Object Detection and Grasping

MSc. Robotics

Simon Bøgh

Associate Professor

Faculty of Engineering, Department of Materials and Production

Spring 2020

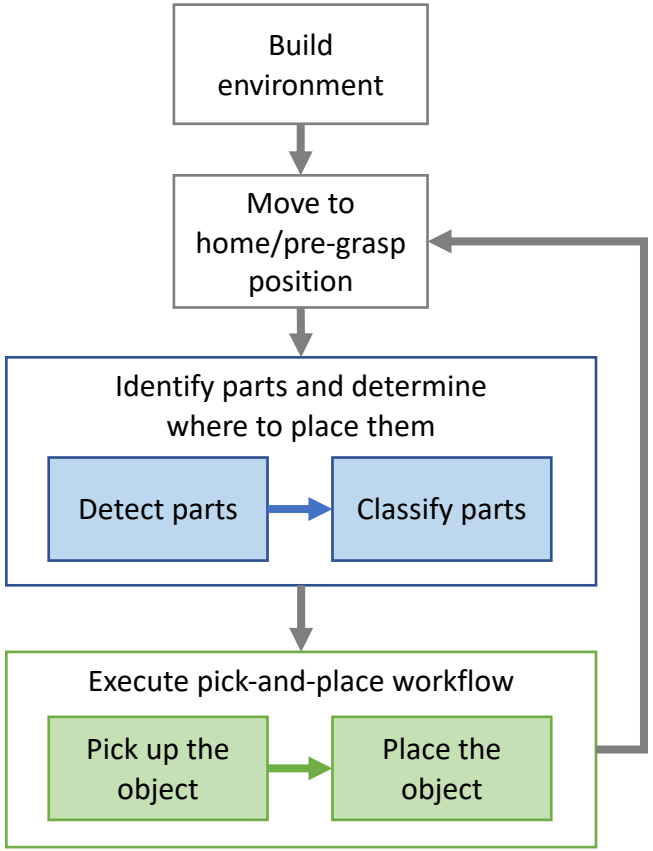
AALBORG UNIVERSITY
DENMARK

Outline

- Manipulation overview
- Pose estimation with logical camera
- Logical camera Gazebo plugin
- Frames and transforms with ROS tf and tf2 packages
- Grasping with a vacuum gripper

Introduction

- So far we have worked with
  - URDF and XACRO modeling
  - World setup – Add robots and objects to the OMTP Factory
  - Set up our robots for motion planning with MoveIt
- We can now do pre-programmed manipulation e.g. pick and place locations
  - MoveIt Commander
- Manipulation in dynamic environments
  - Use cameras to detect, recognize and estimate the poses of objects of interest



Manipulation - Overview

- Manipulation in fixed/static environments
  - Robot arms moved to pre-programmed pick and place locations
  - Traditional automation (e.g. car manufacturing)
- Manipulation in dynamic environments
  - Use cameras to detect, recognize and estimate the pose of the objects of interest

http://motioncontrolsrobotics.com

Bin-picking Robot Deep Learning

Manipulation – main idea

- Robot vision (perception): Inspired by image processing in computer science
  - Detection, recognition and pose estimation using cameras (2D/3D)
  - This course will not cover this in deep detail
- This course: **Logical camera** in our OMTP factory simulation
  - Detects, recognizes and provides **pose of the object of interest**
  - Integrate this information with the simple pick and place pipeline with **ROS tf/tf2** package

Manipulation – goals for this lecture

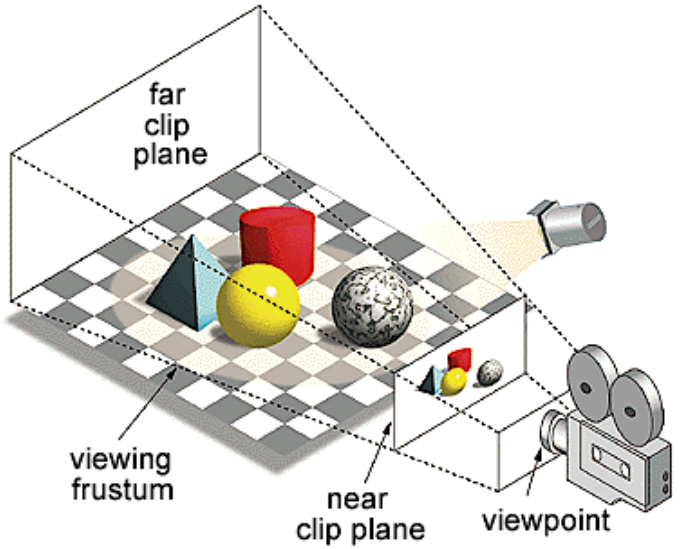
- Add (a) logical camera(s) to the factory
  - Inspect and use the logical camera data
- Basic concepts of ROS tf package (transforms)
  - Specify poses in a **reference frame**
  - Create and view the ROS **TF tree**
  - **Transforms pose** from a given reference frame to a desired reference frame
- Use vacuum gripper plugin to grasp objects

Logical camera

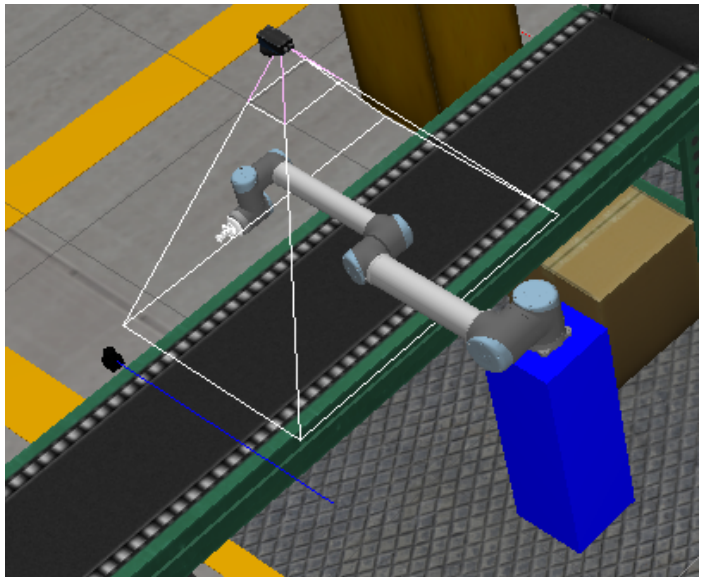
A special kind of camera

What is a logical camera?

