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
```
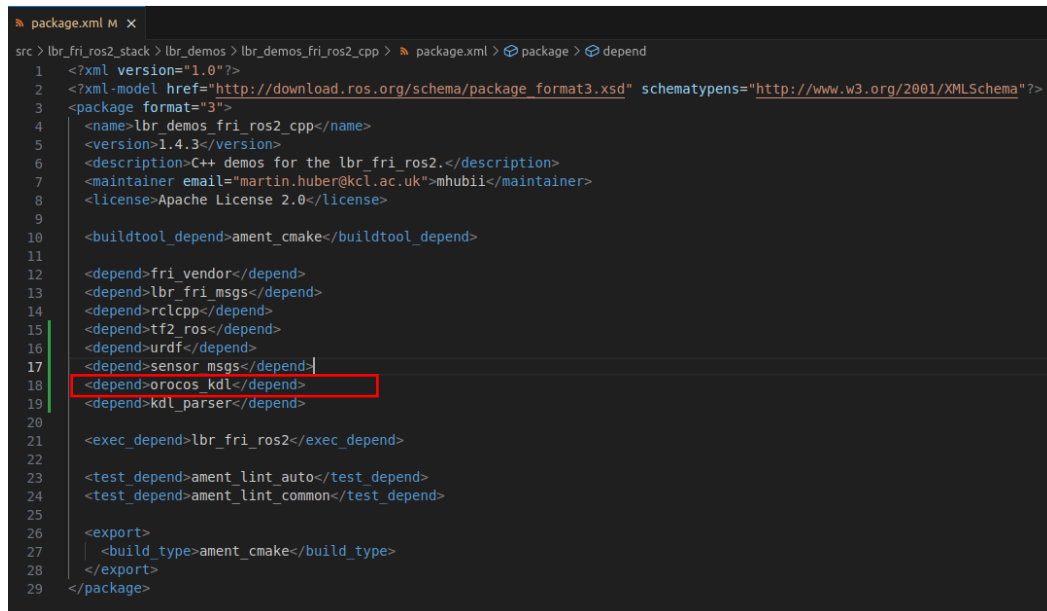
Step4: Install the compiled code in your system

```
sudo make install
```

After doing this, the compiled code would be in your system and the default installation path is '/usr/local/'.
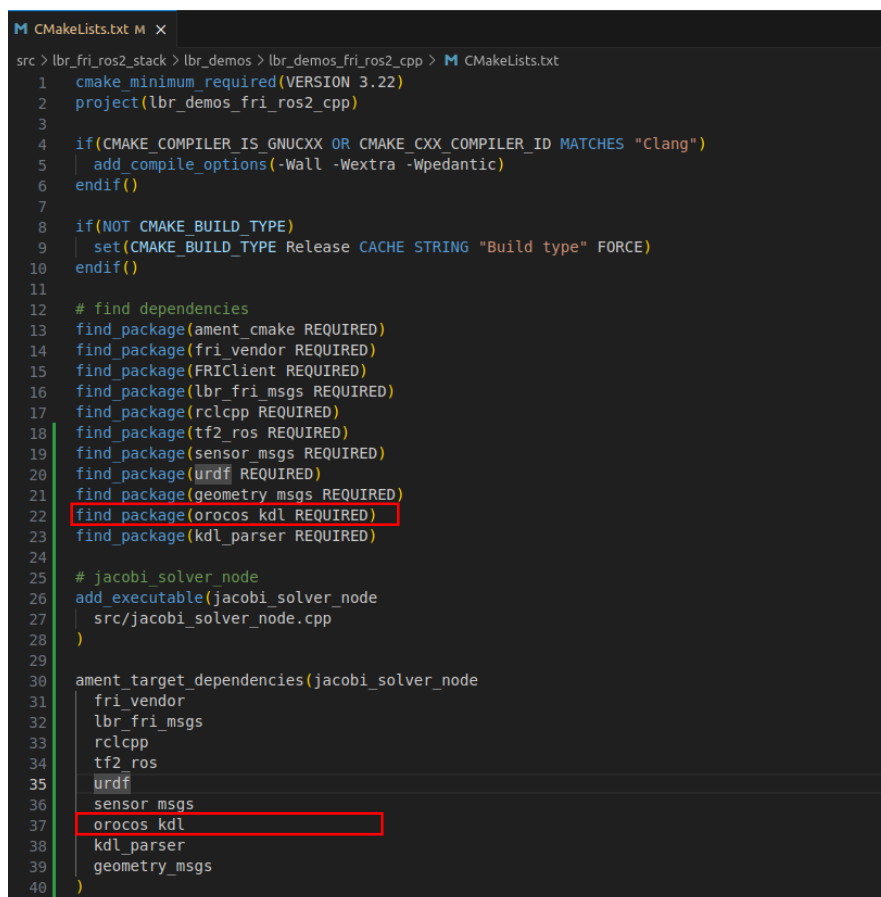
Tips: The above steps have been tested in ubuntu 22.04. If any problems in your installation process, please ask ChatGPT or find solutions on the Internet.

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).

