This file introduces a way to install KDL(Orocos Kinematics and Dynamics Library), and shows a demo based on KDL.

In general, we need download KDL source code and compile it, then, install it in our system. After doing these steps, we can use it just like using rclcpp or geometry_msgs provided by ROS 2. The detailed steps are as follows.

Step 1: Install dependencies (when compiling KDL, we need these tools and libraries, such as **cmake**, **git**, **build-essential** and **libeigen3-dev**)

```
sudo apt update
sudo apt install cmake git build-essential libeigen3-dev
```

Step2: Git clone KDL source code

```
git clone https://github.com/orocos/orocos_kinematics_dynamics.git
```

After doing this, in the current directory, we can get a folder named by "orocos_kinematics_dynamics", in which KDL source code stores.

Step3: Compile KDL source code

```
cd orocos_kinematics_dynamics/orocos_kdl
mkdir build
cd build
cmake ..
make
```

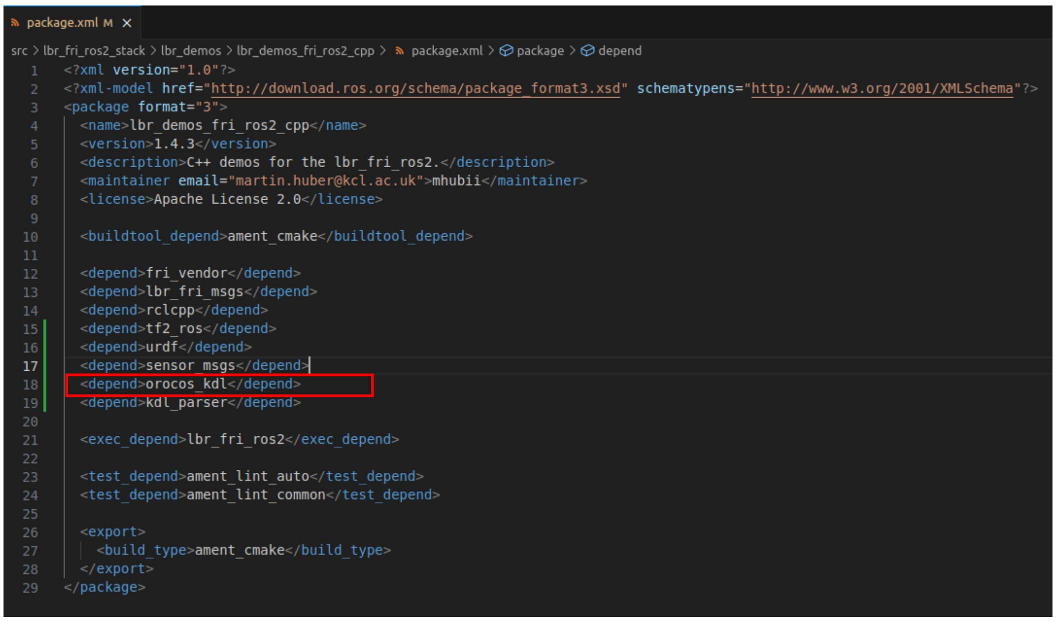
Step4: Install the compiled code in your system

```
sudo make install
```

After doing this, the compiled code would be in your system and the default installation path is '/usr/local/'.