- While a **camera outputs an image**, a **logical camera outputs model names and poses**
- It shows which models might be visible to a camera in the same location
- I.e. the intersection of **camera viewing frustum** with the object

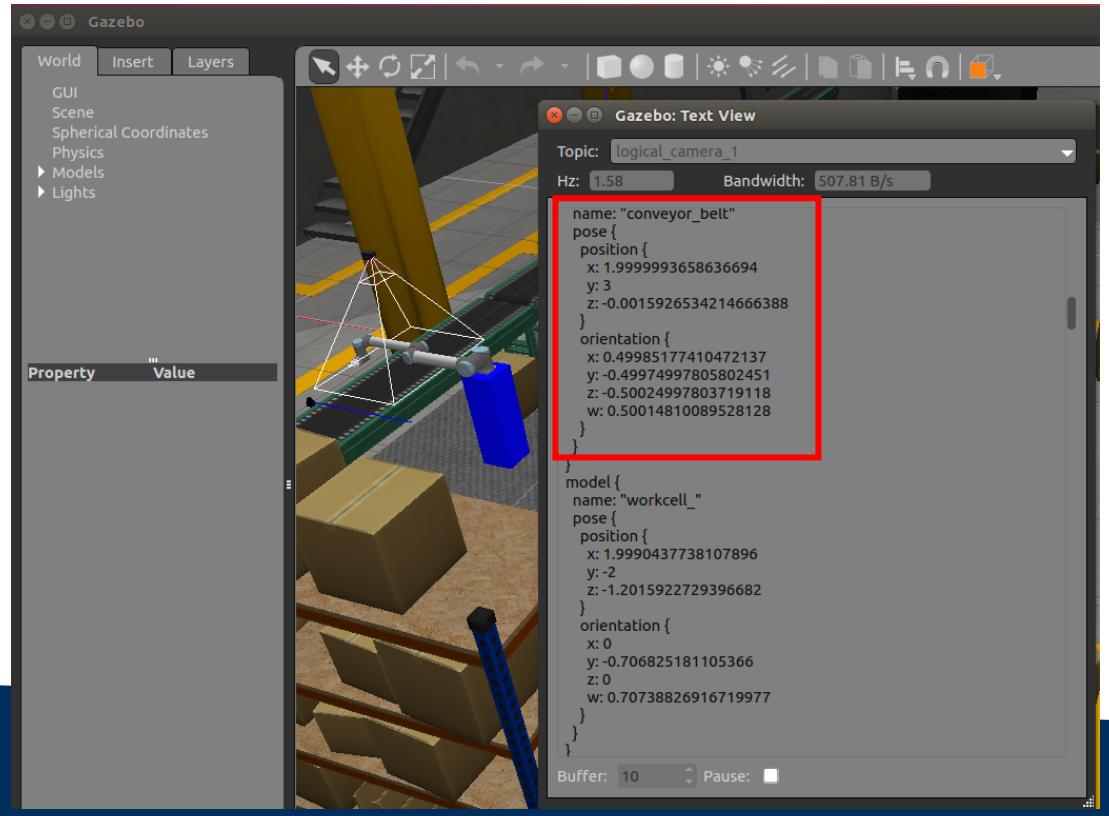


https://encyclopedia2.thefreedictionary.com/View+frustum



Logical camera output

- Logical camera **outputs model names and poses** (position and orientation)
  - Models are created as .stl/.dae files (Meshlab, Blender, SolidWorks®)



Note!

- Logical camera is only a simulation concept
- Real world applications use 2D/3D cameras for robot vision
  - E.g. with open source libraries such as OpenCV and PCL
- This course
  - The pose of an object for manipulation is the main goal

Logical camera in the OMTP factory simulation

Gazebo world file: basics

- We create world models that can be used in the Gazebo physics simulator
  - New format instead of URDF: **s**imulation **d**escription **f**ormat (sdf) models
  - Multiple .sdf models composed in a world file
  - Similar to URDF models, but more features than URDF [1]
    - friction
    - non-robotic elements (cameras, light, sun …)
  - URDF and SDF can be used together
    - Spawn URDF models in gazebo with **spawner** ROS node

- Model functionality can be configured via Gazebo plugins
  - For example: Logical camera plugin (libROSLogicalCameraPlugin.so) [2]

[1] http://gazebosim.org/tutorials?tut=ros_urdf
[2] http://gazebosim.org/tutorials?tut=ros_gzplugins

SDF models for OMTP factory simulation

- Gazebo models we will use are in the updated **omtp_gazebo** package
  - omtp_gazebo/models
- Camera model files
  - omtp_gazebo/models/logical_camera/**model.sdf file**
- Conveyor object(s) to pick up
  - **config/conveyor_objects.yaml**
  - We will pick up a simple box object
  - You can modify this file to include your own objects

omtp_gazebo: models and plugins

- SDF models used in omtp.world (we will add logical camera)

```
sb@omtp:~/omtp_course_ws/src/omtp_course/omtp_gazebo/models$ tree -L 1
.
├── conveyor
├── deletion_wall
└── logical_camera
```

- Plugins used in the OMTP factory simulation

```
sb@omtp:~/omtp_course_ws/devel/lib$ ls -1 libROS*
libROSConveyorBeltPlugin.so
libROSLogicalCameraPlugin.so
libROSProximityRayPlugin.so
libROSVacuumGripperPlugin.so
```

Gazebo world file: omtp.world

- Definition of our world: **omtp_gazebo/worlds/omtp.world**
- Here the conveyor has been added
- Your task is to add (a) logical camera(s) with the correct pose

3D position and orientation with respect to a fixed reference frame e.g. the world

Orientation uses Euler one convention: roll, pitch and yaw (RPY)

```
<?xml version="1.0" ?>
<sdf version="1.6">
    <world name="omtp_world">

    <include>
        <uri>model://conveyor</uri>
        <pose>1.2 5 0 0 0 -1.571</pose>
    </include>

    <!-- include logical camera model(s) here -->

    </world>
</sdf>
```

Logical Camera Configuration

Logical Camera Configuration

- **Model file**: models/logical_camera/model.sdf
- The **model name** is “logical_camera”
  - When adding multiple objects, they need to have unique names
  - Therefore, when adding multiple logical cameras, give them different model names e.g. logical_camera_1
- We have a **plugin specification** that simulates the functionality of the logical camera
- **robotNameSpace**
  - If camera for example provides image information it will be under topic /<robotNameSpace>/logical_camera
- **Camera frustrum** is defined by *near* and *far*
- Logical camera can also **provide noise to measurements**

Logical camera loaded from .world file

- The camera model logical_camera_1 or 2 (model.sdf) is called from omtp.world
- Launch file uses the omtp.world to activate the environment including the cameras and the robots

lecture6_assignment1.world

omtp_gazebo > worlds > lecture6_assignment1.world

```
54      <include>
55        <uri>model://conveyor</uri>
56        <pose>1.2 5 0 0 0 -1.571</pose>
57      </include>
58
59      <include>
60        <uri>model://deletion_wall</uri>
61        <pose>1.2 -5.1 1.425 0 0 1.5708</pose>
62      </include>
63
64      <!-- Lecture 6 Assignment1 Part 1 -->
65      <!-- Include the two logical camera models here -->
66
67
68    </world>
69  </sdf>
```

lecture6_assignment1.world | model.sdf

omtp_gazebo > models > logical_camera1 > model.sdf

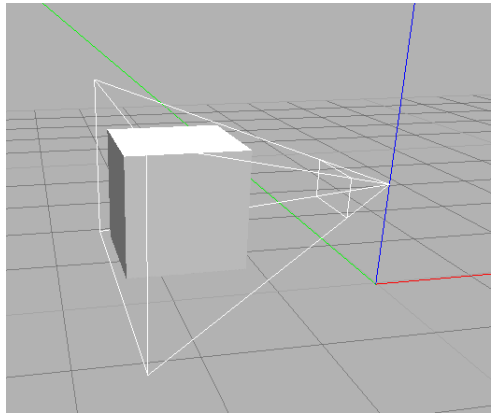
```
1   <?xml version="1.0" ?>
2   <sdf version="1.6">
3     <model name=<write your code here>>
4       <plugin name="ros_logical_camera_1" filename="libROSLogicalCameraPlugin.so">
5         <robotNamespace>omtp</robotNamespace>
6         <!-- <position_noise>
7           <noise>
8             <type>gaussian</type>
9             <mean>0.0</mean>
10            <stddev>0.001</stddev>
11          </noise>
12        </position_noise> -->
13        <!-- <orientation_noise>
14          <noise>
15            <type>gaussian</type>
16            <mean>0.0</mean>
17            <stddev>0.01</stddev>
18          </noise>
19        </orientation_noise> -->
20      </plugin>
```

