Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Mobile Manipulation in Cluttered Environment

Joonhong Min

Supervisor: Dr. Gaël Ecorchard
Field of study: Cybernetics and Robotics
May 2021

Acknowledgements

Great thanks to Dr. Gaël Ecorchard for his sincere advice and patience.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

I declare that I have prepared the submitted work independently and that I have listed all the used literature.

Prague, 10. May 2021

Abstract

The main purpose of this thesis is to examine the representations of the robot's components, how they are structured, and for what purpose. The creation of representation will be tried with the examined information, which can be applied to both simulation and real robots. The Difference between simulation and the real environment will be accessed and compared. Implementation of state-machine to pick an object will be tried upon created representation of the robots.

Keywords: robot, ROS, statemachine, representation, simulation, manipulation

Supervisor: Dr. Gaël Ecorchard

Abstrakt

Hlavním účelem této práce je zkoumat reprezentace komponent robota, jak jsou strukturovány a za jakým účelem. Vytvoření reprezentace bude vyzkoušeno se zkoumanými informacemi, které lze aplikovat jak na simulaci, tak na skutečné roboty. Rozdíl mezi simulací a reálným prostředím bude zpřístupněn a porovnán. Implementace stavového stroje pro výběr objektu bude vyzkoušena na vytvořené reprezentaci robotů.

Klíčová slova: robot, ROS, statemachine, reprezentace, simulace, anipulace

Překlad názvu: Mobilní Manipulacev Prostředí s Překážkami

Contents

Figures

Tables

Chapter 1

Introduction

For humans, one does not need to know or parameterize how heavy one's arm is, where the center of mass is, at what limit it can twist its own arm, and what degrees of freedom the elbow has to move the one's arm around. Nevertheless, for computers or robots, they require all the minor details should be presented and enlisted.

This thesis will take a closer look into how the robot's components are represented in a way that the robot can understand and try to implement a state machine onto that information.

This thesis is consists of five chapters, excluding this chapter. A robot models chapter will introduce two specific robot models used in this thesis and explain how each component is represented in detail. The representation elements, in general, will be examined, and how different robots or components can be integrated will be explained within the robot models chapter.

The Setups chapter will explain the brief scenario given for the thesis, how its environment is set for physical world and simulation, and the modifications made from the existing representations to meet the given condition.

The visualization and simulation chapter will mainly talk about how created robot representations are manipulated within the simulation with what methods.

In the experiments chapter, the implementation of the state machine to pick an object will be tried and explained. Furthermore, after that, the overall thesis will be recapitulated and discuss the possible future works.

Chapter 2

Robot Models

This chapter will introduce and briefly explain the robots simulated and tested for the thesis. First, the way how the robots are represented in general will be explained. Then the way how they are represented in the language that Robot Operating System(**ROS**) can understand.

Initially, there was only one robot, built upon Clearpath Robotics' Husky[ROSa] mobile base, attached Universal Robots' UR5[ROSd] robotic arm with Robotiq's 3-Finger Adaptive Robot Gripper[ROSb], denoted this robot as "HUR" further on. In the meantime, another robot was introduced for the thesis called TIAGo++ or TIAGo Dual[ROSc], made by Pal Robotics. In the thesis, the word "Tiago" will refer to TIAGo Dual.

In the section 2.2 Robots, representations of HUR and Tiago will be explained in depth, and how each elements and parameters were used by the packages.

2.1 Robot Representations

In order to visualize a real robot on a screen or spawn the 3D model on a simulation, it is necessary to define the robot representations in a language that ROS can understand. For all the robots used in this thesis are represented in Unified Robot Description Format(**URDF**), written in Extensible Markup

Language(**XML**). It is possible to represent a robot in a single URDF, but it will be hard to organize and understand URDF as the robot gets more components and gets complicated. It is possible to visualize the structure of a robot parts if a robot is defined in a single URDF file as shown in fig.2.1 using `urdf_to_graphiz`, but since all the robots used in the thesis were divides into parts and combined using XML Macro(**Xacro**)[ROSe], which are not possible to visualize using `urdf_to_graphiz`.

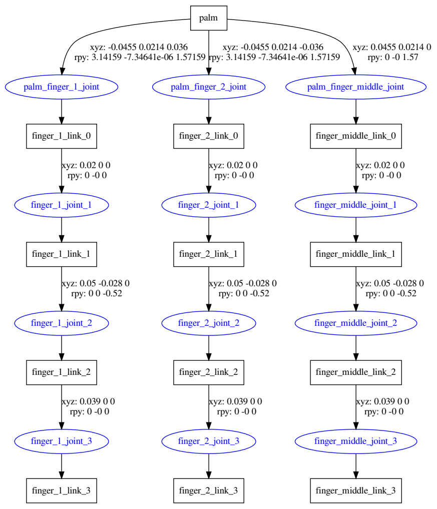


Figure 2.1Visualized `robotiq-3f-gripper_articulated.urdf` using `urdf_to_graphiz`[Rob15]

