ELEG4701: Intelligent Interactive Robot Practice

Lecture 10: Introduction to Robot Arm

Jiewen Lai

Research Assistant Professor

jwlai@ee.cuhk.edu.hk

EE, CUHK

2024 Spring

Outline

Part 1. Basic Knowledge

Classification of Robots (Tangible)

Definition of 'Robot' from Oxford English Dictionary:
A robot is a machine – especially one programmable by a computer – capable of carrying out a complex series of actions automatically.

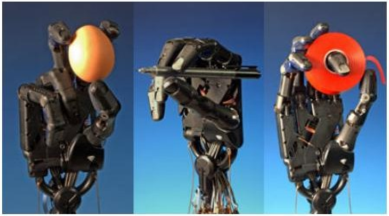
Different form refers to the way it looks:

- Robot arm
- Humanoid Robot
- Bio-inspired Robot
- Mobile Robot

Note: Chatbots (like GPT) / trading robots / etc., can be considered robots, but they are not tangible (usually no physical body).

Classification of Robots (Robot Arms)

Kuka LWR (KUKA Lightweight robot)



Shadow Robotics hand

Righthand robotics Reflex hand

Universal Robotics UR3

Classification of Robots (Humanoid Robots)

Figure: Robots with arms and legs that mimic the form of human beings

Classification of Robots (Bio-Inspired Robots)

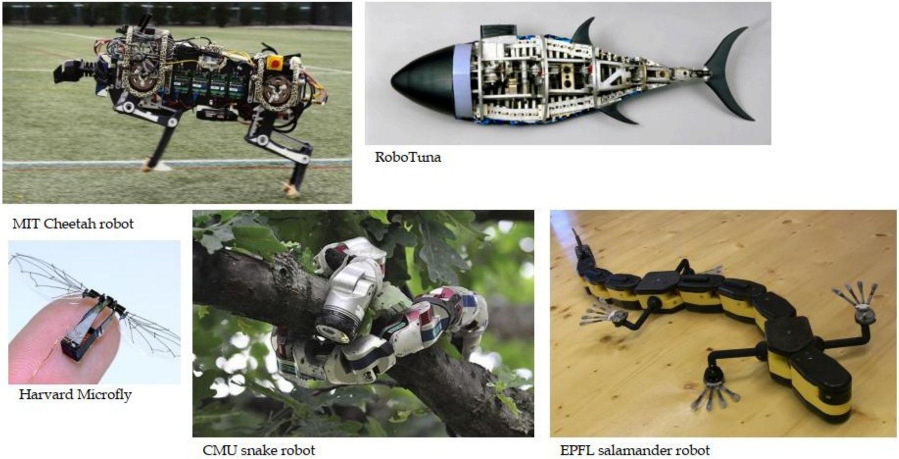


Figure: Robots that mimic the form of living creatures

Classification of Robots (Mobile Robots)

- Ability to move around freely

Aldebaran Robotics and Softbank mobile's Pepper service robot

iRobot PackBot

TU Eindhoven soccer robots

EPFL Alice swarm robot

Cambridge Unmanned Aerial Vehicle

Figure: Robots with the ability to move around freely

Introduction to Cobotta Robot

- **Manipulator**: 6 links & 6 joints
- **End-effector**: 1 gripper
- **Actuators**: AC motors
- **Sensors**: RealSense depth camera
- **Processor**: 1 Computer
- **Software**: OS, robotic software, and other applications

Part 2. Arm Control Demo in Sim

Arm Package Installation

Installation steps:

1. Install the following dependencies

Terminal (using TAB can be faster)

```
$ sudo apt install ros-noetic-moveit ros-noetic-controller-manager
ros-noetic-position-controllers ros-noetic-joint-state-controller
ros-noetic-joint-trajectory-controller
ros-noetic-joint-limits-interface ros-noetic-transmission-interface
ros-noetic-realtime-tools
```

2. Download the source code

Terminal

```
$ cd ~/catkin_ws/src
$ git clone -b noetic-devel https://github.com/rizgiak/denso
_cobotta_ros.git
```

Arm Package Installation

3. Build

Terminal

```
$ cd ~/catkin_ws
$ source /opt/ros/noetic/setup.bash
$ catkin_make
$ source ~/catkin_ws/devel/setup.bash
```

Simulation Demo

Cobotta simulation steps:

1. Install the dependent package (using TAB can be faster)

```
$ sudo apt install ros-noetic-gazebo-ros-control
```

2. Source your env

```
$ source devel/setup.bash
```

Note: If you find some errors suggesting that ROS cannot find your file, usually, you can try to source your environment or give the file executable permissions.