Access information of the camera

- Logical camera outputs model names and poses of objects
  - **Models[]** array contains all objects found
  - Poses contain **position and orientation information**
  - The pose is **published as a ROS topic**
  - It uses the **logical camera image msg type**
  - The model message type **only contains a pose,** and **no reference information** as the PoseStamped message
  - Part of your assignment is to provide timing and reference frame information so the objects can be used

```
$ rosmsg show omtp_gazebo/LogicalCameraImage -r
# Logical camera image message
Model[] models                # models detected (poses in the frame of the camera)
geometry_msgs/Pose pose   # camera pose
```

logical_camera_frame

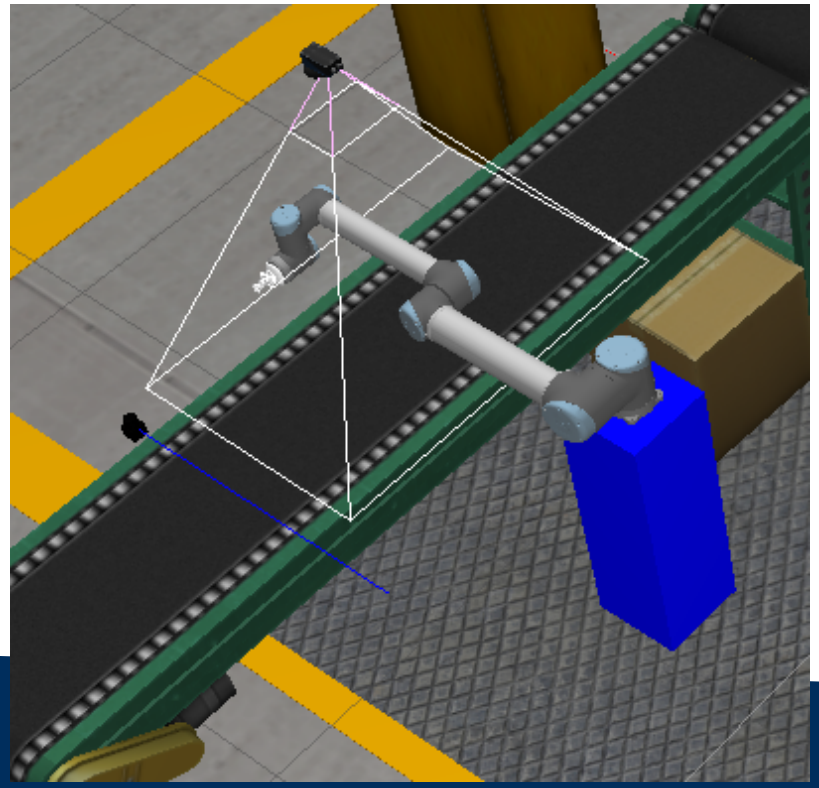


```
$ rosmsg show omtp_gazebo/Model
string type                          # object name
geometry_msgs/Pose pose
        geometry_msgs/Point position
                float64 x
                float64 y
                float64 z
        geometry_msgs/Quaternion orientation
                float64 x
                float64 y
                float64 z
                float64 w
```

Logical camera topic

- Logical camera data: object(s) in Models[]

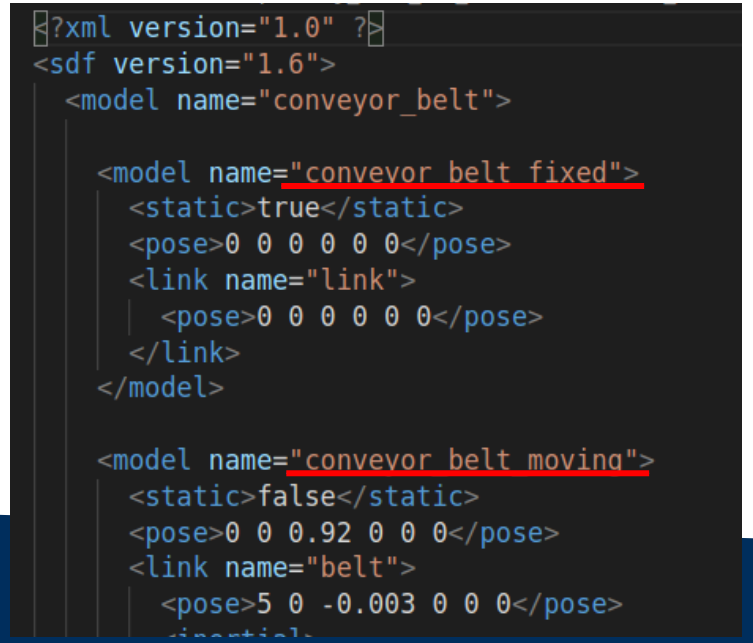
```
$ rostopic echo /omtp/logical_camera
```



```
models:
  -
    type: "conveyor_belt"
    pose:
      position:
        x: 1.99999936586
        y: 3.0
        z: -0.00159265342147
      orientation:
        x: 0.499851774105
        y: -0.499749978058
        z: -0.500249978037
        w: 0.500148100895
  -
    type: "conveyor_belt_fixed"
    pose:
      position:
        x: 1.99999936586
        y: 3.0
        z: -0.00159265342147
      orientation:
        x: 0.499851774105
        y: -0.499749978058
        z: -0.500249978037
        w: 0.500148100895
  -
    type: "conveyor_belt_moving"
```

Logical camera data – model name

- Logical camera knows the names of objects
  - In the conveyor belt model.sdf we defined a *model name*
  - Logical camera publishes the name information
  - **Unknown objects** e.g. the ones we spawn on the conveyor, are **published just as "object"**



ROS TF

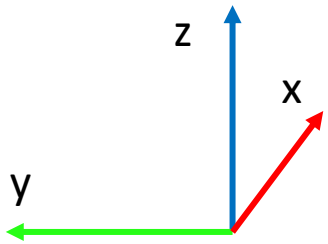
How to use the Logical Camera information

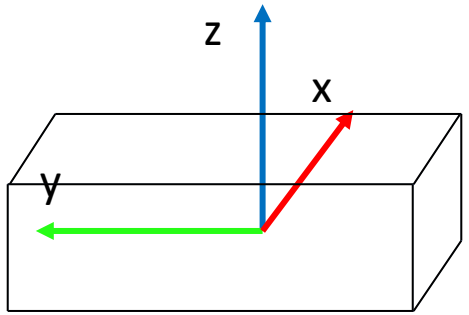
Finding reference information

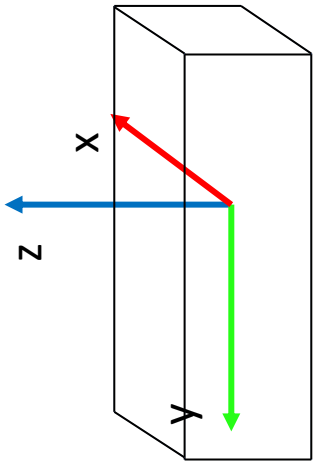
- Goal
  - Connect the logical camera information with our robots
- In order for us to manipulate the objects that the camera sees we will use the ROS tool **tf**
- Very important to achieve the goal of manipulating the objects
  - We add timing and reference information