Since it gets difficult to design complicated shapes with XML alone, it is possible to use 3D modeling tools to design a model, and by using Xacro, it allows calling similar repetitive parts easily. All the robot models used in this thesis were based on provided mesh files in COLLAborative Design Activity(**Collada**, with the extension of .dae) and stereolithography(**STL**) format from the companies. Following is an example of a link representation and import of mesh files from a husky's `decorations.urdf.xacro`[CR15].

```
<!-- Spawn the top plate -->
...
<link name="top_plate_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://.../large_top_plate.dae" />
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://.../
                                    large_top_plate_collision.stl" />
    </geometry>
  </collision>
</link>
```

As shown in Fig. 2.2, When visualized, Collada and STL models of Husky `base_link` seems like they are identical, yet all the companies have used Collada to define the `<visual>` and STL for the `<collision>`. This is because Collada can hold various information about the part, not only the shape, while STL only contains triangle meshes. From Fig. 2.2, it is possible to see that Collada mesh has a similar color to an actual Husky's `base_link`, since it also has information about the color of the part.

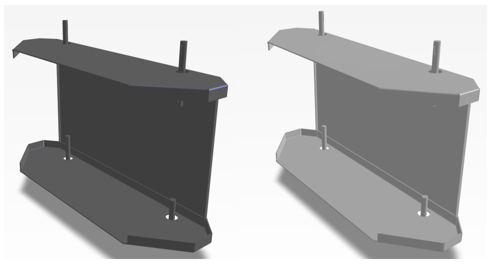


Figure 2.2Visualized `base_link` with left) Collada; right) STL

Once the links of robot parts are defined, it is necessary to define the joints, an element containing the coordinates and degrees of freedom of the parts attached. Following is defined joint in between husky's `top_plate_joint` and `base_link`[CR15].

```
<!-- Attach top plate -->
<joint name="top_plate_joint" type="fixed">
  <parent link="base_link" />
  <child link="top_plate_link"/>
  <origin xyz="0.0812 0 0.225" rpy="0 0 0"/>
</joint>
```

In order to properly simulate the robot, information about the robot part's moment of inertia must be presented. In URDF, the information is defined with a mass of the part, the center of mass, and six elements of an inertial matrix. For example, the inertial expression of husky wheels and usage of Xacro is provided below.

```
<xacro:macro name="husky_wheel"
             params="wheel_prefix *joint_pose">
    <link name="${wheel_prefix}_wheel_link">
        <inertial>
            <mass value="2.637" />
            <origin xyz="0 0 0" />
            <inertia  ixx="0.02467" ixy="0" ixz="0"
                      iyy="0.04411" iyz="0" izz="0.02467" />
        </inertial>
        ...
    </link>
</xacro:macro>
```

The way how ROS is manipulating the robot is sending a commands to a hardware and gets the changes in `joint_states` through the ROS controllers[CMEM+17]. To specify the parts controlled by specific controller, it is required to define the relationship between joint and an actuator by `<transmission>` element. `transmission_interface` and `hardware_interface` can be defined as an elements of `transmission` within the URDF along with other representation elements, or define separately and combined later with Xacro. Following is defined interface for the wheels of Husky[CR15]:

```
<transmission name="${wheel_prefix}_wheel_trans"
              type="SimpleTransmission">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="${wheel_prefix}_wheel_motor">
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
    <joint name="${wheel_prefix}_wheel">
        <hardwareInterface>
            hardware_interface/VelocityJointInterface
        </hardwareInterface>
    </joint>
</transmission>
```

One last element expressed in URDF is `<gazebo>`. The gazebo element is not essential like the others, but it helps to have a more realistic simulation. Various parameters can differ according to the referenced part, whether a link or joint. For Husky wheel link, it has defined friction coefficients(`mu1, m2`), contact stiffness(`kp`), damping for rigid body contacts(`kd`), and direction of friction coefficient(`fdir1`)[Opeb].

```
<gazebo reference="${wheel_prefix}_wheel_link">
    <mu1 value="1.0"/>
    <mu2 value="1.0"/>
    <kp value="10000000.0" />
    <kd value="1.0" />
    <fdir1 value="1 0 0"/>
</gazebo>
```

2.2 Robots

As explained early in the chapter, there are two robots used, HUR and Tiago. For the HUR model, there was an already built model from the preceding study by Enzo Geromin, which was based on a different ROS version. For Tiago, there is a package provided by Pal Robotics, which has a working robot description for melodic development. In subsections for each robot, their representations will be explained in detail.

2.2.1 HUR

Since HUR is built with three different robots made by different companies, it is necessary to integrate them into one robot description. There was a already built combined representation of HUR from the previous study, but some changes were needed to used it properly and these changes made to previous study and original definitions provided by the companies will be discussed it setup chapter. Just like the way how different representations were combined in previous section, it is possible to attach different robots onto each other by using Xacro. It was required to put a prefix on some parts because of that, some links from different robots could share the same names, for example both Husky and UR5 had `base_link`.

