# 5th Exercise
# Robot Operating System Essentials – Summer School
## Inverse kinematics server

IGMR Institute of Mechanism Theory, Machine Dynamics and Robotics | RWTH AACHEN UNIVERSITY

# Inverse kinematics as a service

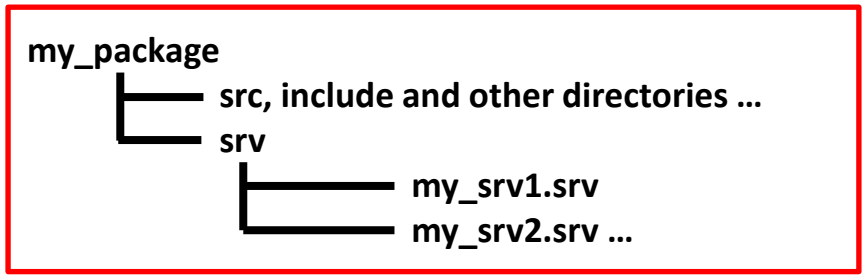
With the visualization of the motor angles out of the way, we can now reimplement the inverse kinematics of the delta robot in ROS again and consequently visualize the complete kinematics of the delta robot

For this, we will create a **ROS2 server** dedicated towards calculating the inverse kinematics
The server will accept a **request** for an effector position, and send the inverse kinematics **response** to the Arduino
(While working with the pseudo_arduino, this fake hardware will instantly jump to the desired angles as you will see in rviz)

In the following, we will add the .srv files, the server and the inverse_kinematics calculations

# Creating the service

ROS services are defined as **.srv** files inside a directory named **srv**, which we put in the **top level of the ROS package**
The resulting package structure is then as follows




Now, create the directory and create a file inside named **Ikin.srv**

This file defines the structure of the service, aka. the type of request and the type of response, which in our case is
Request structure:

The request consists of 3 real numbers (**float64** in ROS) for the end effector position
The numbers are not in an array but instead in individual variables called **x**, **y** and **z**

Response structure:

The response consists of 3 real numbers (**float64** in ROS) for the joint angles (in degrees)
The numbers are not in an array but instead in individual variables called **phi_11**, **phi_12** and **phi_13**

# Creating the service

**The request and response fields are separated by three dashes ---**

Here is an example for a service

common_interfaces / nav_msgs / srv / **LoadMap.srv**

Name of the service

christophebedard Fix typo in nav_msgs/LoadMap (#246)

Code | Blame | 15 lines (14 loc) · 474 Bytes

```
1   # URL of map resource
2   # Can be an absolute path to a file: file:///path/to/maps/floor1.yaml
3   # Or, relative to a ROS package: package://my_ros_package/maps/floor2.yaml
4   string map_url
5   ---
6   # Result code definitions
7   uint8 RESULT_SUCCESS=0
8   uint8 RESULT_MAP_DOES_NOT_EXIST=1
9   uint8 RESULT_INVALID_MAP_DATA=2
10  uint8 RESULT_INVALID_MAP_METADATA=3
11  uint8 RESULT_UNDEFINED_FAILURE=255
12
13  # Returned map is only valid if result equals RESULT_SUCCESS
14  nav_msgs/OccupancyGrid map
15  uint8 result
```

Request contains one string

Request of the service

Delimiters between request and response

Response of the service

Response contains service result and maybe an OccupancyGrid

# Inverse kinematics server

Now, we shift our attention the inverse kinematics server **ikin_server.cpp**

As this server will handle services of type **Ikin.srv**, it needs to know about the definition of this service
The point of the srv directory convention was such that ROS can **autogenerate** the necessary C++ files (headers) for them
All we have to do for autogeneration of messages/services/actions is modify the CMakeLists.txt accordingly (later)

**However, it is very important you are aware of the naming conventions of custom msgs/srvs/actions**
**If you do not follow these conventions, you will run into errors when compiling the code**