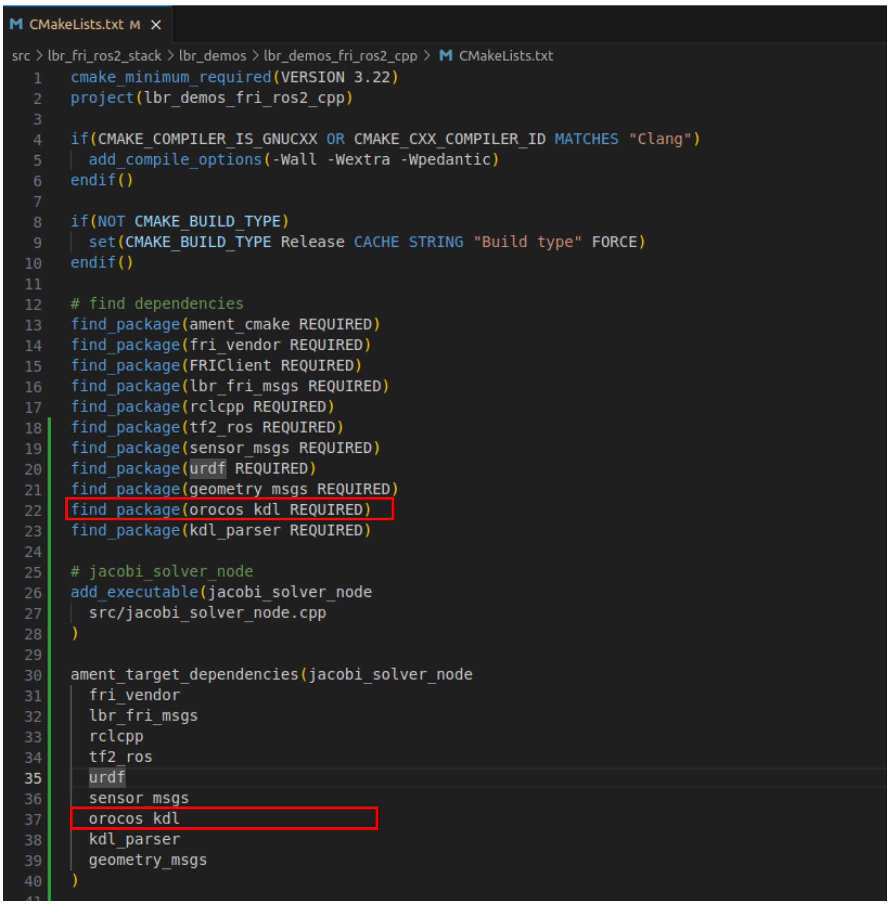
Tips: The above steps have been tested in ubuntu 22.04. If any problems in your installation process, please ask ChatGPT or find solutions on the Internet.

Finally, we can use KDL in our code. We need add some configurations in package.xml(please refer to Fig. 1) and CMakeLists.txt(Fig. 2).


package.xml M ×
src > lbr_fri_ros2_stack > lbr_demos > lbr_demos_fri_ros2_cpp > package.xml > package > depend

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>lbr_demos_fri_ros2_cpp</name>
  <version>1.4.3</version>
  <description>C++ demos for the lbr_fri_ros2.</description>
  <maintainer email="martin.huber@kcl.ac.uk">mhubii</maintainer>
  <license>Apache License 2.0</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>fri_vendor</depend>
  <depend>lbr_fri_msgs</depend>
  <depend>rclcpp</depend>
  <depend>tf2_ros</depend>
  <depend>urdf</depend>
  <depend>sensor_msgs</depend>
  <depend>orocos_kdl</depend>
  <depend>kdl_parser</depend>

  <exec_depend>lbr_fri_ros2</exec_depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