3. Run the sim

```
$ roslaunch denso_cobotta_bringup denso_cobotta_bringup.launch
sim:=true gripper_type:=none
```

After you run this simulation, you can see two windows about Rviz and Gazebo.

Arm Control Demo

You will find the following information:

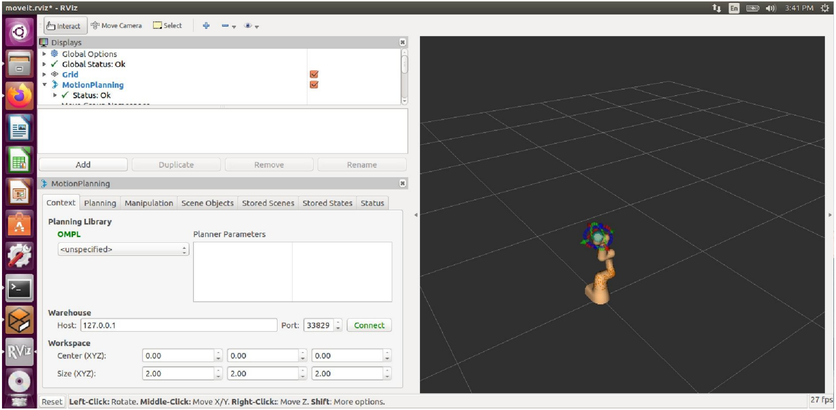


Figure: RViz is a visualization tool of ROS, used to visualize data and status information of robots and sensors.

You will find the following information:

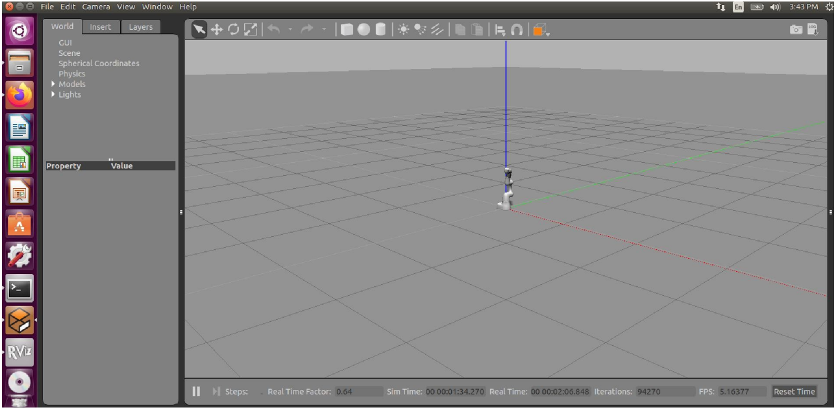


Figure: Gazebo is a powerful 3D physics simulation platform with a powerful physics engine. We can create our own environment and verify the robot's algorithm on it.

Arm Control Demo

Interactive GUI

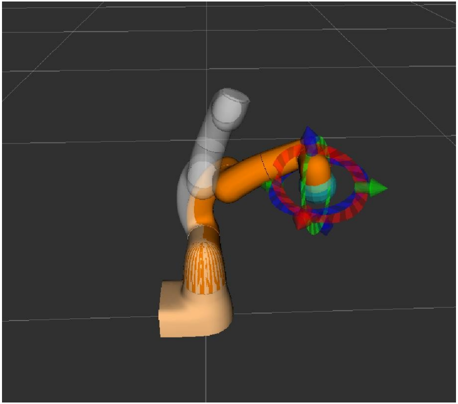


Figure: You can change the robot pose by dragging the end of the robot in RViz. Like this figure.