In together with the URDF representations, to manipulate the robots, the controller for the parts should be presented. In general, controllers are organized in YAML Ain't Markup Language(**YAML**) format, which can be spawned and managed easily with the `controller_manager` package.

Husky UGV

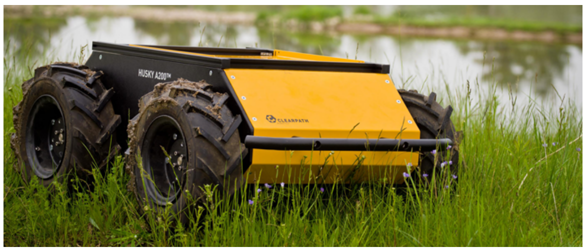


Figure 2.3: Clearpath Robotics' Husky[ROSa]

As shown in the figure2.3, the Husky Unmanned Ground Vehicle is a non-holonomic mobile base developed by Clearpath Inc. Its maximum speed is 1.0 m/s, weight is 50 kg with a maximum payload of 75 kg, and battery runtime is around 3 hours. With the real robot, Husky was powered with battery and communicated through the USB.

Clearpath provided representation of the Husky is made of three Xacro files, `husky.urdf.xacro`, representing overall contruction of the robot, `decorations.urdf.xacro`, describing exteriors like chassis, and wheel defining, `wheel.urdf.xacro`.

`husky.urdf.xacro` hold not only the information about the representation but also has various gazebo plugins like `gazebo_ros_control` and information plugins needed for possible peripherals like IMU, GPS, and Intel Realsense modules. Husky's wheels are joined with `base_link` pointing parallel to the `base_link`'s direction as continuous joint. Since all four wheels directions are bound to the same direction, Husky is using `diff_drive_controller`, controller for differential drive wheel systems, from `ros_controllers` package.

UR5

Universal Robot's UR5 is a robotic arm with 8 links and 7 joints in a single chain. Fig. 2.4 is not the exact UR5, but it helps to have some idea, since

UR5e and UR5 shares the same look. UR5 weighs 18.4 kg with 5kg of payload and has 6 degrees of freedom. In laboratory, UR5 was powered through AC power plugs, and communicated through network with ethernet cable. Due to the limited wiring, the maximum area that HUR can reach was limited.

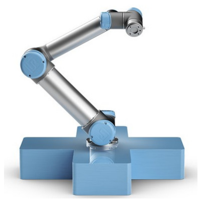
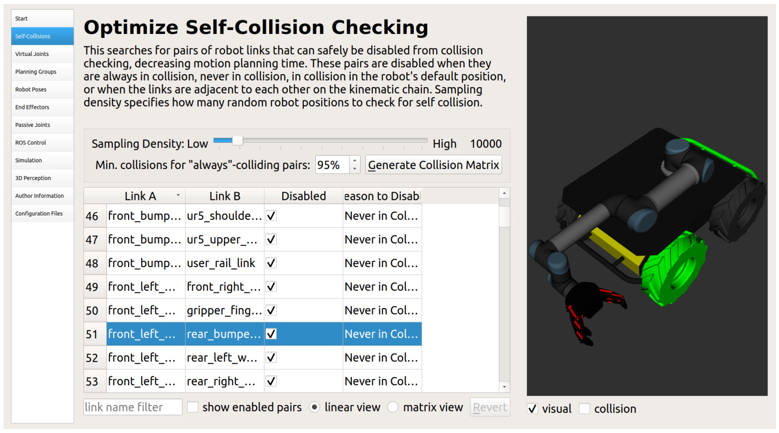


Figure 2.4: Universal Robot's UR5e[Robb]

While husky had three Xacros representing, footprint, exterior and wheels, UR5 has all base, arm and wrist are represented in `ur5.urdf.xacro` and included `transmission` and `gazebo` elements with `ur.transmission.xacro` and `ur.gazebo.xacro` which are compatitable with other Universal Robot's models.

Controller for UR5 is also defined in YAML, `JointTrajectoryController` as default. While it is relatively easy to manipulate a mobile bases like Husky, it is hard to manipulate individual joints in chain with human intuitions. The main task that is expected from the robot arm is sending a tool attached on arm's end effector to a desired coordinate. To achieve this the framework called **Moveit!**[CSCC14] is been used.

One of the first thing that Moveit requires is semantic description in Semantic Robot Description Format(SRDF). SRDF is written in XML, it contains information about the groups of joints and links, specified end effector and collision checking informations. Unlike URDF representation, SRDF can not be combined using Xacro. It is possible to make a description by hand, but the recommanded way of making SRDF is using `MoveIt Setup Assistant`.



Optimize Self-Collision Checking

This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling density specifies how many random robot positions to check for self collision.