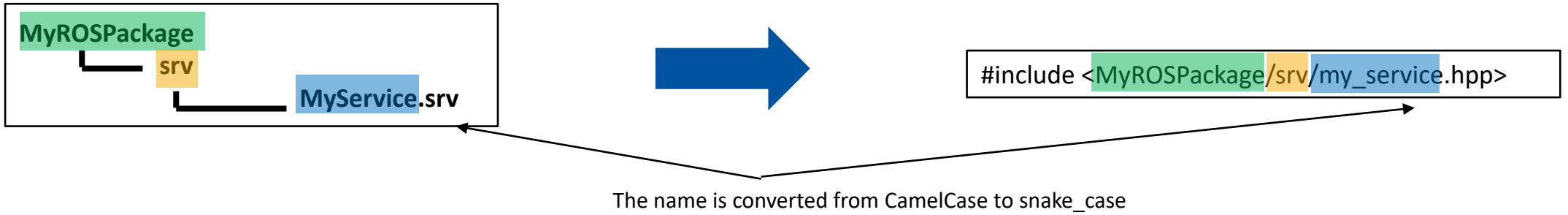
**Custom messages, services and actions must be in upper camel case and contain alphanumeric values only**

| This: | Not This: |
|---|---|
| MyMessage | my_message |
| Position | position |
| CoolAction | Cool_Action |

This convention is used for ensuring the C++ headers and the class names are autogenerated correctly

# Inverse kinematics server

The convention for services is as follows




In our case, since our service is called Ikin.srv, we import the headers as such

```
#include <delta_robot_serial/srv/Ikin.hpp>
```

This header includes the an autogenerated **C++ class with the same name as the .srv file,**

Which in the example becomes `MyROSPackage::srv::MyService`

In our case, this means our class is of type **delta_robot_serial::srv::Ikin**

# Inverse kinematics server

We follow the lazy way of creating the server, i.e. we do not explicitly write a class (in contrast to delta_joint_pub)
For this, start with the main function and add a server to the ikin_server node with the identifier **/ikin**

For the callback function, we need to first make sure we get the request and response object type right
Based on the autogeneration convention, the function arguments (request and reponse part of the service) are in general

```
callback(const MyROSPackage::srv::MyService::Request  &req, const MyROSPackage::srv::MyService::Response &res);
```

Which in our case becomes

```
callback(const delta_robot_serial::srv::Ikin::Request  &req, const delta_robot_serial::srv::Ikin::Response &res);
```

In the body, add the code to handle the request to
Calculate the inverse kinematics using **inverse_kinematics(…)** from the inverse_kinematics.h file
Use the (given) **sendToSerial(…)** function to send it to the Arduino
Populate the response object with the commanded motor values

**Important: We want to only send actual numbers to the Arduino! If the IK leads to any NaN values, we ignore the request!**

Where does the inverse_kinematics function come from?
You have to program it yourself next! (But you already did it for Delta_IK.ino, so this should not be a big problem…)

# CMakeLists.txt

**CMakeLists.txt modifications:**

Since we have specified a custom service definition, we must (auto)generate its C++ headers which we referenced in the node
This we do using the rosidl_default_generators package (hence a dependency)

We then add the following after the **find_package()** block

```
rosidl_generate_interfaces(${PROJECT_NAME} "srv/Ikin.srv" DEPENDENCIES std_msgs LIBRARY_NAME ${PROJECT_NAME})
```

As you can see, our service depends on **std_msgs** (for the floats), so you should also add this to the dependencies

Add the executable for the node, however for the ikin_server we need additional commands before install(…)
Because we are specifying the services in the same package that we also have the node in, we need the following
(official ROS2 practice is to put message/service/action definitions in a standalone package)

```
rosidl_get_typesupport_target(cpp_typesupport_target ${PROJECT_NAME} rosidl_typesupport_cpp)
target_link_libraries(ikin_server "${cpp_typesupport_target}")
```

# package.xml

**package.xml modifications:**

In order to use rosidl during the build process, we need the following dependencies
A buildtool depency on **rosidl_default_generators**
An execution depency on **rosidl_default_runtime**
And a special dependency because of not having a standalone service package

```
<member_of_group>rosidl_interface_packages</member_of_group>
```