Arm Control Demo

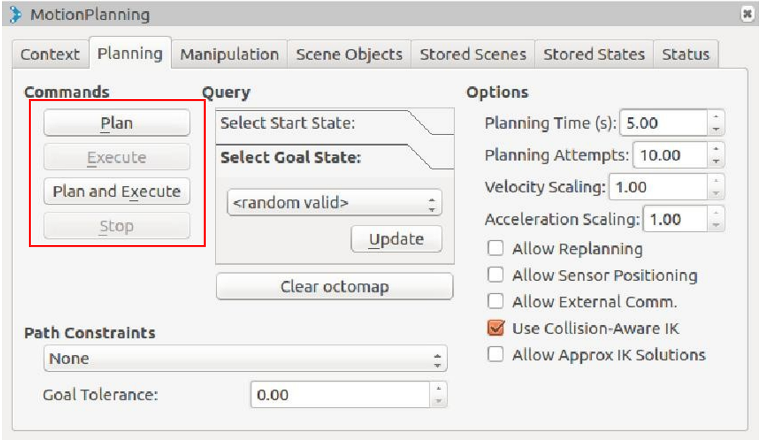


Figure: Then, click “Plan” “Execute” or “Plan and Execute” to plan a trajectory and control the robot to the goal in the “MotionPlanning” interface.

Arm Control Demo

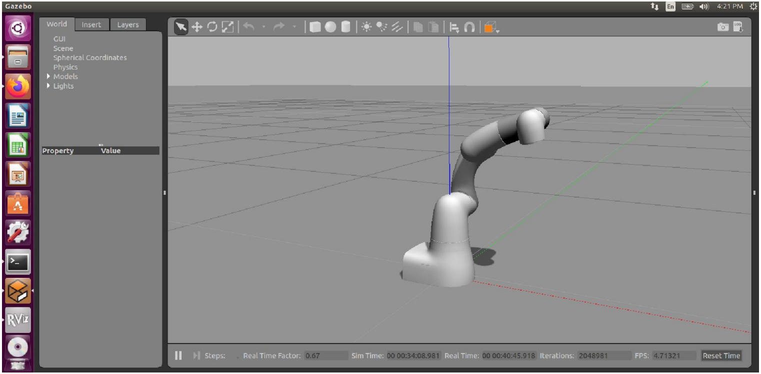


Figure: After execution, the simulation robot is controlled to reach the goal in gazebo.

Later in Task 1, you will need to reproduce this simulation and show it to the TAs.

Part 3. Introduction to URDF

URDF Concepts

What is an URDF?

- Abbrv. for **Unified Robot Description Format**
- A kinematic and basic physics description of a robot

How it works?

- XML format
- Tags: link, joint, transmission, ...
- Kinematic **tree structure**
- **Order** in the file **does not matter**

URDF Tags in XML

<link>

- The link element describes a rigid body with an inertial, visual feature, and collision properties.

<joint>

- The joint element describes the kinematics and dynamics of the joint and also specifies the safety limits of the joint.

<robot>

- The root element in a robot description file must be a robot, with all other elements must be encapsulated within.

Example for an URDF tag file

```
<robot name='your robot name'>
   <link> ... </link>
   <link> ... </link>
   <joint> ... </joint>
   <joint> ... </joint>
</robot>
```

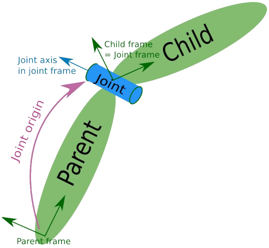
URDF Tags in XML

Example of defining a joint:

XML file (more can be found in http://wiki.ros.org/urdf/XML)