ROS tf main idea

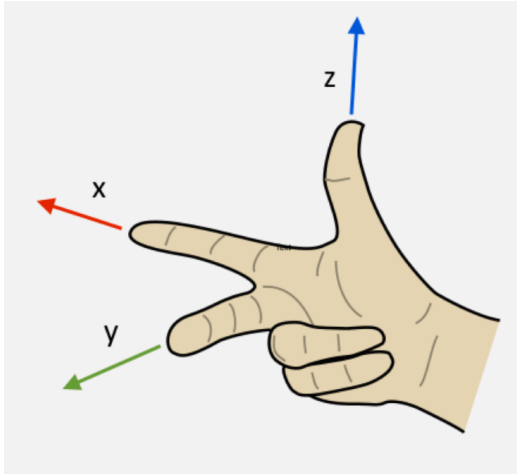
- tf keeps track of spatial and temporal relationships between different objects in our environment
  - Done by using the concept of reference frames in 3D
  - Examples of reference frames are seen below
    (including rotation)
  - ROS uses the *Right Hand Rule* convention





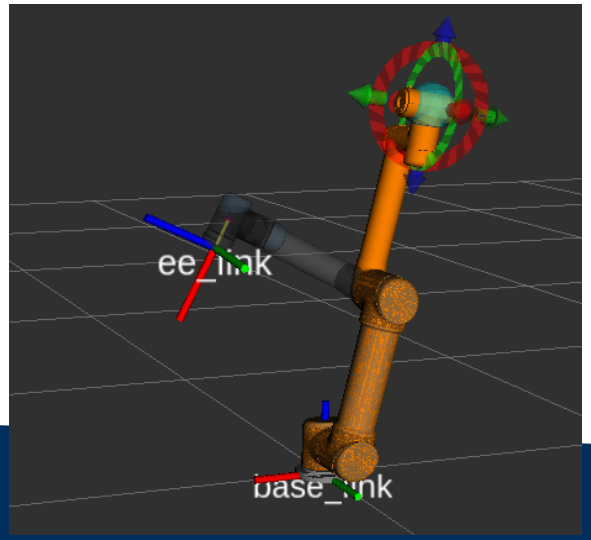


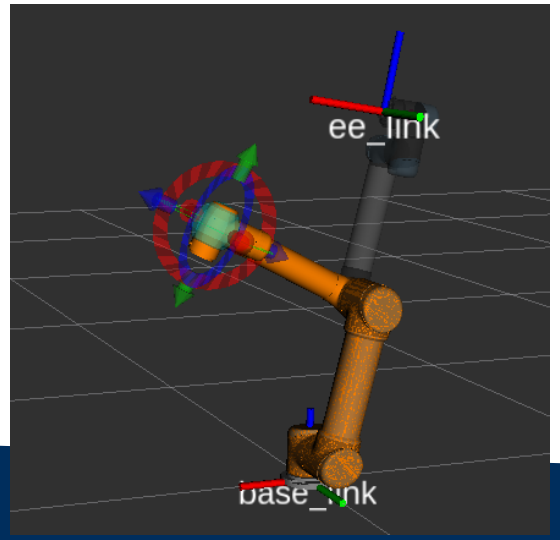
Right Hand Rule



ROS tf main idea

- Maintain relationship between multiple coordinate frames over time
- Transform points, or vectors between two coordinates
  - Published to the system which can be accessed by any node subscribed to it
- **Note!** tf-package is deprecated in favor of the more powerful tf2_ros package





http://wiki.ros.org/tf/Tutorials

How are tf reference frames generated?

- ROS package: robot state publisher (robot_state_publisher)
  - Generates and updates reference frames
  - Uses two sources of information as input
    - Movable joints i.e. joint state information: /joint_states topic
    - robot_description parameter: URDF/XACRO (omtp_factory.xacro)
  - Fixed frame of reference e.g. root link of the XACRO file: “world”
    - Typically it is used as fixed reference, but not necessarily e.g. with a mobile robots

Joint state information

- In the OMTP factory we combine the joint state information of the two robot arms

omtp_gazebo/launch/omtp_environment.launch

```
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <rosparam param="source_list">[/robot1/joint_states,/robot2/joint_states]</rosparam>
</node>
```

- Several nodes can publish to the /joint_states topic at the same time

```
$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers:
 * /joint_state_publisher (http://localhost:11311)
 * /gazebo (http://localhost:11311/)

Subscribers:
 * /robots_state_publisher (http://localhost:11311/)
 * /move_group (http://localhost:11311/)
```

robot_description information

- The information is taken from the URDF/XACRO we created
  - URDF defines the environment geometry
  - The information is loaded onto the parameter server
  - We use the URDF to define where the different objects are with respect to each other

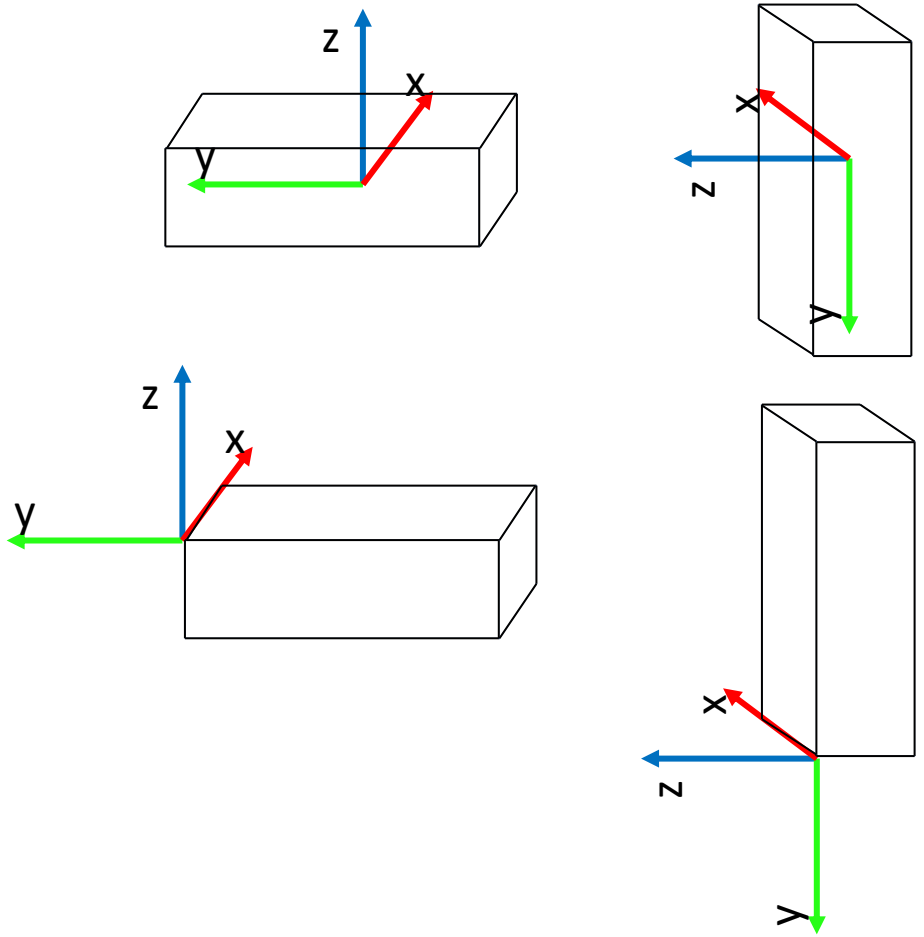
omtp_gazebo/launch/omtp_environment.launch

```
<!-- Load the URDF into the ROS Parameter Server -->
<include file="$(find omtp_support)/launch/load_omtp.launch"/>
```

```
<?xml version="1.0"?>
<launch>
<param name="robot_description" command="$(find xacro)/xacro '$(find
omtp_support)/urdf/omtp_factory.xacro'" />
</launch>
```

Where are reference frames located?

