F&P Personal Robotics

ROS Interface for P-Rob

Version 1.0.1

2015-09-25

The information contained in this document is property of F&P Robotics AG and shall not be reproduced in whole or in part without prior written approval of F&P Robotics AG. The information provided herein is subject to changes without notice and should not be constructed as a commitment by F&P Robotics AG. This manual is periodically reviewed and revised. F&P Robotics AG assumes no responsibility for any errors or omissions in this document.

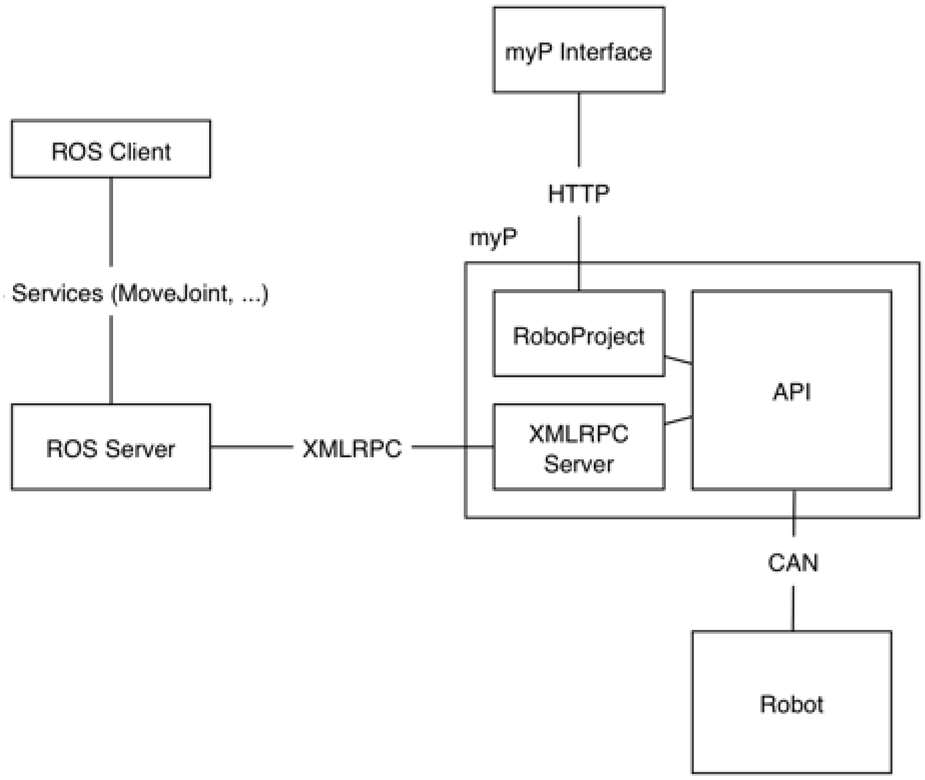
Copyright © 2011-2015 by F&P Robotics AG. All rights reserved.

The F&P logo as well as P-Rob®, P-Grip® and myP® are registered trademarks of F&P Robotics AG in Zurich, Switzerland.

Contents

1. Concept

Any Prob can be used remotely through the XML_RPC Server on myP. For the remote control of the Prob through ROS, a special ROS package was made, which contains all necessary tools for working with the Prob in ROS.
The ROS Package “prob_interface” contains the node “prob_server”. This node has a ROS Service for every function the XML_RPC Server offers and also for the most important MyP Script Commands.
A ROS Service can be called from any ROS client, a test example is included in the package. The prob_server communicates through XMLRPC to the myP API, which includes all necessary functions. For more infromation about how the XMLRPC server is working have a look at our XML_RPC Interface for Prob manual.



2. XML_RPC Server Commands

The XMLRPC interface works together with the webinterface of myP or any other client. So it’s possible to use the webinterface of myP to configure the robot and write a script and execute this script afterwards through an XMLRPC call. All scripts, paths and poses are stored in myP and are always up to date whatever client is used to create/edit them.

2.1 Command Reference

Robot control

```
initialize(model='PRob1R', kind='real', channel_name='1',
channel_type='PEAK_SYS_PCAN_USB', protocol='TMLCAN', host_id='10', baudrate='500000')
```

Initializes the P-Rob.

Parameter	*Type*	*Description*
model	String	Model of the robot as configured in the config file
kind	String	It can be: • **real:** real robot • **sim**: robot simulation • **no_robot**: no real robot
channel_name	String	Channel name of the connection to the P-Rob.
channel_type	String	Channel type of the connection to the P-Rob.
protocol	String	CAN protocol to use: 'TMLCAN' or 'CANOPEN'
host_id	String	Axis ID of the host computer, have to be different to the one of the axis.
baudrate	String	CAN bus baudrate, default value is 500'000.

```
finalize()
```

Disconnect the P-Rob.

```
calibrate(use_existing=True)
```

Calibrate the P-Rob.

Parameter	*Type*	*Description*
use_existing	Boolean	use the existing calibration values or do a new calibration

release(joint_id)

Releases the given joints.

Parameter	*Type*	*Description*
joint_id	Integer, List	a list of joints

hold(joint_id)

Hold the given joints.

