bottle, we first define a pre-position like the Fig 4.1 shows.
To get the pre-position, we used the planning in joint space based on MoveIt. The position is dependent on the target object. We can use this below command line to launch the arm controller or we can just use the web commander to first move the robot arm to a desired position, then we can record the joint position for further control.

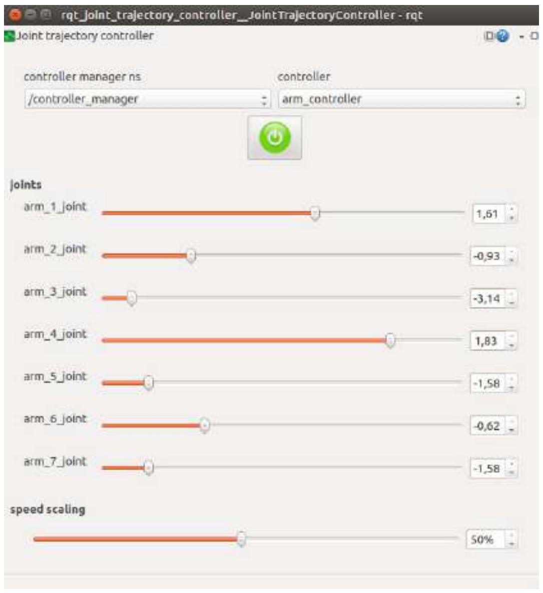
$ rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller

The second step is to tune the robot arm to closer position to the goal object. And the total process can be seen below. Considering the safety problem, two points should be put forward. One is the speed,this process should be slow in order to avoid breaking the gripper or handle. The other point is the height of



(a) pre-position1 (b) pre-position2

Figure 4.1 Pre-position



rqt_joint_trajectory_controller__JointTrajectoryController - rqt

Joint trajectory controller

controller manager ns: /controller_manager

controller: arm_controller

joints

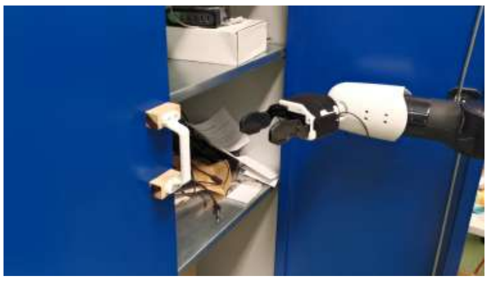
arm_1_joint	1,61
arm_2_joint	-0,93
arm_3_joint	-3,14
arm_4_joint	1,83
arm_5_joint	-1,58
arm_6_joint	-0,62
arm_7_joint	-1,58

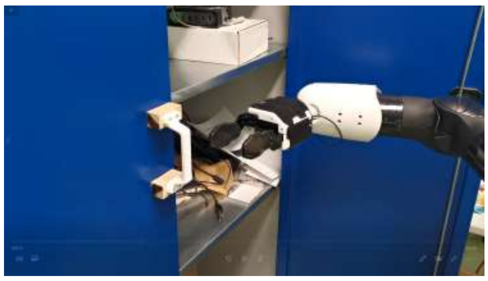
speed scaling: 50%

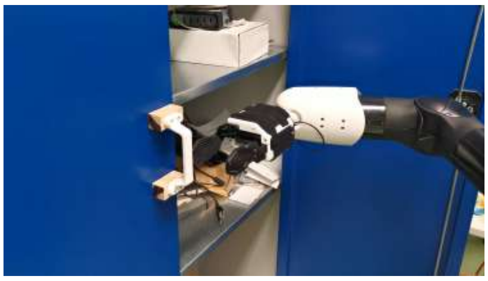
Figure 4.2 rqt_joint_trajectory_controller

the end-effector. When using the planning in cartesian space, all the joints including torso link will be considered. Hence it is not so safe to use planning in cartesian space to do the tuning part. Instead, we use the Trac-IK based method, because this method considered only on the arm joints and we can interpolate waypoints in the trajectory to ensure the height of the end-effector will not change. The target position is based on the 3D point of the handle with respect to the torso_lift_link.

Considering the bottle, the free area for the robot to move its arm is bigger and safer, so that we can use planning in cartesian space to tune the end-effector. The tuning based on the 3D position with respect to the base_link of the robot.


(a)


(b)


(c)

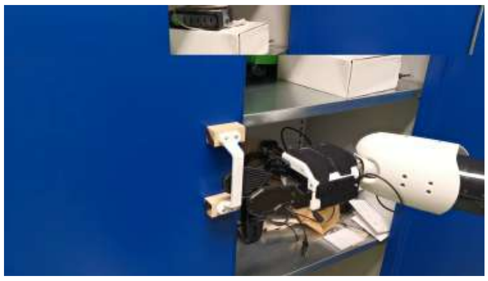
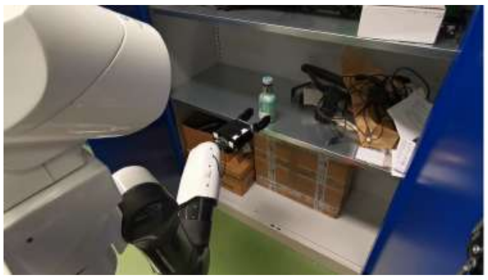
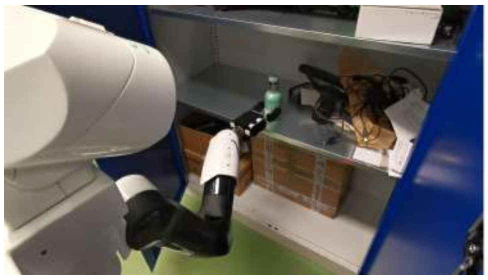
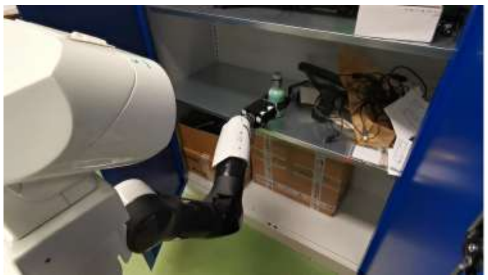