Sampling Density: Low — High 10000
Min. collisions for "always"-colliding pairs: 95% Generate Collision Matrix

	Link A	Link B	Disabled	eason to Disab
46	front_bump...	ur5_shoulde...	✓	Never in Col...
47	front_bump...	ur5_upper_...	✓	Never in Col...
48	front_bump...	user_rail_link	✓	Never in Col...
49	front_left_...	front_right_...	✓	Never in Col...
50	front_left_...	gripper_fing...	✓	Never in Col...
51	front_left_...	rear_bumpe...	✓	Never in Col...
52	front_left_...	rear_left_w...	✓	Never in Col...
53	front_left_...	rear_right_...	✓	Never in Col...

link name filter ☐ show enabled pairs [● linear view] ○ matrix view Revert ✓ visual ☐ collision

Figure 2.5: GUI of MoveIt Setup Assistant

When SRDF for HUR was made, most of the names were changed from the UR5's SRDF because of the prefixes and to avoid the interuptions, but since UR5 is only part that is manipulated with Moveit, the principles were kept from UR5's sementic description. The detail changes will be given in Setup chapter. Following is example how the UR5 SRDF groups and end effector was defined.

```
<group name="manipulator">
    <chain base_link="base_link" tip_link="ee_link" />
</group>
<group name="endeffector">
    <link name="ee_link" />
</group>

<end_effector name="moveit_ee" parent_link="ee_link" group="
    endeffector" />
```

`joint_limits.yaml` can be generated with the Moveit Setup Assistance, which can specify and limit the parameters like maximum velocity and acceleration. Initially when the `joint_limits.yaml` is generated, it is just copy of defined limits in URDF, but since it is overriding the URDF when executed, user can change the limit parameters without changing the URDF. Yet, it is possible to use Moveit without `joint_limits.yaml`. The way how the `joint_limit` for UR5's `shoulder_pan_joint` is defined in YAML:

```
joint_limits:
  ur5_shoulder_pan_joint:
    has_velocity_limits: true
    max_velocity: 3.15
    has_acceleration_limits: true
    max_acceleration: 3.15
```

In order to send a tool attached to UR5's end effector, it is needed to know the specific joint states when the tool was sent to a designated position. For Moveit to solve relations in kinematic chain, it is necessary to define which method will be used with `kinematics.yaml`. There are various kinematic plugins, and for the thesis Kinematics and Dynamics Library(**KDL**)[Smi] was used, which is a default kinematic solver used in Moveit. In following lines from `kinematics.yaml` it is possible to see that a solver defined group name is matching wih the group defined in SRDF and the way how the plugin was added and defined:

```
manipulator:
  kinematics_solver: kdl_kinematics_plugin/
   KDLKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
```

At last, by running `move_group` node from `moveit_ros_move_group`, Moveit will be initiated and able to caculate and manipulate the UR5.

Robotiq 3-Finger Adaptive Robot Gripper

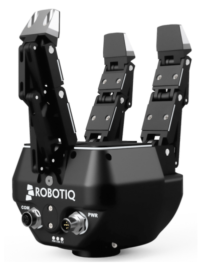


Figure 2.6: Robotiq's 3-Finger Adaptive Robot Gripper[Roba]

Robotiq 3-Finger Adaptive Robot Gripper, later denoted as Robotiq in the thesis has three individually controlable fingers, with 30 to 70 N of grip force.

Like UR5, Robotiq is also connect to ROS with network through ethernet cable.

Robotiq's representation was described in a single Xacro file, `robotiq_hand_macro.urdf.xacro`. Because it was defined as Xacro, it is not possible to visualize the structure using `urdf_to_graphiz` as in Fig. 2.1. All the fingers shares the same look and principle, except for middle figer is only allowed to rotation in one direction, while other fingers can move in two ways.

Unlike other two robots, Robotiq was controlled with company provided Gazebo plugin, `RobotiqHandPlugin` which has been modified for the thesis. About the specific plugin will be discussed later in the thesis.

2.2.2 Tiago Dual

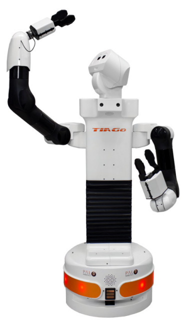


Figure 2.7: PAL Robotics' Tiago++[ROSc]

While HUR needed a entegration of different robot's representation, the whole Tiago itself is built by Pal Robotics, and it was possible to use provided descriptions straight away. Tiago's representation has different type of end effector defined as a preset, they can be interchangable easily by changing parameter arguments. For both left and right arms, end effector `pal-gripper` shown in Fig. 2.7 are used for the thesis.

In terms of manipulation, tiago has six controllers: `torso_controller` controlling one degree vertical movement, `gripper_right_controller`, `gripper_left_controller` controlling position of the finger joints, `arm_left_controller`, `arm_right_controller` controlling joints of each arm having 7 degrees of freedom, `mobile_base_controller` which is also using `diff_drive_controller` like Husky, and `head_controller` for moving the Tiago's head around z-axis of `head_1_link` and z-axis of `head_2_link`.