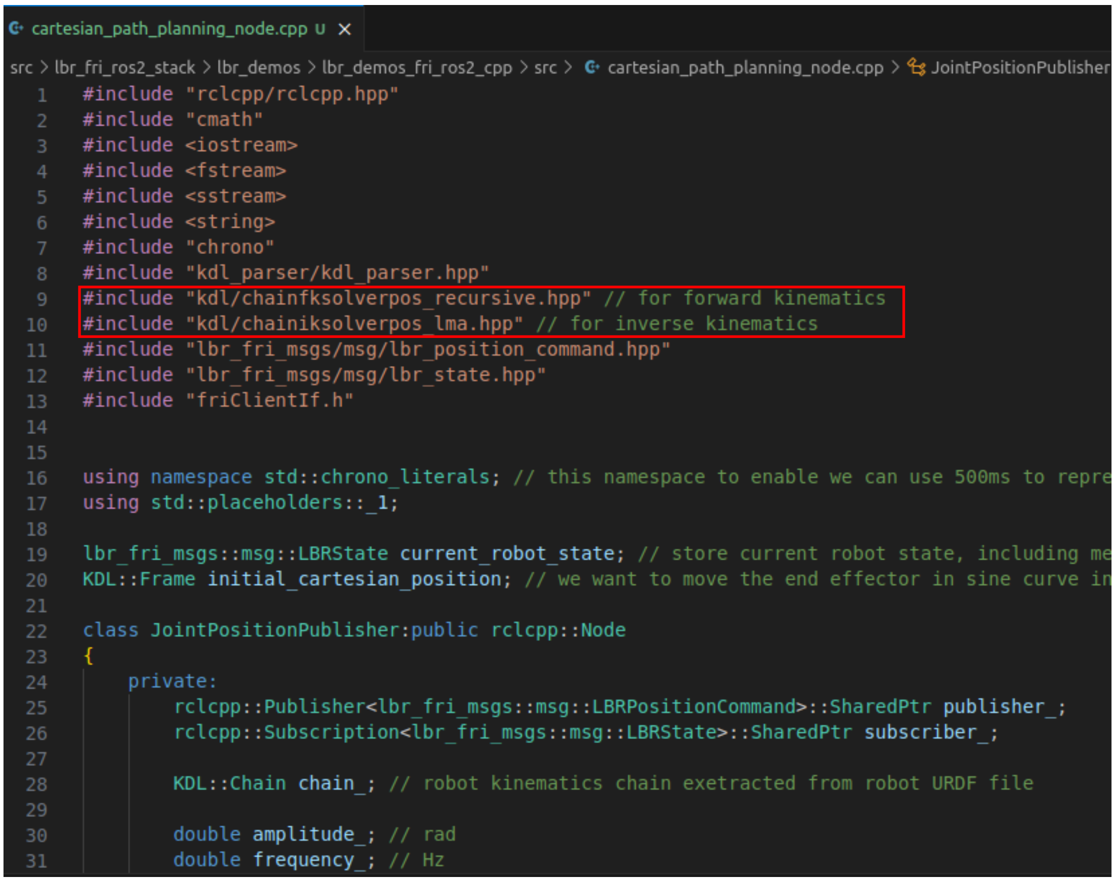
Fig. 1 package.xml


CMakeLists.txt M ×
src > lbr_fri_ros2_stack > lbr_demos > lbr_demos_fri_ros2_cpp > CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 3.22)
project(lbr_demos_fri_ros2_cpp)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

if(NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE Release CACHE STRING "Build type" FORCE)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(fri_vendor REQUIRED)
find_package(FRIClient REQUIRED)
find_package(lbr_fri_msgs REQUIRED)
find_package(rclcpp REQUIRED)
find_package(tf2_ros REQUIRED)
find_package(sensor_msgs REQUIRED)
find_package(urdf REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(orocos_kdl REQUIRED)
find_package(kdl_parser REQUIRED)

# jacobi_solver_node
add_executable(jacobi_solver_node
  src/jacobi_solver_node.cpp
)

ament_target_dependencies(jacobi_solver_node
  fri_vendor
  lbr_fri_msgs
  rclcpp
  tf2_ros
  urdf
  sensor_msgs
  orocos_kdl
  kdl_parser
  geometry_msgs
)
```

Fig. 2 CMakeLists.txt

In .cpp file, we can include KDL header file as shown in Fig. 3


cartesian_path_planning_node.cpp U ×
src > lbr_fri_ros2_stack > lbr_demos > lbr_demos_fri_ros2_cpp > src > cartesian_path_planning_node.cpp > JointPositionPublisher

```
#include "rclcpp/rclcpp.hpp"
#include "cmath"
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include "chrono"
#include "kdl_parser/kdl_parser.hpp"
#include "kdl/chainfksolverpos_recursive.hpp" // for forward kinematics
#include "kdl/chainiksolverpos_lma.hpp" // for inverse kinematics
#include "lbr_fri_msgs/msg/lbr_position_command.hpp"
#include "lbr_fri_msgs/msg/lbr_state.hpp"
#include "friClientIf.h"


using namespace std::chrono_literals; // this namespace to enable we can use 500ms to repre
using std::placeholders::_1;

lbr_fri_msgs::msg::LBRState current_robot_state; // store current robot state, including me
KDL::Frame initial_cartesian_position; // we want to move the end effector in sine curve in

class JointPositionPublisher:public rclcpp::Node
{
    private:
        rclcpp::Publisher<lbr_fri_msgs::msg::LBRPositionCommand>::SharedPtr publisher_;
        rclcpp::Subscription<lbr_fri_msgs::msg::LBRState>::SharedPtr subscriber_;