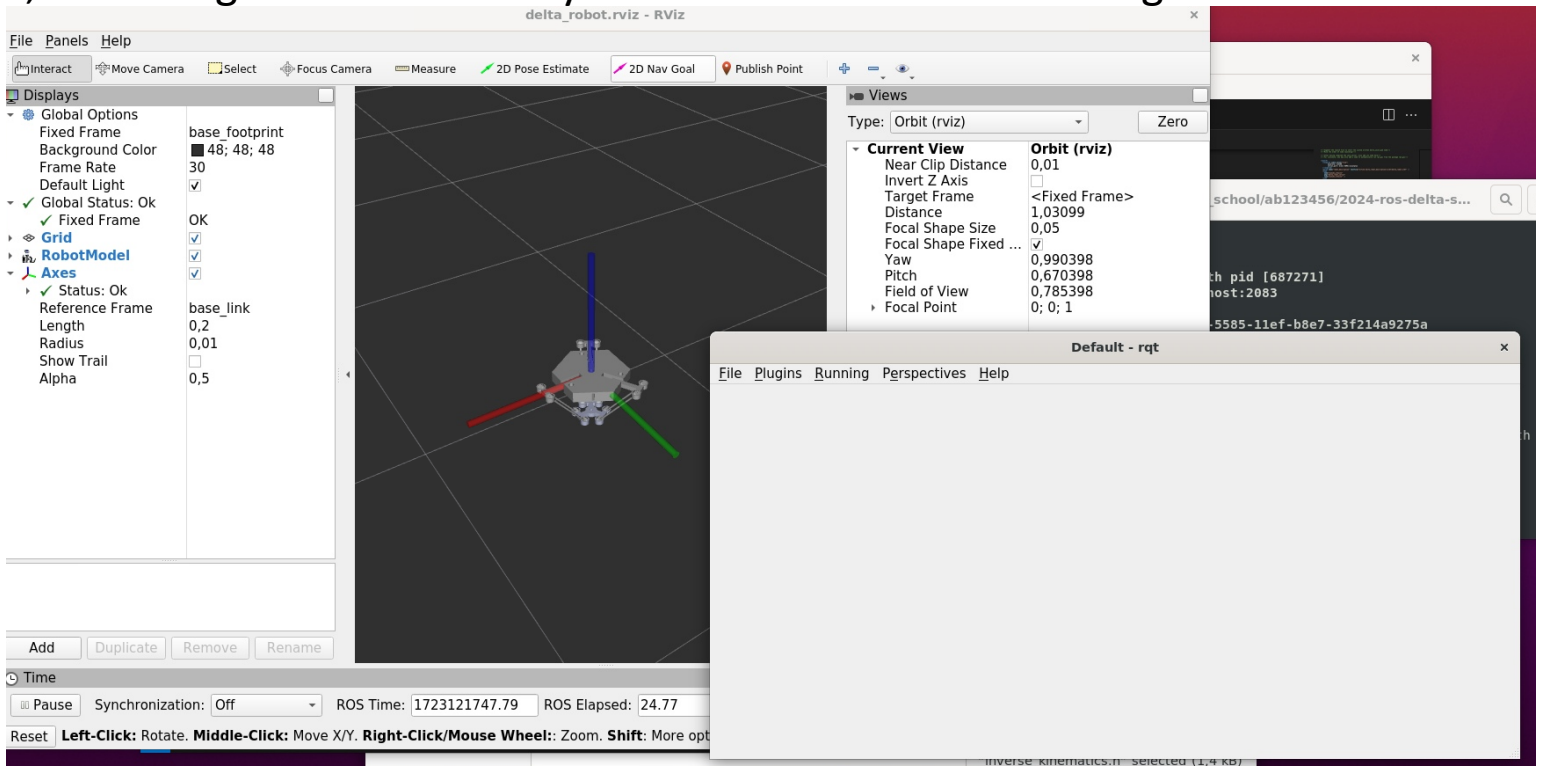
Also, because we depend on **std_msgs** in the CMakeLists.txt, we should add it here

# PseudoArduino Launch modifications

To test our server, we will use RQT. Let us immediately modify our PseudoArduino launch file instead of using ros2 run