- All these reference frames are fine
- How do we know where they are?
  - Where do they get defined?



Reference frames: joints in XACRO

- tf helps us maintain and inspect relative information in our world
- XACRO-file
  - Objects are defined via **links**
  - Links are connected to each other via **joints**. This is how we know how to connect one object to each other
  - The **origin tag** defines where the tf frames are placed

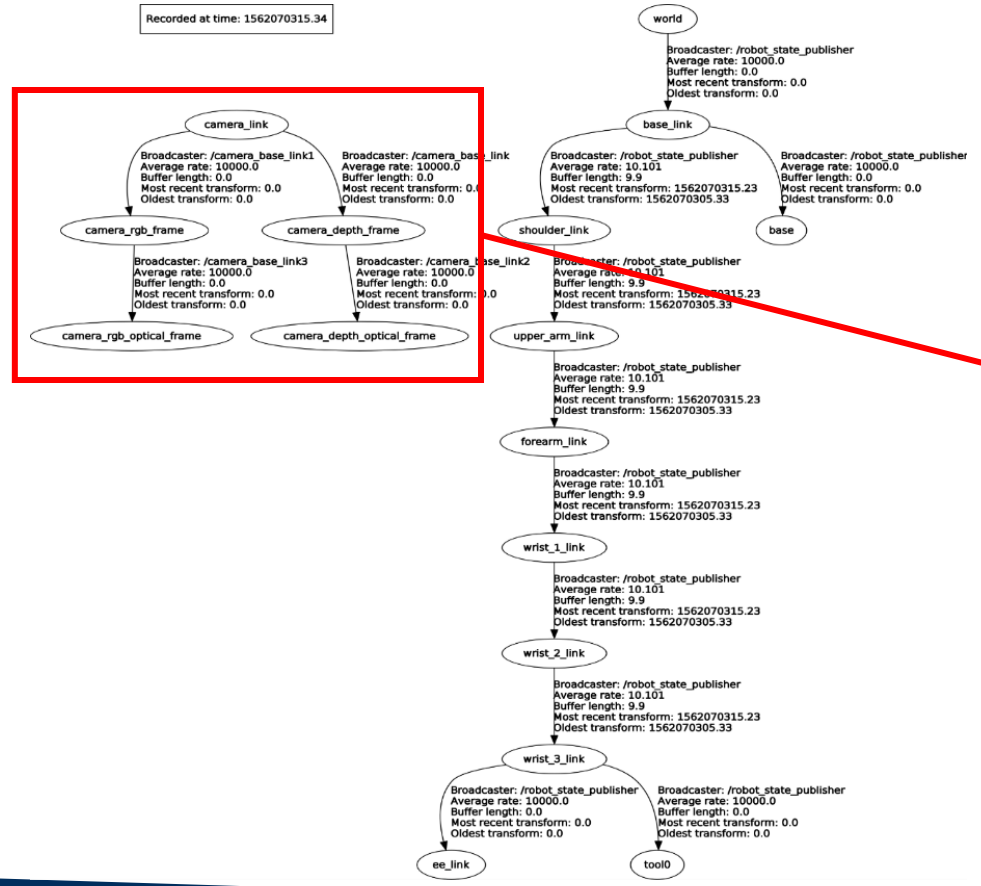
```
<!-- robot1 to pedestal. -->
<joint name="robot1_to_pedestal" type="fixed">
    <parent link="robot1_pedestal_link" />
    <child link="robot1_base_link" />
    <origin xyz="0 0 0.95" rpy="0 0 0" />
</joint>
```

Tf visualized in RViz

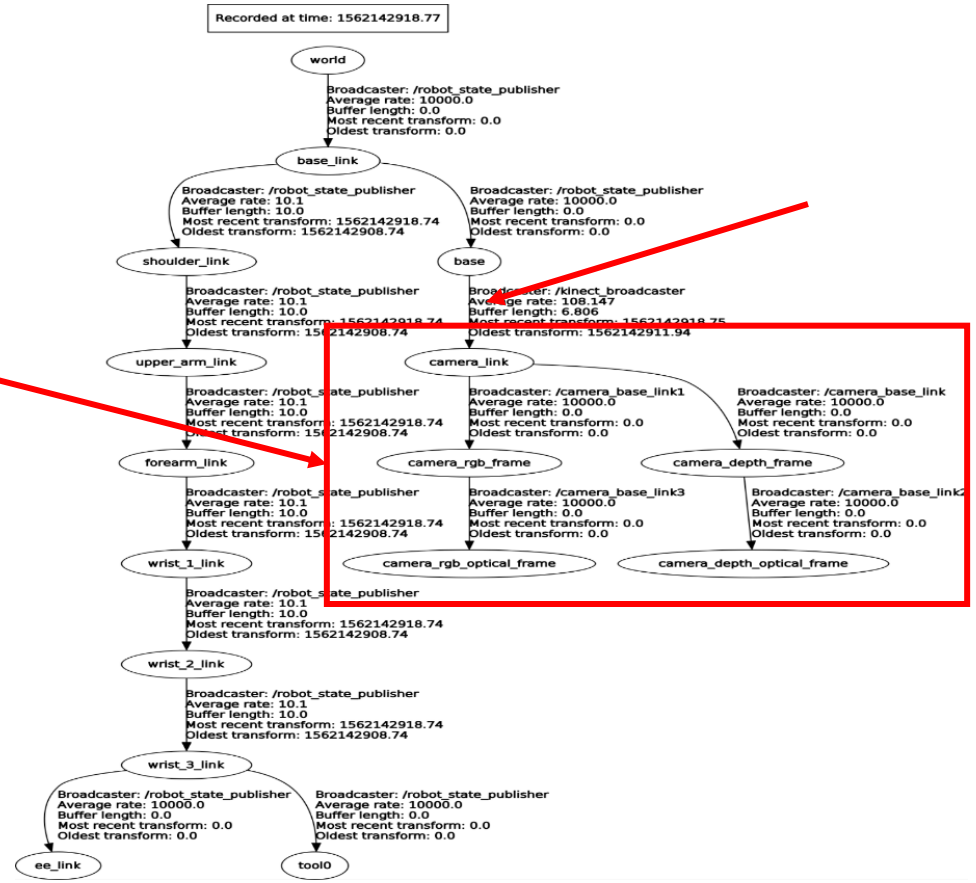
https://youtu.be/IMPik8V3nsI

Example of detached camera

Before calibration



After calibration



tf2 package

The second generation
of the transform library

tf2 transform library

- tf2_ros package: http://wiki.ros.org/tf2
  - tf2_ros is a ROS package that implements the functional aspects of tf
  - Actively **maintains spatial and temporal relationships** between reference frames
  - Takes care of the linear algebra computations to find the translational and rotational offsets between different objects
- **Transform** pose from a given reference frame to a target reference frame
  - For example: given a pose of an object in the logical camera reference frame, transform to the robot arm end-effector frame
- Advanced functionality
  - "time travel" functionality: look up spatio-temporal relationships between frames in the past (not covered in this course)

tf2 vs tf (”tf1”)