Tiago URDF decription is based on `tiago_dual.urdf.xacro` which calls other parts, move base, torso, arms, end effector, head and gazebo plugins using `xacro:include`. Than the links relations are also defined in `tiago_dual.urdf.xacro` like how the links between Husky and UR5, UR5 and Robotiq are defined. Elements `gazebo` and `transmission` are defined in individual part's URDF.

Chapter 3

Setups

In this chapter, the scenario for the simulation and testing will be explained briefly. For the Environment section, the physical installation of the testing site will be introduced and explained. Also, the software used and their specific versions and settings will be discussed. Also any changes or modifications have made from the previous study and provided packages from the companies will be given in this chapter.

3.1 Environments

The basic task used for both previous study and this thesis is grabbing a plastic water bottle standing on the rectangular table when all the position of objects and robot are known.

The whole simulation and the manipulation of the robots were done with the Robot Operating System(ROS) Melodic distribution, upon Ubuntu 18.04.5 LTS. Unless written in another form, when the word ROS is used, it means specifically about Melodic distribution. When this thesis was written, there are newer distribution noetic for ROS1 and Robot Operating System2 existed, but not all the companies were providing updated packages. To avoid the compatibility issues avoided those environments at the time when this thesis was written. The main languages for ROS are C++ and Python. Python was used as a main language for the thesis, and C++ for specific task.

ROS workspace is made of multiple packages. There are two methods to build the ROS workspace, `catkin_make`, and `catkin build`. For the thesis, `catkin build` was used whenever the workspace was required to built, rebuilt and adding a new package. Development of `catkin build` was followed by the `catkin_make_isolated`, which was aiming to build the workspace in parts. With the `catkin build` it is possible to build individual packages and it give flexibility in manipulating the workspace[Ope14].

Figure 3.1: VICON Vantage+[Ltd]

For the robot and objects localization in real experiments, Vicon Motion Systems' Vantage cameras were used. There are cameras installed on the ceiling of the laboratory, surrounding rectangular area which has a pillar at the center of this area. These cameras are point toward the area and tracking the object. To track the objects or robots, attached three pearl hard markers on them. In fact, what the VICON is tracking are the markers, not the actual robots and objects.

The actual label and defining of the object and robot will be done via VICON NEXUS, which is a software from the same company. Accessed data will be transmitted and recieved with virtual reality peripheral network client package, `vrpn_client_ros`, which then publish the `geometry_msgs` as a topic.

3.2 Modifications

While combining the robot parts in HUR, prefixes were given to UR5 and Robotiq in order to avoid the conflict in between them by some links sharing same namespaces. In previous study prefixes were `ur5` and `gripper` which later change into `ur5_` and `gripper_`. When first spawned the HUR representation from the previous study, there was a vibration occuring at a joint in between UR5 end effector link `ur5_ee_link` with `gripper_robotiq_hand_joint`. This issue was solved by disabling self collision of `ur5_ee_link`.

```
<gazebo reference="ur5_ee_link">
  <selfCollide>false</selfCollide>
</gazebo>
```

Because of added prefix `ur5_`, provided controllers were not properlly called, because they were still defined with the joint names without the prefix, when all the transmissions were automatically defined with the prefix by Xacro. For that matter, added a copies of controller YAML files with prefix on the joint names.

When tried to simulate the Robotiq in Gazebp for the first time, there were no Moveit configurations nor controller files like other two robots of HUR. There were example nodes worked on real robots, but they all required parameters that only available with real robots, like IP adress of the robot. First tried the method used in previous study. Generate a Moveit configurations for the Robotiq and see if Moveit is able to manipulate the Robotiq. To do so, added hardware interface definitions to a URDF using Xacro, added new group `gripper` in the SRDF, and generated controller definition in YAML for the gripper. The idea of using Moveit to manipulate the Robotiq worked, but it was hard to say the behaviors of simulated and the real robots are alike. While real robot can be manipulated by sending a commands directly to the robot, simulated one had to use Moveit instead. And manipulation through Moveit was only able to close and open the gripper, which is limitting all the capability of the Robotiq which allows to manipulate individual fingers, close and open the gripper to a designated portion and use of different modes.

Later found out there was a provided gazebo plugin in one of the provided packages `robotiq_3f_gripper_articulated_gazebo_plugins`. But it was not suitable to apply directly, so created a package `robotiq_gazebo_plugins` with `RobotiqGazeboPlugin`, which modified the `RobotiqHandPlugin` from the provided package. The original plugin was assuming there were two grippers, for left and right arm each. During the modification, found out that actuating bars of Robotiq must be presented in order to manipulate it with the plugin. For that changed used representation of Robotiq to `robotiq_hand_macro .urdf.xacro` from `robotiq-3f-gripper_articulated_macro.xacro` which only represented the fingers not the actuating bars.