```xml
src > lbr_fri_ros2_stack > lbr_demos > lbr_demos_fri_ros2_cpp > package.xml > package > depend
1   <?xml version="1.0"?>
2   <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
3   <package format="3">
4     <name>lbr_demos_fri_ros2_cpp</name>
5     <version>1.4.3</version>
6     <description>C++ demos for the lbr_fri_ros2.</description>
7     <maintainer email="martin.huber@kcl.ac.uk">mhubii</maintainer>
8     <license>Apache License 2.0</license>
9
10    <buildtool_depend>ament_cmake</buildtool_depend>
11
12    <depend>fri_vendor</depend>
13    <depend>lbr_fri_msgs</depend>
14    <depend>rclcpp</depend>
15    <depend>tf2_ros</depend>
16    <depend>urdf</depend>
17    <depend>sensor_msgs</depend>
18    <depend>orocos_kdl</depend>
19    <depend>kdl_parser</depend>
20
21    <exec_depend>lbr_fri_ros2</exec_depend>
22
23    <test_depend>ament_lint_auto</test_depend>
24    <test_depend>ament_lint_common</test_depend>
25
26    <export>
27      <build_type>ament_cmake</build_type>
28    </export>
29  </package>
```

Fig. 1 package.xml

```cmake
src > lbr_fri_ros2_stack > lbr_demos > lbr_demos_fri_ros2_cpp > CMakeLists.txt
1   cmake_minimum_required(VERSION 3.22)
2   project(lbr_demos_fri_ros2_cpp)
3
4   if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5     add_compile_options(-Wall -Wextra -Wpedantic)
6   endif()
7
8   if(NOT CMAKE_BUILD_TYPE)
9     set(CMAKE_BUILD_TYPE Release CACHE STRING "Build type" FORCE)
10  endif()
11
12  # find dependencies
13  find_package(ament_cmake REQUIRED)
14  find_package(fri_vendor REQUIRED)
15  find_package(FRIClient REQUIRED)
16  find_package(lbr_fri_msgs REQUIRED)
17  find_package(rclcpp REQUIRED)
18  find_package(tf2_ros REQUIRED)
19  find_package(sensor_msgs REQUIRED)
20  find_package(urdf REQUIRED)
21  find_package(geometry_msgs REQUIRED)
22  find_package(orocos_kdl REQUIRED)