- Is there a tf1?
  - Yes, it was (and is) called just tf
  - tf or ”tf1” still works and is used in many applications
  - We will learn a few command line tools
  - All in one (vs. clear separation of functionalities in tf2 via packages)
- **tf2_ros** is recommended and all tf functionalities have been migrated to tf2
  - Main advantage: transform buffer to cache transforms over a certain duration

Tf2_ros: information representation

- How are the spatio-temporal relationships quantified?
  - **3D transformation**: **translation, rotation(quaternion)** and **timestamp** (**geometry_msgs/TransformStamped**)
  - ROS topic: /tf (tf2_msgs/TFMessage, array of TransformStamped)

- TF Command line tools
  - Start the OMTP simulation environment
    - Lower graphics demand with gui:=false argument

      $ roslaunch omtp_gazebo omtp_environment.launch gui:=false rviz:=false
    - Unpause Gazebo physics

      $ rosservice call /gazebo/unpause_physics
  - Start another terminal, source the workspace setup files

tf/tf2 command line tools

- Print transformation between source and target frames in the terminal

```
$ rosrun tf tf_echo <source_frame> <target_frame>
```

- View the TF tree – generate .pdf and .gv

```
$ rosrun tf view_frames
$ rosrun tf2_tools view_frames.py
```

- Publish static transform between a parent and a (new) child frame

```
$ rosrun tf2_ros static_transform_publisher <trans> <rot> “parent” “child”
```

tf_echo output

- Static transform: time is always 0.0

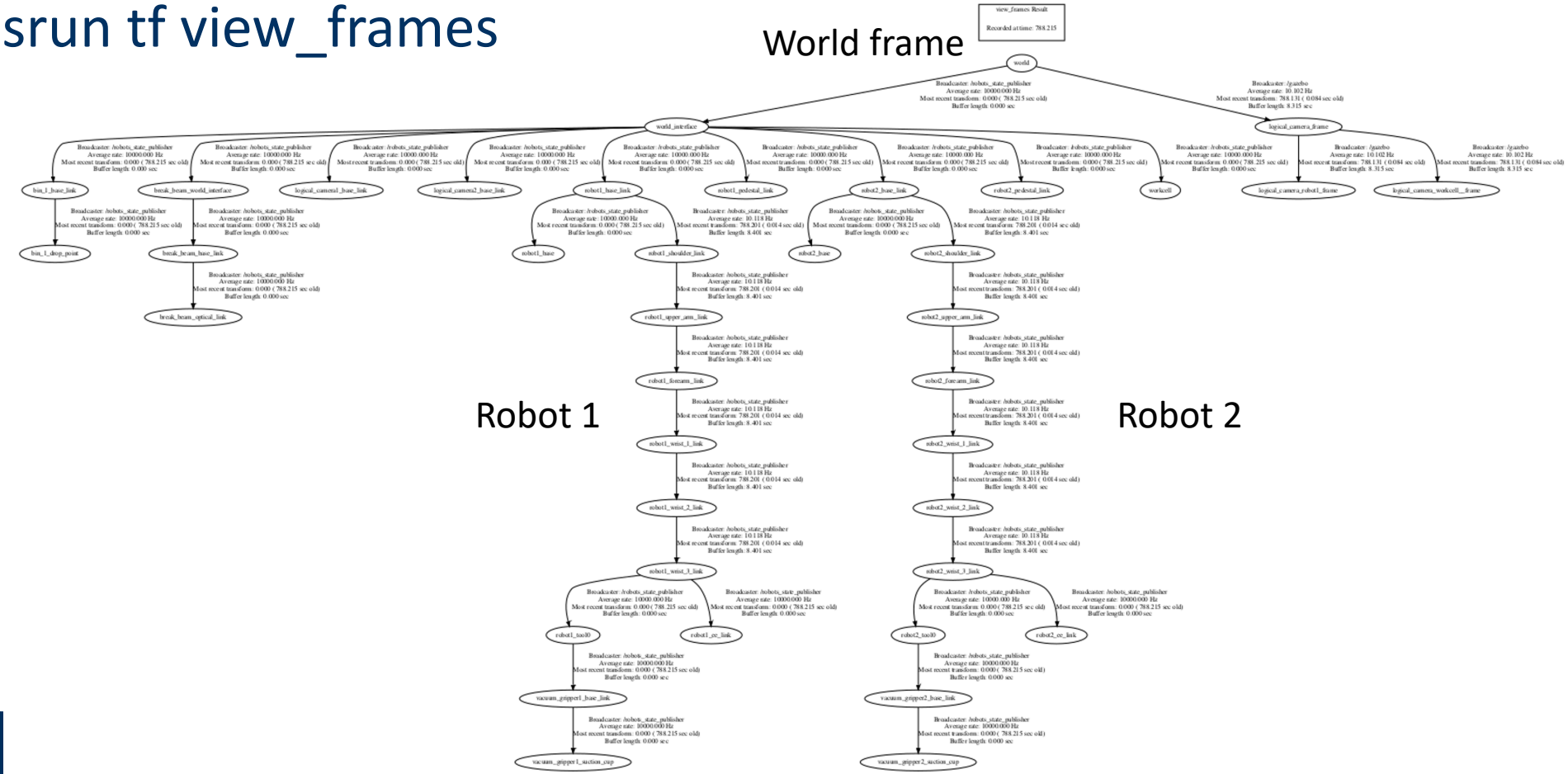
```
parallels@omtp:~$ rosrun tf tf_echo world robot1_pedestal_link
At time 0.000
- Translation: [0.500, 1.800, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
At time 0.000
- Translation: [0.500, 1.800, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
At time 0.000
- Translation: [0.500, 1.800, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
At time 0.000
- Translation: [0.500, 1.800, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
```

- Moveable joint: time is changing even though the robot is not moving

```
parallels@omtp:~$ rosrun tf tf_echo world robot1_forearm_link
At time 1159.701
- Translation: [1.043, 1.849, 1.359]
- Rotation: in Quaternion [0.000, 0.672, -0.000, 0.741]
            in RPY (radian) [-0.000, 1.473, -0.000]
            in RPY (degree) [-0.000, 84.401, -0.012]
At time 1159.701
- Translation: [1.043, 1.849, 1.359]
- Rotation: in Quaternion [0.000, 0.672, -0.000, 0.741]
            in RPY (radian) [-0.000, 1.473, -0.000]
            in RPY (degree) [-0.000, 84.401, -0.012]
At time 1160.702
- Translation: [1.043, 1.849, 1.359]
- Rotation: in Quaternion [-0.000, 0.672, 0.000, 0.741]
            in RPY (radian) [-0.000, 1.473, 0.000]
            in RPY (degree) [-0.000, 84.401, 0.012]
At time 1161.702
- Translation: [1.043, 1.849, 1.359]
- Rotation: in Quaternion [-0.000, 0.672, 0.000, 0.741]
            in RPY (radian) [0.000, 1.473, 0.000]
            in RPY (degree) [0.000, 84.401, 0.012]
```

view_frames output PDF

$ rosrun tf view_frames



tf2_ros API

- Buffer 10 seconds from tf (10 seconds default)

```
tfBuffer = tf2_ros.Buffer()
```