Since new actuacing bar links were added to HUR representation, it was necessary to define a new SRDF. Previously, There were groups defined for all Husky, UR5, and Robotiq, but since UR5 is the only robot manipulated with Moveit, deleted groups husky but left group gripper since it need to be represented as a end effector of UR5:

```
<group name="gripper">
    <link name="ur5_ee_link" />
</group>
```

Before the modification, group ur5 was defined as collection of joints from shoulder_pan_joint to link-tool0_fixed_joint but decided to followed style used in original UR5's SRDF, so changed group ur5's name to arm and changed its formation as a chain from base_link to ee_link just like single UR5 is defined:

```
<group name="arm">
    <chain base_link="ur5_base_link" tip_link="ur5_ee_link"/>
</group>
```

And added group states home and up for group arm as a test measure, which are list of joints with defined state. To generate new collision information, used Moveit Setup Assistance.

Chapter 4

Visualization and Simulation

This chapter will provide infomations about the ways how to simulate the constructed robot representations and how to visualize both real robot and the simulated robot.

To have a same scenario within the simulation, it is necessary to define the objects for the simulation as well. For the thesis, acquired beer and table Simulation Description[Opea] and modified them in a shape that grippers of the robots can grab. like URDF, SDF is written in XML, containing information of the object's dimension, struction, material, and frictions.

For visualizing the real or simulated environment, robot models and their properties used the software Rviz. With Rviz it is possible to visualize not only the visible representations of the robot, but invisible quantitative properties, for example the coordinate frames of the robot joints, robot's goal position, the route to the goal, and occupancy maps.

With Rviz it is possible to visualize real robot and simulated robot, but rviz itself can not be worked as a simulator. To simulate the generated robot representations, used a software Gazebo. By spawning the defined robot URDF model, it is possible to spawn and interact with the robot model within the simulation. In the real environment, VICON is used to get the object's ground truth coordinate, orientation, and dimension. To have the similar environment as in real laboratory in the simulation, used the Gazebo plugin. `GazeboRosP3D` controller is one of the Gazebo provided plugins used to publish ground truth information of the robot and objects in simulation. To use the plugins, they need to be defined in the URDF of a subject that the ground truth information is wanted. For Tiago it was already implemented, publishing its ground truth to a topic `ground_truth_odom`, therefore added `GazeboRosP3D` plugins to HUR, `beer` and `table` representations:

```
<gazebo>
  <plugin name="ground_truth" filename="libgazebo_ros_p3d.so
   ">
    <frameName>map</frameName>
    <bodyName>base_link</bodyName>
    <topicName>base_link_ground_truth</topicName>
    <xyzOffsets>0 0 0</xyzOffsets>
    <rpyOffsets>0 0 0</rpyOffsets>
  </plugin>
</gazebo>
```

As explained in Chapter 3.2, the control of Robotiq in simulation is also done with the modified Gazebo plugin, `RobotiqGazeboPlugin`. The way how the plugin is presented is similar to `GazeboRosP3D`:

```
<gazebo>
  <plugin name="robotiq_gazebo_plugin" filename="
   libRobotiqGazeboPlugin.so">
    <kp_position>1.0</kp_position>
    <kd_position>1.0</kd_position>
    <prefix>gripper_</prefix>
  </plugin>
 </gazebo>
```

The PID values of `RobotiqGazeboPlugin` are not optimized, but it generally works fine.

Whether it is real environment with VICON, or simulated environment with `GazeboRosP3D`, the topic what both of them are publishing are not sufficient to describe the relations in between robot and objects. Whatever the joint or object it is, each individual parts have their own coordinate frames and these relationships between the frames is done with transform library(`tf`) package[Foo13]. What topics published by VICON and `GazeboRosP3D` have in common are `Pose` and `twist`. To express the robot and objects as a child frame of a fixed map, generated node, `model_tf_broadcaster`.

`model_tf_broadcaster` will subscribe all the ground truthes of robot and objects and reform them into `transformStamed` message and broadcast them using `tf` package.

Chapter 5

Experiments

In this chapter, the procedure of the experiment will be introduced, and the how to implement a state machine for specific given task.

5.1 Procedure

One of the basic tasks for the thesis is to implement a high-level control to pick an object at a known location with limited obstacles at known locations. For this specific task, set up a plastic water bottle standing upon a rectangular table where all the dimensions and locations are known.

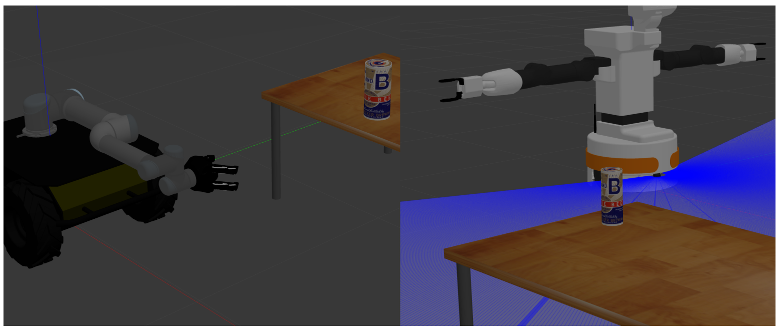


Figure 5.1: Simulated scenario for HUR and Tiago

By high-level control, instead of design the whole manipulation of the robot in a finer level language where its only understandable by the robot but in a instinctively understandable commands[SKS16].