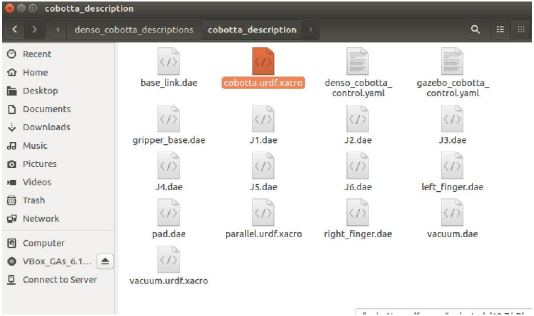
```
<joint name=‘my_joint’ type=‘floating’> #floating: This joint allows motion for all 6 DOFs
   <origin xyz=‘0 0 1’ rpy=‘0 0 3.1416’/> #defining a joint as in the figure
   <parent link=‘link1’/> #parent link
   <child link=‘link2’/> #child link

   <calibration rising=‘0.0’/>
   <dynamics damping=‘0.0’ friction=‘0.0’/>
   <limit effort=‘30’ velocity=‘1.0’ lower=‘-2.2’ upper=‘0.7’ />
   <safety_controller k_velocity=‘10’ k_position=‘15’ soft_lower_limit=‘-2.0’ soft_upper_limit=‘0.5’ />
</joint>
```



Check the URDF file in your pkg

You can find the 'cobotta.urdf.xacro' in the 'cobotta_description'



- 'xacro' is an improvised modeling file of URDF (XML Macros)[1]
- Similar to URDF, it also contains the model information of the robot

[1] http://wiki.ros.org/xacro

Check the URDF file in your pkg

After you look at the '`cobotta.urdf.xacro`', you know the info about the robot, and you can answer some questions below:

1. **How many joints** does this robot have, only including the joints of the robot itself and not the joints in the environment?
2. What is the **name of the base of the robot**? Who fixed the base to the world environment (joint name)?
3. In this model, what is **the name of the link** at the end of the robot?
4. Since the model of the robot is known, if we set the angle of each joint, can the pose of the robot end be calculated? If we set the pose of the end of the robot, can the angle of each joint be calculated? (Yes/No, explain why briefly?)

Please put down your answer on the Lab sheet (Task 2) later.

Part 4. Kinematics

Kinematics in a ROBOT ARM

Forward Kinematics (Joint Space –> Cartesian Space)

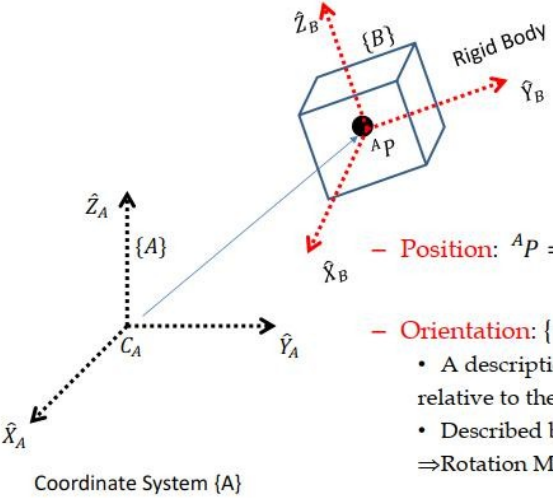
- **Defining:** Length of each Link (L1,L2,L3,L4,L5) & Angle of each joint (J1,J2,J3,J4,J5,J6)
- **Solving:** Position of any point along the robot (x,y,z)

Inverse Kinematics (Joint Space <– Cartesian Space)

- **Defining:** Position of any point along the robot (x,y,z)
- **Solving:** Length of each Link (L1,L2,L3,L4,L5) & Angle of each joint (J1,J2,J3,J4,J5,J6)

- In robotics, we often concerned about the **location of objects in three-dimensional space**. These objects can be the links of the manipulator, parts and tools, and other objects in the environment.
- **Position** and **orientation** are two main attributes to describe the location and configuration of these objects