- Listen for updates in the buffer

```
tf2_ros.TransformListener(tfBuffer)
```

- Lookup transform between two frames in the buffer

```
transform_r1 = tfBuffer.lookup_transform(“transform_1”, ”transform_2”)
```

- Now you can look up the values of the transform

```
transform_r1.transform.translation.x
transform_r1.transform.translation.y
…
```

lookup_transform with logical camera

- Example of **detecting objects on the conveyor** with the logical camera
- Launch omtp environment (and unpause physics)

```
$ roslaunch omtp_gazebo omtp_environment.launch
```

- Move robot to upright position using moveit commander

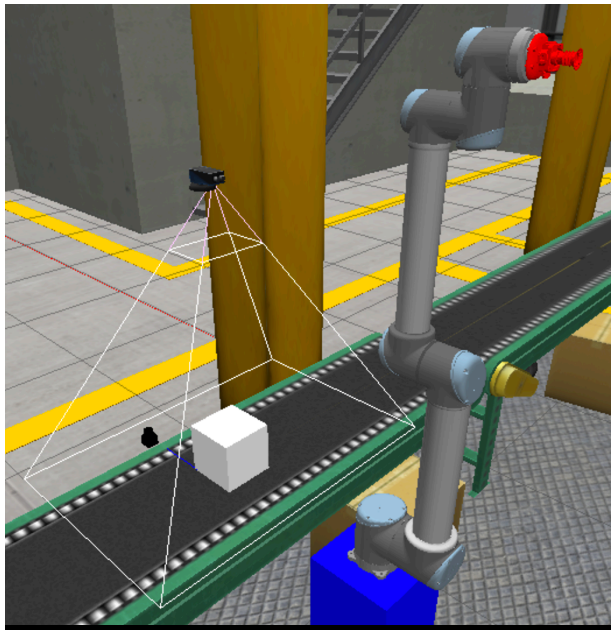
```
> use robot1
> go R1Up
```

- Spawn an object on the conveyor

```
$ rosservice call /spawn_object_once
```

- Check the output from the logical camera detection. Look for “object” in the output

```
$ rostopic echo /ompt/logical_camera
```



```
type: "object"
pose:
  position:
    x: 0.979459744622
    y: 0.000368919496984
    z: -1.45961158144e-05
  orientation:
    x: -0.707889709741
    y: -0.00175580432109
    z: -0.000573341045375
    w: 0.706320569765
```

Script: transform_object_pose.py

- omtp_lecture6/scripts/transform_object_pose.py
- Transform from logical camera reference frame to the world reference frame

```
object_world_pose = tf_buffer.transform(object_pose, "world")
```

```
# The logical camera always outputs the pose of an object it its own
# reference frame and here it is called logical_camera_frame
# If we have more than one logical camera, this name will change
object_pose.header.frame_id = "logical_camera_frame"
object_pose.pose.position.x = data.models[-1].pose.position.x
object_pose.pose.position.y = data.models[-1].pose.position.y
object_pose.pose.position.z = data.models[-1].pose.position.z
object_pose.pose.orientation.x = data.models[-1].pose.orientation.x
object_pose.pose.orientation.y = data.models[-1].pose.orientation.y
object_pose.pose.orientation.z = data.models[-1].pose.orientation.z
object_pose.pose.orientation.w = data.models[-1].pose.orientation.w
while True:
    try:
        object_world_pose = tf_buffer.transform(object_pose, "world")
```

Script: transform_object_pose.py

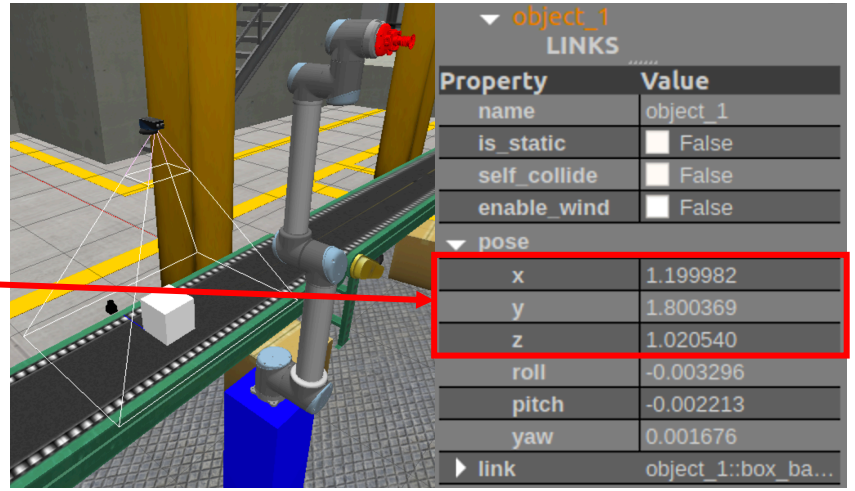
- Run the script to detect and transform from logical camera frame to world frame

```
$ rosrun omtp_lecture6 transform_object_pose.py
```

transform_object_pose.py output

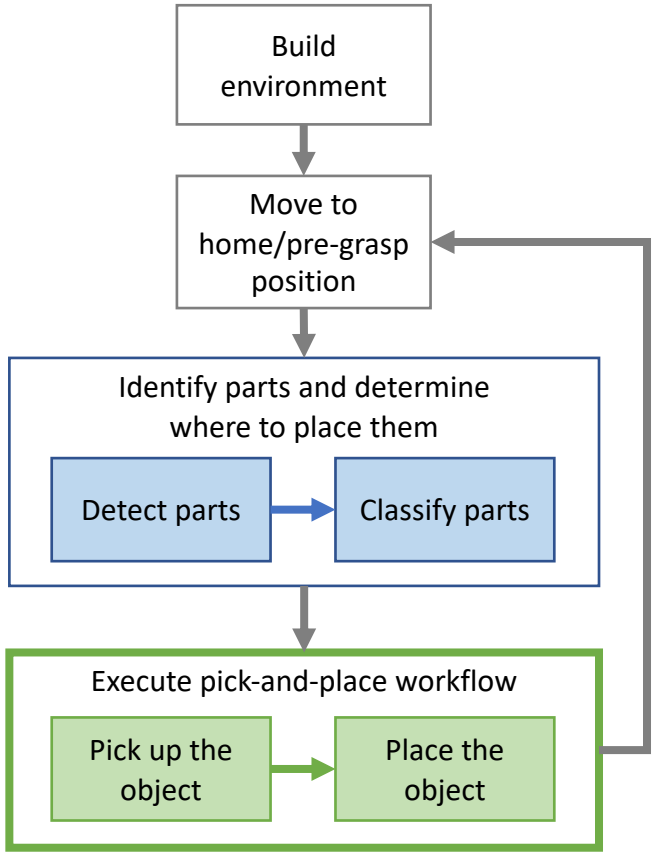
```
  seq: 0
  stamp:
    secs: 247
    nsecs: 694000000
  frame_id: "world"
pose:
  position:
    x: 1.19998180613
    y: 1.8003689195
    z: 1.02054025544
  orientation:
    x: -0.500958108857
    y: 0.49820344299
    z: 0.500149120811
    w: 0.500684690714
```