Parameter	*Type*	*Description*
joint_id	Integer, List	a list of joints

pause()

Pause the P-Rob.

resume()

Resume a previous paused P-Rob.

stop()

Stops a movement or a running script of the P-Rob

recover()

Recover the P-Rob after an error.

Status

get_poses()

Get all saved poses.

get_paths()

Get the all saved paths.

get_current()

Get the current of all joints.

get_euler_position()

Get the euler position of all joints.

get_position()

Get the position of all joints.

get_actuator_release_state()

Get the release state of all joints.

False: Released

True: Hold

get_connection_info()

Get the connection state:

0: not initialized

1: initializing, not calibrated

2: initialized, not calibrated

3: initialized, calibrating

4: initialized, calibrated

get_status_info()

Get the status:

0: None

1: Ready

2: Stopped

3: Paused

4: Running

5: Released

6: Error

get_message_info()

If the status is 'ERROR', this function will return the message describing the error.

get_application_info()

Get the name and ID of the script which is currently active.

Script Control

get_scripts()

Get the name and ID of all saved scripts.

get_script(script_id)

Get the code of a saved script.

Parameter	*Type*	*Description*
script_id	Integer	ID of the script. If the ID is unknown, use get_script_id()

get_script_id(name)

Get the ID of the script.

Parameter	*Type*	*Description*
name	String	Name of the script

save_script(name, code)

Save a new script. The script supports all functions which are described in myP Script Command Reference Manual.

Parameter	*Type*	*Description*
name	String	Name of the script to save
code	String	Script code

delete_script(script_id)

Delete the specified script.

Parameter	*Type*	*Description*
script_id	Integer	ID of the script. If the ID is unknown, use get_script_id()

test_script(code)

Execute a script code directly without saving it. The script supports all functions which are described in myP Script Command Reference Manual.

Parameter	*Type*	*Description*
code	String	Script code

execute_script(script_id)

Execute a previous saved script. The script can be paused/resumed and stopped.

Parameter	*Type*	*Description*
script_id	Integer	ID of the script. If the ID is unknown, use get_script_id()

3. Installing ROS

1) Install ROS Indigo: http://wiki.ros.org/ROS/Installation
2) Create a catkin workspace
3) Copy the F&P package "prob_ interface" to the workspace
4) Compile the code and source your setup
```
$ catkin_make
$ source devel/setup.bash
```
5) run roscore
6) Now you are ready to run the example program

4. The ROS package prob_interface

The prob_interface package provides an interface from ROS to the Prob. With this package the Prob can be steered and programed like in MyP.

4.1 prob_server

This Node builds a connection to the XML_RPC server of MyP and offers ROS services for all necessary MyP functions.

Services

All the ROS Services of the prob_server take the arguments of the corresponding XML_RPC Server function as request input and yield a success boolean as result. There are Services which call XML_RPC Server commands directly like “initialize”, and there are Services which call MyP Script Commands like “move_joint” through the XML_RPC command “test_script”. This means that it is possible to execute every MyP Script Command there is. All the MyP Script Commands can be found in the MyP Script Command Reference Manual.

The following Services are implemented:

- initialize(model, kind, channel_name, channel_type, protocol, host_id, baudrate)
- calibrate(use_existing)
- test_script(script_code)
- execute_script(script_id)
- release(joind_ids)
- hold(joint_ids)
- move_joint(joint_ids, positions, velocity, acceleration, block, relative)
- move_tool(x, y, z, phi, theta, psi, velocity, acceleration, block, relative)
- move_linear(x, y, z, phi, theta, psi, velocity, acceleration, block, relative)
- move_to_pose(name, velocity, acceleration, block)
- open_gripper(angles, velocity, acceleration)
- close_gripper(velocity, acceleration, current)
- wait(time)

4.2 Clients

The package also contains useful client executables and client functions that make working with the prob_server even easier.

Client Functions

In the file ‘client_functions.py’ contains a function for every service of the prob_server, which takes the same arguments as the server and also has the same name as the server. So you can just import all the functions of the client functions file and work with them instead of writing your own service calls.

execute_script.py

This client executable takes a file path + name as argument and sends the whole content of the file to MyP. So it is possible to write a MyP Script into a file and execute it very easily with this client.

5. Examples

Requirements:

- ROS Indigo has to be installed
- prob_ros_interface is installed
- MyP needs to be running (start_server.py)
- XMLRPC Server needs to be running (xmlrpc_server.py)

Open a new Terminal and run roscore:

```
$ roscore
```

Start the Prob server with the IP address of the host that runs MyP in another Terminal:

```
$ rosrun prob_interface  prob_server.py 192.168.21.157
```

Open a third Terminal and run your client, for example the sample demo to move joint 1 of P-Rob by 10 degrees and back again:

```
$ rosrun prob_interface client_demo.py 1 10 45 60
$ rosrun prob_interface client_demo.py 1 0 45 60
```

Also possible is to write a script into a file and send it to the robot:

```
$ rosrun prob_interface execute_script.py filename
```

, where filename is the full path + name of the file containing the script,
for example ~/scripts/test_script.script

Example Script:

```
#This is an Example Script
open_gripper()
close_gripper()
move_joint(1,30) #Move Joint 1 to 30 Degrees
```