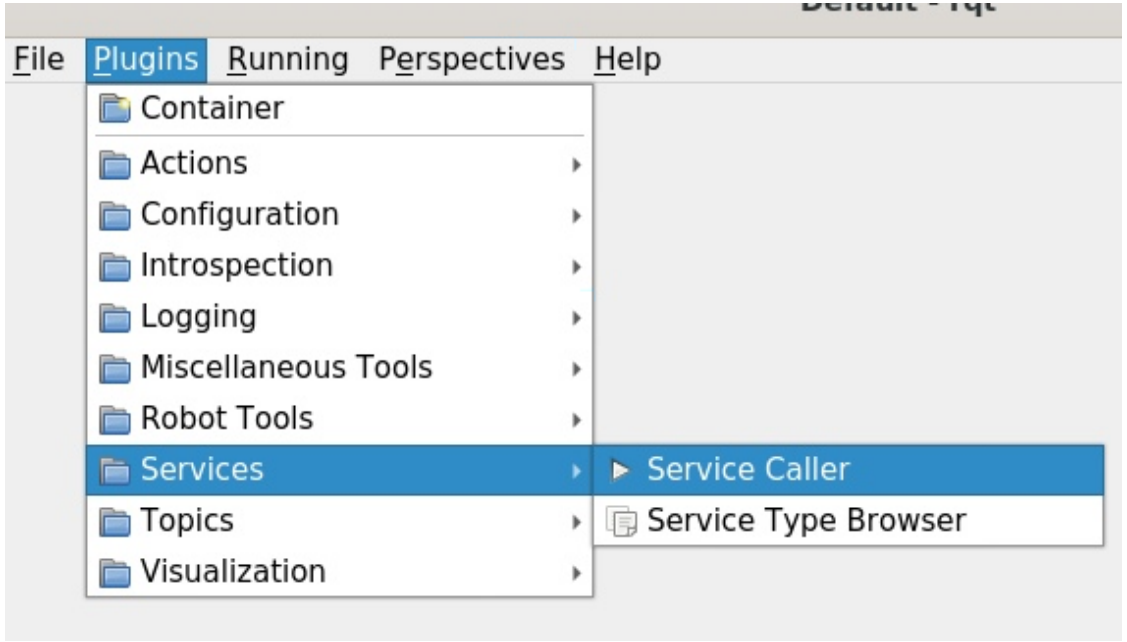
After starting the **delta_joint_pub**, add the launching of the **ikin_server** node
Also, after starting rviz2, also launch a node of type **rqt_gui** from the ROS package **rqt_gui** and name it **rqt_gui**

If everything is correct, launching PseudoArduino you should now see something like this

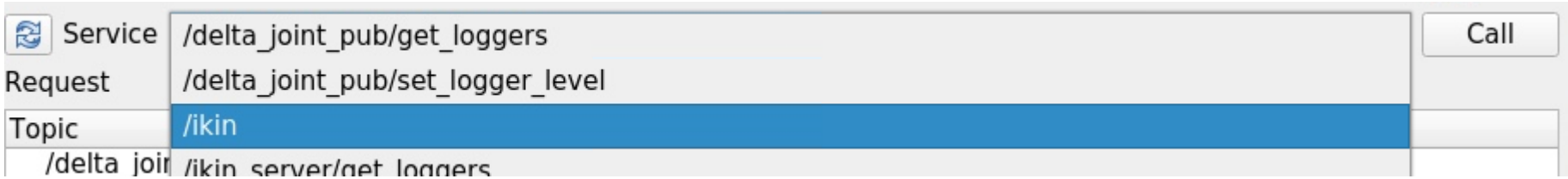


# RQT

We use rqt to directly interface with the inverse kinematics server
For this, add a service caller plugin panel to rqt