23  find_package(kdl_parser REQUIRED)
24
25  # jacobi_solver_node
26  add_executable(jacobi_solver_node
27    src/jacobi_solver_node.cpp
28  )
29
30  ament_target_dependencies(jacobi_solver_node
31    fri_vendor
32    lbr_fri_msgs
33    rclcpp
34    tf2_ros
35    urdf
36    sensor_msgs
37    orocos_kdl
38    kdl_parser
39    geometry_msgs
40  )
```

Fig. 2 CMakeLists.txt

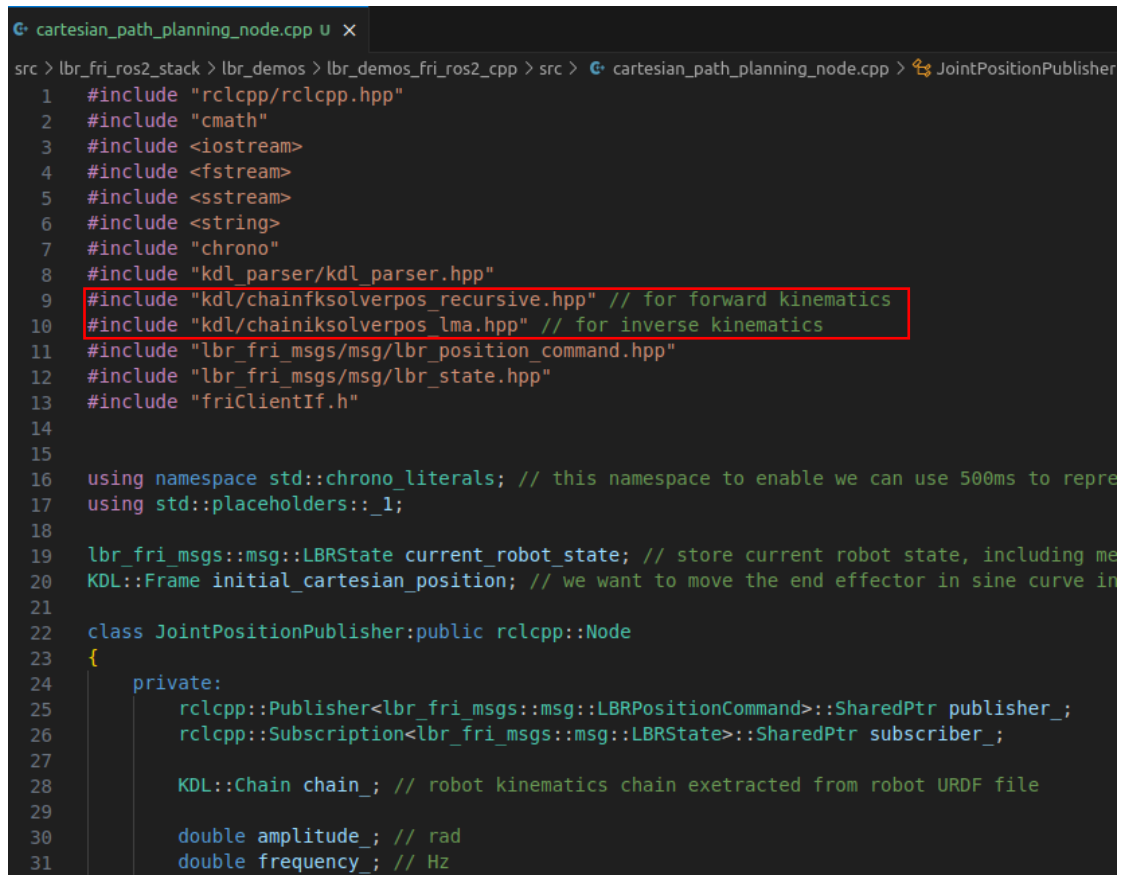In .cpp file, we can include KDL header file as shown in Fig. 3

```cpp
1   #include "rclcpp/rclcpp.hpp"
2   #include "cmath"
3   #include <iostream>
4   #include <fstream>
5   #include <sstream>
6   #include <string>
7   #include "chrono"
8   #include "kdl_parser/kdl_parser.hpp"
9   #include "kdl/chainfksolverpos_recursive.hpp" // for forward kinematics
10  #include "kdl/chainiksolverpos_lma.hpp" // for inverse kinematics
11  #include "lbr_fri_msgs/msg/lbr_position_command.hpp"
12  #include "lbr_fri_msgs/msg/lbr_state.hpp"
13  #include "friClientIf.h"
14
15
16  using namespace std::chrono_literals; // this namespace to enable we can use 500ms to repre
17  using std::placeholders::_1;
18
19  lbr_fri_msgs::msg::LBRState current_robot_state; // store current robot state, including me
20  KDL::Frame initial_cartesian_position; // we want to move the end effector in sine curve in
21
22  class JointPositionPublisher:public rclcpp::Node
23  {
24      private:
25          rclcpp::Publisher<lbr_fri_msgs::msg::LBRPositionCommand>::SharedPtr publisher_;
26          rclcpp::Subscription<lbr_fri_msgs::msg::LBRState>::SharedPtr subscriber_;
27
28          KDL::Chain chain_; // robot kinematics chain exetracted from robot URDF file
29
30          double amplitude_; // rad
31          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_path_planning_node

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.

Fig. 4 KDL calculation time for inverse kinematics