        KDL::Chain chain_; // robot kinematics chain exetracted from robot URDF file

        double amplitude_; // rad
        double frequency_; // Hz
```

Fig. 3 include KDL in .cpp file

Here, we provide a demo, which uses KDL to calculate forward kinematics and inverse kinematics to make the robot's end effector run in z axis in Cartesian space. Please run the following commands in the terminal.

ros2 launch lbr_fri_ros2 app.launch.py model:=iiwa7

ros2 run lbr_demos_fri_ros2_cpp cartesian_pose_node --ros-args -p robot_name:=iiwa7

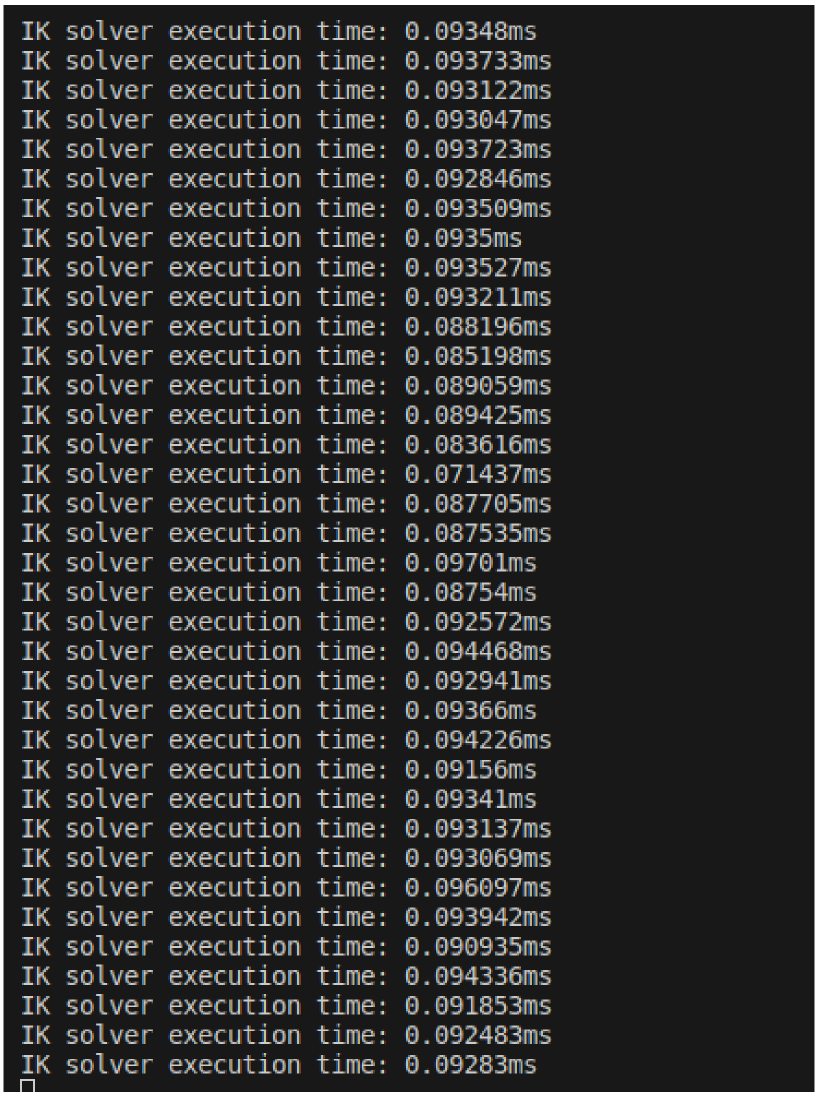
ros2 run lbr_demos_fri_ros2_cpp cartesian_path_planning_node

Tips: please modify 'iiwa7' to your robot name（such as iiwa14, med7, med14）

cartesian_pose_node is used to publish Cartesian Pose of the real robot and receive Cartesian Pose command from other ros nodes.

cartesian_path_planning_node is a demo, which is used to publish Cartesian Pose commands.

For our test result, KDL calculating inverse kinematics once needs less than **0.1ms**(please refer to Fig. 4, our OS is Ubuntu 22.04, and CPU is 13th Gen Intel(R) Core(TM) i9-13900K), which satisfies requirements of real-time control.



```
IK solver execution time: 0.09348ms
IK solver execution time: 0.093733ms
IK solver execution time: 0.093122ms
IK solver execution time: 0.093047ms
IK solver execution time: 0.093723ms
IK solver execution time: 0.092846ms
IK solver execution time: 0.093509ms
IK solver execution time: 0.0935ms
IK solver execution time: 0.093527ms
IK solver execution time: 0.093211ms
IK solver execution time: 0.088196ms
IK solver execution time: 0.085198ms
IK solver execution time: 0.089059ms
IK solver execution time: 0.089425ms
IK solver execution time: 0.083616ms
IK solver execution time: 0.071437ms
IK solver execution time: 0.087705ms
IK solver execution time: 0.087535ms
IK solver execution time: 0.09701ms
IK solver execution time: 0.08754ms
IK solver execution time: 0.092572ms
IK solver execution time: 0.094468ms
IK solver execution time: 0.092941ms
IK solver execution time: 0.09366ms
IK solver execution time: 0.094226ms
IK solver execution time: 0.09156ms
IK solver execution time: 0.09341ms
IK solver execution time: 0.093137ms
IK solver execution time: 0.093069ms
IK solver execution time: 0.096097ms
IK solver execution time: 0.093942ms
IK solver execution time: 0.090935ms
IK solver execution time: 0.094336ms
IK solver execution time: 0.091853ms
IK solver execution time: 0.092483ms
IK solver execution time: 0.09283ms
```

Fig. 4 KDL calculation time for inverse kinematics