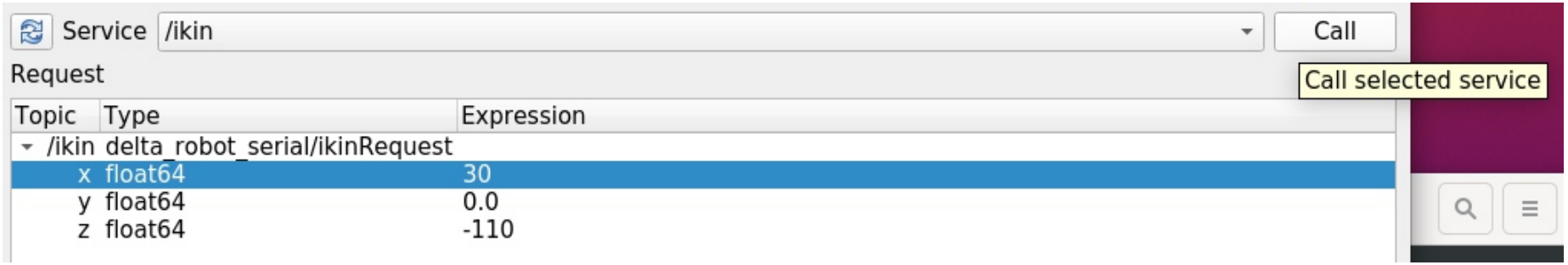



If the server is working as intended, it should advertise a service called ikin in the services list

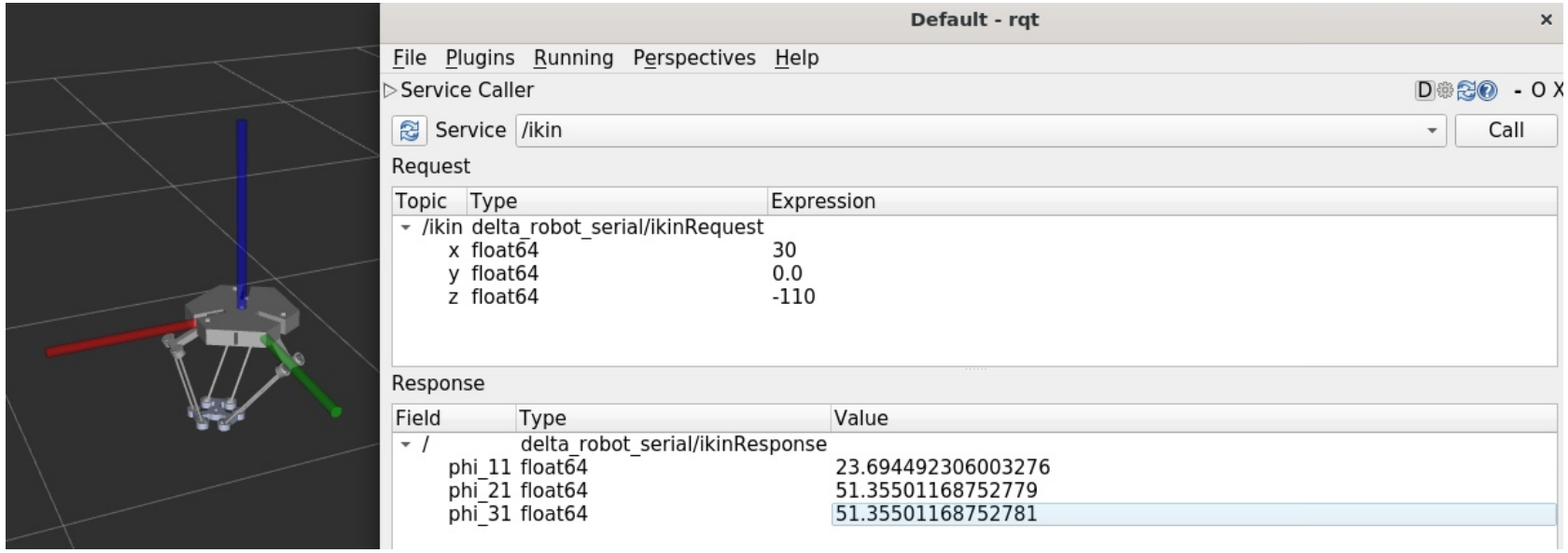


# RQT

RQT is programmed such that it automatically infers the request field from the service definition
Try putting the following values and then hit **Call**




You should see a) the robot jump to the desired position in rviz and b) the rqt service updating to the following



# Thank you
# for your kind attention

Contact:

Institute of Mechanism Theory, Machine Dynamics and Robotics
RWTH Aachen University
Eilfschornsteinstraße 18
52072 Aachen

(+49)-241 80-95546
intac-rosdelta@igmr.rwth-aachen.de

www.igmr.rwth-aachen.de