The previous study's HUR behavior is not defined as a state machine, but it was following certain state orders:

- Send Husky to a calculated point, parallel to the closest table edge from the bottle.
- move Husky along the parallel axis to the point where Husky and the bottle are the closest or axis between them is perpendicular to the edge axis.
- sends UR5's end effector to bottle.
- Initiate the Robotiq to grab the bottle.
- Move away from the table following the parallel path continuing from the approaching path.

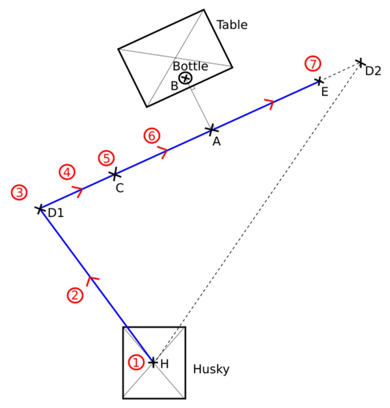


Figure 5.2: Diagram of the path, *Enzo Geromin*

Fig. 5.2 shows the how the path of the HUR was defined in previous study and it was planned in a certain numerical order.

Base on the work from the previous study, tried to implement state machine on Tiago. There are ROS packages used to state machinize the procedure like SMACH and SMACC. The developer of SMACC clames it has a better agility because it is C++ oriented, but due to personal lack of knowledge in C++, choose SMACH[BC10] which is Python oriented.

SMACH is initialized by stating the possible outcomes of the state machine itself, and than each states in the state machines with its possible outcomes and transitions are enlisted:

```
self.sm = smach.StateMachine(outcomes=['succeeded',
                                       'failed'])

smach.StateMachine.add('HEAD_DOWN', HeadDown(),
                   transitions={'succeeded':'MOVE_TO_BOTTLE',
                                'aborted':'failed'})
...
smach.StateMachine.add('GRIPPER_OPEN', gripperOpen(),
                   transitions={'succeeded':'ARM_BOTTLE',
                                'aborted':'failed'})
```

Each states are defined as a class, and defined as:

```
class HeadDown(smach.State):
    def __init__(self):
        smach.State.__init__(self,
                             outcomes=['succeeded',
                                       'aborted'])
        ...

    def execute(self, userdata):

        ...

        return 'succeeded'
```

Each state will be executed and return its outcome, and will execute the following state according to defined transission followed by the specific outcome.

Chapter 6

Conclusions

Through the thesis, it showed how HUR and Tiago are represented, and possible ways how to manipulate them through different packages like controllers package, Moveit, and Gazebo plugins. And it was able to state how environments were set up, for both in the physical world and simulated world.

Unfortunately, due to the current pandemic situation, the thesis was more simulation-oriented than expected, in the future the methods used in the thesis can be tested on real robots to see the compatibility of them.

There are uncleared and unfinished points for the thesis, like object recognition and 3D mapping, they can be tried later on if the methods are valid.

One of the points not mentioned in the thesis is `moveit_servo`, the package also provided by Moveit, it was possible to generate extra `joint_group_position_controller` to manipulate the UR5 and Tiago arm using `moveit_servo` package, but unable to solve unexpected behavioral problems and was not able to use as the primary controller. If further studies were made it could be possible to have a quick actuator response than Moveit.

Chapter 7

Source Code

This chapter explains about the attached source codes of the thesis.

In the attachement, it holds ROS packages divided into two groups: HUR and Tiago.

For HUR the integration of statemachine node and robot initialization nodes was not implemented yet. It is required to roslaunch the launch file using command:

```
roslaunch husky_ur5_gripper_demos pick_the_bottle.launch
```

and ros run the following command:

```
rosrun husky_ur5_gripper_demos pick_the_bottle.py
```

For Tiago by launching `manipulator.launch` in `tiago_dual_demos` package it is possible to initiate the robot and the state machine. This launch file also has arguments like `moveit_servo` which turns of the original left arm controller and starts the controller for the `moveit_servo` and arguments `rviz` and `gazebo` which can turn on or off the GUIs.

For both robots, the compatibility with real robots were not tested properly.

Appendix A

Bibliography

[BC10] Jonathan Bohren and Steve Cousins, *The smach high-level executive [ros news]*, IEEE Robotics Automation Magazine **17** (2010), no. 4, 18–20.

[CMEM^{+}17] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo, *ros_control: A generic and simple control framework for ros*, The Journal of Open Source Software (2017).

[CR15] Inc. Clearpath Robotics, *husky_description*, Source code, 2015, `https://github.com/husky/husky/tree/melodic-devel/husky_description`.

[CSCC14] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll, *Reducing the barrier to entry of complex robotic software: a moveit! case study*, 2014.