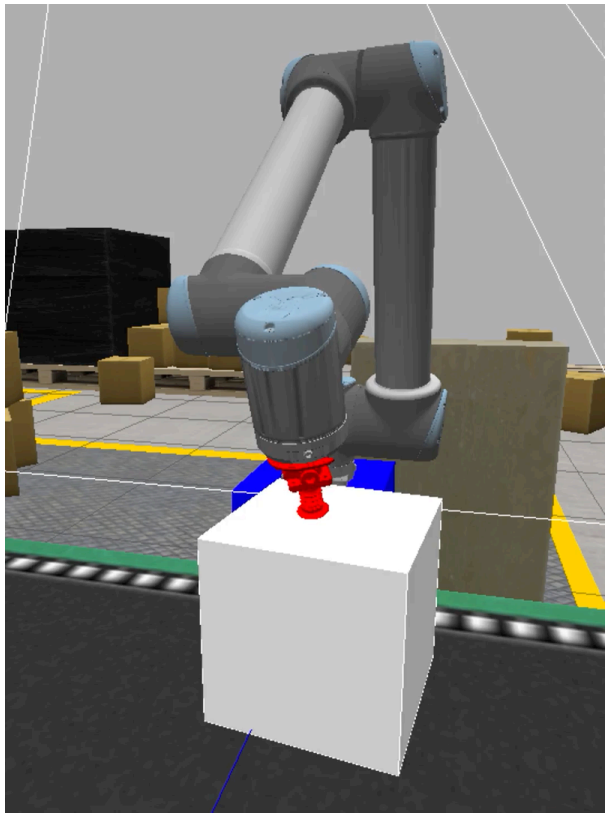
Gazebo



Grasping in OMTP Factory

Grasping pipeline





Grasping the object

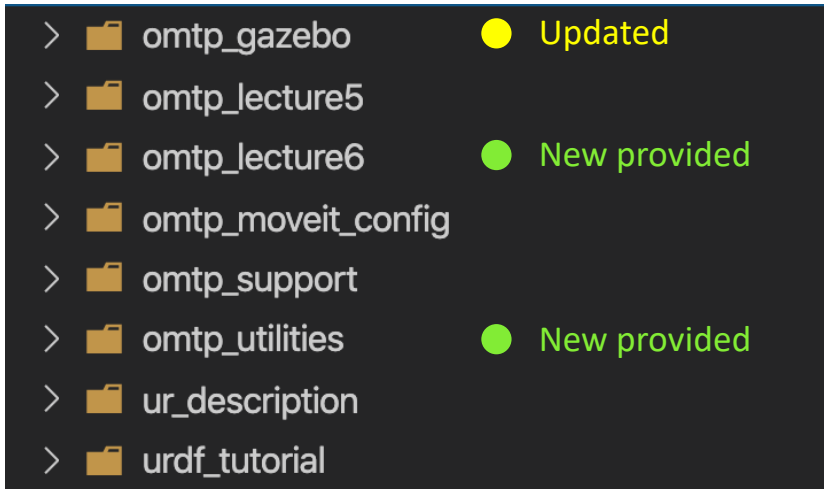
1. Move robot 1 to R1PreGrasp as configured with MoveIt Setup Assistant
2. Spawn object on conveyor
   $ rosservice call /spawn_object_once
3. Get the object pose detected by the logical camera
   $ rostopic echo /omtp/logical_camera
4. Activate suction gripper
   $ rosservice call /gripper1/control {enable: true}
5. Make cartesian move until in contact with object. When in contact the object gets attached to the gripper
6. Pick up object and move it to a place location e.g. a bin

Recap

- How to use the logical camera
- Reference frames and we can use them in ROS with the tf2 package
- Transform the pose found by the logical camera to the world frame
- This is useful since the end-effector of the robot is specified in the world frame
- Activate the tool to pick up the object

Assignment files

- **omtp_gazebo** updated with logical camera models and one new .world file for this lecture
- **omtp_utilities** contains script to spawn objects on conveyor
- **omtp_lecture6** contains exercises



Assignment 1a – add two cameras

1. Two logical cameras have been included in **omtp_gazebo/models**
   - logical_camera_1 and logical_camera_2
2. Add the two logical cameras to a new world file named **lecture6_assignment1.world**, located in **omtp_gazebo/worlds**
   - Include the two cameras at the bottom of the .world file
   - Suggested poses for the logical cameras models
     - Camera 1: `1.2 1.8 2.0 0 1.57 0`
     - Camera 2: `-8.3 -1.23 1.8 0 1.57 0`

Assignment 1b – configure cameras

1. Configure the two cameras
   - Update the model.sdf files in the folders for each logical camera wherever you are instructed to <write_code_here>
   - Use the following configuration parameters

     Configuration for Logical Camera 1:

     - Model name: logical_camera_1 Link name: logical_camera_1_link Sensor name: logical_camera_1

     Configuration for Logical Camera 2:

     - Model name: logical_camera_2 Link name: logical_camera_2_link Sensor name: logical_camera_2
2. After you have completed the sdf files with the above configurations, start the factory simulation with:

   ```
   $ roslaunch omtp_lecture6 lecture6_assignment1.launch
   ```
3. Verify if the relevant topics are available using $ rostopic list

Assignment 2 – tf view_frames

1. Start the OMTP factory simulation

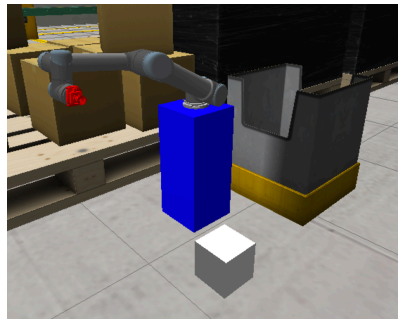
   ```
   $ roslaunch omtp_lecture6 lecture6_assignment1.launch
   ```

2. In the terminal execute

   ```
   $ rosrun tf view_frames
   ```

3. Open the PDF and verify that the two logical cameras are present in the tf tree
   - Include the tf tree in your git repo README.md as an image

Assignment 3 – transform object pose



1. Start the assignment 3 OMTP factory simulation

   ```
   $ roslaunch omtp_lecture6 lecture6_assignment3.launch
   ```

2. Spawn an object next to robot 2

   ```
   $ rosservice call /spawn_object_at_robot2
   ```

3. Complete the script lecture6_assignment3.py where you are instructed with <write your code here>.
   - Subscribe to the relevant topic, published by the logical camera 2, it would contain information about the object next to robot 2
   - Update the relevant reference frame for the pose of the object (Hint: You can use the view_frames command and TF tree to find out what name should be used for the reference frame)
   - Transform the pose of the object to the vacuum_gripper2_suction_cup reference frame
   - A few handy ROS command line tools for this assignment could be: rostopic info and rosmsg show

4. Run the completed script

   ```
   $ rosrun ompt_lecture6 lecture6_assignment3.py
   ```

Assignment 4 – pick up objects

1. Write your own python script to either
   - Pick up object on the conveyor next to robot 1
   - Add a new bin next to robot 1 and place the object(s) in the bin

   or
   - Pick up object on floor next to robot 2
   - Place object in bin 1 next to robot 2

NOTE!

- lecture6_assignment**1**.launch has rosservice **/spawn_object_once**, which spawns an object **on the conveyor** at robot 1
- lecture6_assignment**3**.launch has rosservice **/spawn_object_at_robot2**, which spawns an object on the floor next to robot 2

Assignement challenge!

- **Can you fill a bin with either robot 1 or 2?**
  - Use your learnings from lecture 5 and 6 to complete this assignment

Submit assignment to git repo

- Submit assignments to git repo and post on MS Teams