(d)

Figure 4.3 tuning process for handle


(a)


(b)


(c)

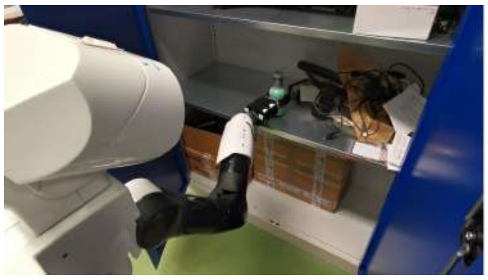

(d)

Figure 4.4 tuning process for bottle

5 Integration

The following flow chart shows how the robot work autonomously.

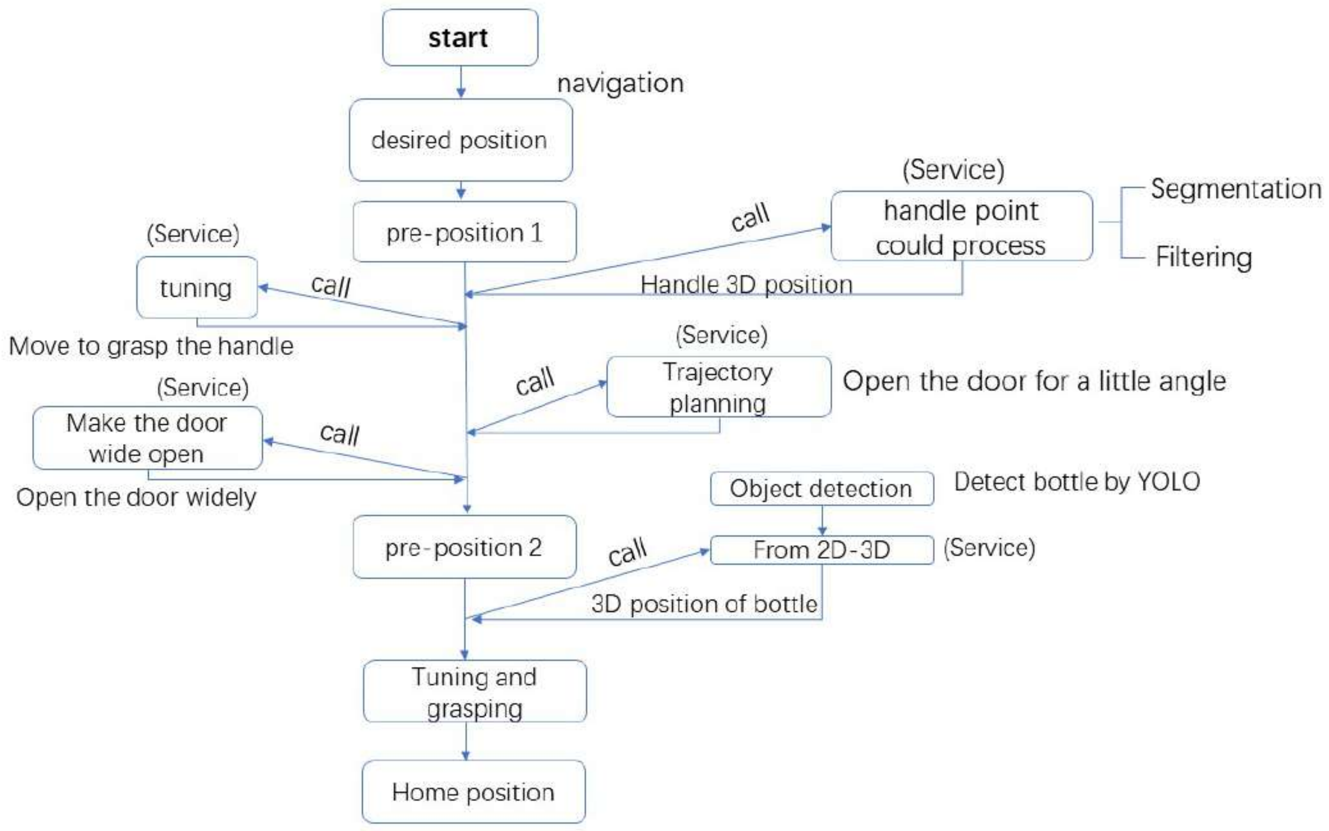


Figure 5.1 Total connection between each part

Bibliography

[1] Dirk Holz, Javier Ruiz-del Solar, Komei Sugiura, and Sven Wachsmuth. On robocup@ home–past, present and future of a scientific competition for service robots. In *Robot Soccer World Cup*, pages 686–697. Springer, 2014.

[2] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[3] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935. IEEE, 2015.