Project 4 FAQ

- **How do run this project in my own Ubuntu machine?**

1. Launch Project 4, then in Vocareum click Actions>Download Starter code. This will download all the files you need to make the project run locally in your computer.
2. Install the needed ROS package(s). Run the following lines on your terminal:
   ```
   sudo apt-get update
   sudo apt-get install ros-kinetic-urdfdom-py
   ```
   Replace kinetic with the ROS version that you are running on your local machine.
3. **IGNORE** all the files other than catkin_ws, lwr_defs folders, and kuka_lwr_arm.urdf. Put the catkin_ws and the kuka_lwr_arm.urdf file in your home directory. Now grab the lwr_defs folder and move it inside your catkin_ws/src folder with the rest of the packages.
4. The downloaded files are structured as a catkin workspace. You can either use this structure directly (as downloaded) and build the workspace using the "catkin_make" command or use whatever catkin workspace you already had, and just copy the packages inside your own src folder and run the catkin_make command. If you are having troubles with this, you should review the first ROS tutorial "Installing and configuring your ROS Environment".
5. Once you have a catkin workspace with the packages inside the src folder, you are ready to work on your project without having to make any changes in any of the files. Navigate to the catkin workspace folder and build the workspace using the command "catkin_make".
6. NOTE: You can source both your ROS distribution and your catkin workspace automatically everytime you open up a terminal automatically by editing the ~/.bashrc file in your home directory. For example if your ROS distribution is Kinetic, and your catkin workspace is called "project4_ws" (and is located in your home directory) then you can add the following at the end of your .bashrc file:
   ```
   source /opt/ros/kinetic/setup.bash
   echo "ROS Kinetic was sourced"
   source ~/project4_ws/devel/setup.bash
   echo "project4_ws workspace was sourced"
   ```
   This way every time you open up a terminal, you will already have your workspace sourced, such that ROS will have knowledge of the packages there.
7. **Before moving forward, if you haven't followed the instructions on step 6, you will need to source ROS and the catkin workspace every time you open a new terminal.** To run the project, first open up a terminal and type "`roscore`". In the second terminal (remember to source ROS and the catkin workspace if you didn't do step 6) make sure you are in your home directory and run "rosparam set robot_description --textfile kuka_lwr_arm.urdf", followed by "`rosrun robot_sim robot_sim_bringup`".
8. On another 2 separate terminals you need to run the scripts for the robot state publisher and interactive markers: "`rosrun robot_state_publisher robot_state_publisher`" and "`rosrun cartesian_control marker_control.py`". Note that you can find these lines from setup_project4.sh in the starter code. Finally you can run your own script in another terminal: "rosrun cartesian_control cartesian_control.py"
   NOTE: you can safely ignore the messages "multiple times: collisionScalar element defined multiple times" on the console.
9. Now we can open up Rviz using "rosrun rviz rviz". Inside Rviz, first change the Fixed Frame to "world_link". Then click Add and select RobotModel from the list of options. At this point you should see the robot arm standing straight up. To add the interactive marker needed to command the robot around, click Add and select "InteractiveMarkers" from the list. Then on the left navigation plane, expand the InteractiveMarkers object, and click on Update Topic > /control_markers/update.
   Now you should see the robot in Rviz, along with an interactive marker to command different positions for the end effect. Once your code works, the robot will follow whatever command is issued by moving this marker around.

- **How to compute delta_X with helper function for the rotation part.**

Remember you need to do this in the End Effector frame. Hence to compute **delta_X**:

Using the knowledge of the transformations **b_T_ee_current** and **b_T_ee_desired** you can compute the transform that goes from ee_current to ee_desired. This transformation contains the translation part and rotation part that you need in the proper frame.

The translation part is right there in the transformation matrix, no need for further processing, just extract it.

However, the rotation part is a rotation matrix, which you will have to work to transform into a 3 number delta vector to represent the change in pose: **delta_w** = **[a,b,c]^t** and be able to build your **V_ee** vector. This is where the helper function comes in and gives you the rotation in the angle-axis representation (which is basically an angular velocity vector, i.e, what you were looking for).

At this point you have your delta translational and rotational. We recommend to work the translation and rotation parts independently:

You compute your EE translational delta, multiply by a gain to obtain your EE translational velocity, and then scale the whole translational vector just in case its norm is too big (that way you cap the maximum velocity). You do the same for rotation,