Spatial Descriptions and Transformations



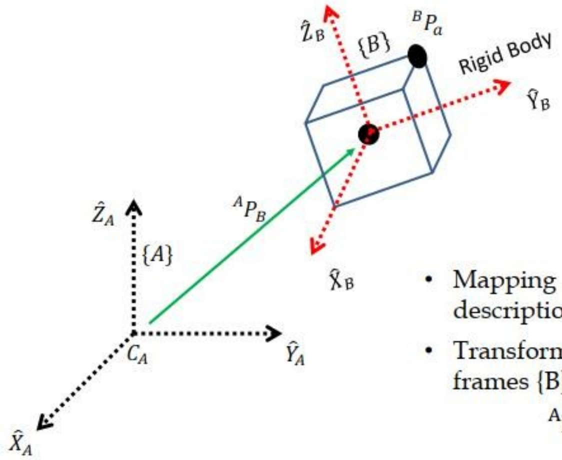
– Position: $^{A}P = \begin{bmatrix} {}^{A}P_x \\ {}^{A}P_y \\ {}^{A}P_z \end{bmatrix}$

– Orientation: $\{ {}^{A}\hat{X}_B, {}^{A}\hat{Y}_B, {}^{A}\hat{Z}_B \}$

- A description of the body coordinate system relative to the reference system
- Described by 3 principle axes

⇒Rotation Matrix

Spatial Descriptions and Transformations



- Mapping – a process to change descriptions from frame to frame
- Transform a vector (or point) $^{B}P_a$ from frames {B} to {A}

$$^{A}P_a = {}^{A}R_B \, {}^{B}P_a + {}^{A}P_B$$

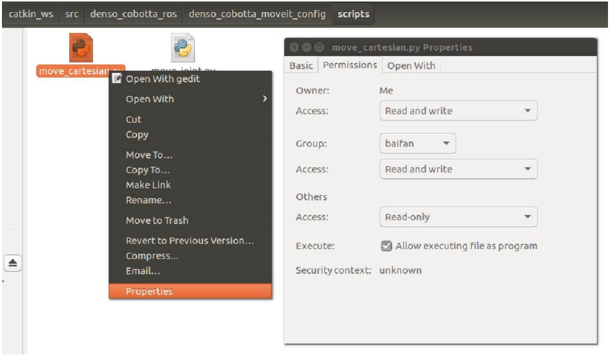
MoveIt!

MoveIt! is an easy-to-use **robotics manipulation platform** for developing applications in:

- Motion Planning
- 3D perception
- Navigation
- ...

Prepare

1. Download 'scripts.tar.gz' from the Blackboard
2. Extract it and put the uncompressed 'scripts' file in 'denso_cobotta_moveit_config'
3. As always, give the .py files permissions by `$ chmod +x node.py`, or



Forward Kinematics

1. Run simulation

```
$ roslaunch denso_cobotta_bringup denso_cobotta_bringup.launch
sim:=true gripper_type:=none
```

2. Run move_group in a new terminal

```
$ roslaunch denso_cobotta_moveit_config move_group.launch
```

3. Run move_joint in a new terminal

```
$ rosrun denso_cobotta_moveit_config move_joint.py
```

In this part, we set 6 joint values to control the robot go to the target. You can also change the value of the joints. Then, the robot will arrive at the joint value that you want.

Inverse Kinematics

1. Run simulation (Skip if it is already running)

```
$ roslaunch denso_cobotta_bringup denso_cobotta_bringup.launch
sim:=true gripper_type:=none
```

2. Run move_group in a new terminal (Skip if it is already running)

```
$ roslaunch denso_cobotta_moveit_config move_group.launch
```

3. Run move_joint in a new terminal

```
$ rosrun denso_cobotta_moveit_config move_cartesian.py
```

In this part, we set 3 positions and 4 orientations. Then, the robot plans the trajectory and executes it to the target. You can also change the positions and orientations. Then, the robot will arrive at the pose that you want.

- Please run move_joint.py and move_cartesian.py successfully and follow the instructions in Task 4 to modify the target value.
- After you read these code examples, you should have a preliminary understanding of how to use moveit to control the robot.
- You should understand the two control modes based on Joint space and Cartesian space.
- In the next class, we will learn in detail how to write the moveit code to control the robot.

Thanks for listening! Please finish your lab sheet!