[Foo13] Tully Foote, *tf: The transform library*, Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, Open-Source Software workshop, April 2013, pp. 1–6.

[Ltd] Vicon Motion Systems Ltd, *Vicon*, Website, `https://www.vicon.com/`.

[Opea] Open Source Robotics Foundation, Inc, *gazebo_models*, Website, `https://github.com/osrf/gazebo_models`.

[Opeb] Open Source Robotics Foundation, Inc., *Tutorial: Using a urdf in gazebo*, Website, `http://gazebosim.org/tutorials/?tut=ros_urdf`.

[Ope14] Open Source Robotics Foundation, Inc., *Migrating from catkin_make, important distinctions between catkin_make and catkin build*, Website, 2014, `https://catkin-tools.readthedocs.io/en/latest/migration.html`.

[Roba] Robotiq, *3-Finger Adaptive Robot Gripper*, Website, `https://robotiq.com/products/3-finger-adaptive-robot-gripper`.

[Robb] Universal Robots, *UNIVERSAL ROBOT UR5e*, Website, `https://www.universal-robots.com/products/ur5-robot/`.

[Rob15] Robotiq, *robotiq_3f_gripper_visualization*, Source code, 2015, `https://github.com/ros-industrial/robotiq/blob/kinetic-devel/robotiq_3f_gripper_visualization`.

[ROSa] ROS.org, *Husky*, Website, `https://wiki.ros.org/Robots/Husky`.

[ROSb] ROS.org, *robotiq*, Website, `https://wiki.ros.org/robotiq`.

[ROSc] ROS.org, *TIAGo++*, Website, `http://wiki.ros.org/Robots/TIAGo%2B%2B`.

[ROSd] ROS.org, *universal_robots*, Website, `https://wiki.ros.org/universal_robots`.

[ROSe] ROS.org, *xacro*, Website, `http://wiki.ros.org/xacro`.

[SKS16] Philipp Schillinger, Stefan Kohlbrecher, and Oskar Von Stryk, *Human-robot collaborative high-level control with application to rescue robotics*, 2016 IEEE International Conference on Robotics and Automation (ICRA) (2016).

[Smi] R. Smits, *KDL: Kinematics and Dynamics Library*, `http://www.orocos.org/kdl`.

Appendix B

Nomenclature

Word	Meaning
HUR	A robot model built upon Clearpath's Husky base with Universal Robot's UR5 arm and Robotiq's 3-finger gripper.
Tiago	In general, it means a PAL Robotic's one-armed robot TIAGo, but on this thesis, it is refered to TIAGo++.
SMACH	Statemachine modular Python library

Appendix C

Abbreviations and Acronyms

Abbr.	Meaning
ROS	Robot Operating System
URDF	Unified Robot Description Format
XML	Extensible Markup Language
Xacro	XML Macros
SRDF	Semantic Robot Description Format
SDF	Simulation Description Format
COLLADA	COLLAborative Design Activity. Its file extension is .dae
STL	Stereolithography
YAML	YAML Ain't Markup Language

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Min Joonhong**	Personal ID number:	**371323**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Measurement**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Mobile manipulation in cluttered environment

Master’s thesis title in Czech:

Mobilní manipulace v prostředí s překážkami

Guidelines:

1) Create a URDF model of the Husky + UR-5 + Robotiq hand. Implement the necessary algorithm, so that the simulated robot and the real one can be programmed with the same piece of software.
2) Implement a state-machine for high-level control of the robotic manipulator to pick an object at a known location with a limited number of obstacles with known location.
3) Implement a 3D mapping algorithm.
4) Implement a trajectory planner to pick an object at a known position while avoiding collision with the detected obstacles.
5) Test algorithms in the simulator and the real robot.
6) Optionnaly, implement an algorithm that localizes the object to be picked.

Bibliography / sources:

[1] ROS URDF Tutorials: http://wiki.ros.org/urdf/Tutorials
[2] Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B. & Konolige, K. The Office Marathon: Robust navigation in an indoor office environment IEEE International Conference on Robotics and Automation, 2010, pp. 300-307
[3] Chitta, S.; Sucan, I. & Cousins, S. MoveIt! IEEE Robotics Automation Magazine, 2012, 19, pp. 18-19
[4] Conner, D. C. & Willis, J. Flexible Navigation: Finite state machine-based integrated navigation and control for ROS enabled robots SoutheastCon, 2017
[5] Guimarães R.L., de Oliveira A.S., Fabro J.A., Becker T., Brenner V.A. (2016) ROS Navigation: Concepts and Tutorial. In: Koubaa A. (eds) Robot Operating System (ROS). Studies in Computational Intelligence, vol 625. Springer

Name and workplace of master’s thesis supervisor:

Dr. Gaël Pierre Marie Ecorchard, Intelligent and Mobile Robotics, CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **18.09.2020** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until:
by the end of summer semester 2021/2022

Dr. Gaël Pierre Marie Ecorchard Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature