F&P Personal Robotics

~~myP~~ P-Rob (myP) Script Command Reference Manual

Version 1.0.10

(The versioning of this document corresponds to the versioning of the myP software)

2015-10-02~~2015-07-31~~

F&P Robotics AG
Rohrstrasse 36
8152 Glattbrugg
Switzerland

Contents

Introduction

This manual is intended for all users of the ~~myP~~ ROS interface to the ~~software and a~~ P-Rob robotic arm. This document contains a list of all functions to be used with P-Rob and P-Grip in ~~myP applications.~~the ROS service “test_script” or with the client “execute_script.py”. The test_script service the execute_script client send srcipt commands through the XMLRPC Server to myP, which then will execute these commands on the P-Rob.

For more information about P-Rob, P-Grip and ~~myP~~ROS, please refer to the **P-Rob User Manual**, **P-Grip User Manual** and ~~**myP User**~~**ROS Interface Manual**, respectively.

1. General Move Functions

Move Joint

move_joint(actuator_ids, position, velocity = None, acceleration = None, block = True, relative = False**)**

This function moves a joint or multiple joints of the robot to a specific position or set of positions in the joint space, respectively. This function has been designed such that the movement of each joint finishes exactly at the same time.

Parameter	*Types*	*Description*
actuator_ids	Various	This argument contains the IDs of the motors to move. It can be - the ID of a single joint (e.g. **6**), - a list of joint IDs for multiple joints (e.g. **[1,4,6]**), - the global constant **ALL_KIN_JOINTS**, which is equal to the list of all kinematic joints, - the global constant **ALL_ACTUATORS**, which is equal to the list of all actuators including end effectors.
position	Float or List of Floats	This argument contains the angles (in **degrees**) that are assigned to the joints. It can be - a single floating point number that will be assigned to each joint (e.g. -**30**), - a list of floating point numbers that will be assigned to each of the joints specified in **actuator_ids** (e.g. **[90, -45, 30]**).
velocity	Float or List of Floats	This argument contains the angular velocity (in **degrees per second**) that is used during the motion. It can be - a single floating point number that defines the maximum velocity of each joint (e.g. **45**), - a list of floating point numbers that will be assigned to each of the joints specified in **actuator_ids** (e.g. **[45, 25, 30]**). Please note that if you are using this option, the movements of the joints will not necessarily finish at the same time.
acceleration	Float or List of Floats	This argument contains the angular acceleration (in **degrees per second squared**) that are assigned to the joints. It can be - a single floating point number that defines the maximum acceleration of each joint (e.g. **60**), - a list of floating point numbers that will be assigned to each of the joints specified in **actuator_ids** (e.g. **[60, 35, 45]**). Please note that if you are using this option, the movements of the joints will not necessarily finish at the same time.
block	Boolean	If **True**, the next command will be executed once this motion has finished. If **False**, the next command will be executed without waiting for this motion to finish.
relative	Boolean	If **True**, the position values are interpreted as a relative offset to the current joint angles. If **False**, the position values are interpreted as absolute joint angles.

Example:

```
move_joint(ALL_KIN_JOINTS, 0)
```

Moving all kinematic joints of the robot to 0°, which executed in a calibrated robot causes it to stand up straight.

```
move_joint(6, -30, 45, 60, relative = True)
```

Moving joint 6 by -30° from its current position, with a velocity of 45°/s and acceleration of 60°/s^2.

```
move_joint([2,3,5], [20,-40,20], 35, 50)
```

Moving joints 2, 3 and 5 to 20°, -40° and 20°, respectively. The motion will be performed such that each joint will finish its movement at the same time. The joint that has the biggest angular distance to move, will move with a velocity of 35°/s and acceleration of 50°/s^2.

```
move_joint([1,4,6], [90, -45, 30], [45, 25, 30], [60, 35, 45], block = True)
# … this is similar to …
move_joint(1, 90, 45, 60, block = False)
move_joint(4, -45, 25, 35, block = False)
move_joint(6, 30, 30, 45, block = True)
```

Simultaneously moving three joints that will not finish their movements at the same point in time.

Move Tool

```
move_tool(x, y, z, orientation = None, velocity = None, acceleration = None, block = True, relative = False)
```

This function moves the tool center point (TCP) of the robotic arm to a specific target position. Optionally, the target orientation and the motion velocity and acceleration can be chosen.

Parameter	*Types*	*Description*
x, y, z	Floats	These values define the three dimensional position of the TCP (in **mm**) expressed in Cartesian coordinates in the world frame, the main coordinate system of the robot. For more details about the coordinate systems, please refer to the **myP User Manual**.
orientation	Float or List of Floats	These values define the orientation of the TCP (in **degrees**). The format of the orientation differs for different types of robots: - P-Rob 1R: the orientation is defined as **[roll, pitch, yaw]** - P-Rob 1U: the orientation is defined as the **angle of the uppermost joint**
velocity	Float	This argument specifies the linear maximum velocity of the TCP (in **mm per second**) during the motion (e.g. **250**).
acceleration	Float	This argument specifies the linear maximum acceleration of the TCP (in **mm per second squared**) during the motion (e.g. **300**).
block	Boolean	If **True**, the next command will be executed once this motion has finished. If **False**, the next command will be executed without waiting for this motion to finish.
relative	Boolean	If **True**, the pose values of the TCP are interpreted as relative offsets to the current pose components. If **False**, the pose values are interpreted as absolute values.

Example:

```
move_tool(200, 600, 350, [30, -20, -90])
```

Moving the TCP of a P-Rob 1R to the position [200mm, 600mm, 350mm] with orientation angles 30°, -20° and -90° for the roll, pitch and yaw angles, respectively.

```
move_tool(800, 0, 300, None, 250, 300)
```

Moving the TCP to the position [800mm, 0mm, 300mm] with a velocity of 250mm/s and acceleration of 300mm/s^2. The orientation is not given and will therefore be chosen by the function.

```
move_tool(0, 0, -20, -90, relative = True)
```

Moving the TCP of a P-Rob 1U by 20mm in the negative z-direction and changing the TCP orientation by -90° relative to the current TCP pose.

Move Linear

```
move_linear(x, y, z, orientation = None, velocity = None, acceleration = None, relative = False)
```

This function moves the tool center point (TCP) of the robot to target position along a linear path. Optionally, the target orientation and the motion velocity and acceleration can be chosen. Please note that there does not necessarily exist a linear path from the current position to the target position of the TCP, since during the motion some of the joints may try to move outside of their angular range to perform the requested movement.

Parameter	*Types*	*Description*
x, y, z	Floats	These values define the three dimensional position of the TCP (in **mm**) expressed in Cartesian coordinates in the world frame, the main coordinate system of the robot. For more details about the coordinate systems, please refer to the **myP User Manual**.
orientation	Float or List of Floats	These values define the orientation of the TCP (in **degrees**). The format of the orientation differs for different types of robots: - P-Rob 1R: the orientation is defined as **[roll, pitch, yaw]** - P-Rob 1U: the orientation is defined as the **angle of the uppermost joint** If the orientation is not given, this function tries to move to the end point without changing the orientation.
velocity	Float	This argument specifies the linear maximum velocity of the TCP (in **mm per second**) during the motion (e.g. **250**).
acceleration	Float	This argument specifies the linear maximum acceleration of the TCP (in **mm per second squared**) during the motion (e.g. **300**).
relative	Boolean	If **True**, the pose values of the TCP are interpreted as relative offsets to the current pose components. If **False**, the pose values are interpreted as absolute values.

Example:

```
move_linear(200, 600, 350, [30, -20, -90])
```

Moving the TCP of a P-Rob 1R to the position [200mm, 600mm, 350mm] with orientation [30°, -20°, -90°] in a linear movement.

```
move_tool(800, 0, 300, None, 250, 300)
```

Moving the TCP to the position [800mm, 0mm, 300mm] with a velocity of 250mm/s and acceleration of 300mm/s^2 in a linear movement. The orientation is not given and will therefore be chosen by the inverse kinematics.

```
move_tool(0, 0, -20, -90, relative = True)
```

Moving the TCP of a P-Rob 1U in a linear movement by 20mm in the negative z-direction and changing the TCP orientation by -90° relative to the current TCP pose.

Move to Pose

```
move_to_pose(pose_name, velocity = None, acceleration = None, block = True)
```

This function moves the robot to a pose that was previously saved in the database.

Parameter	*Types*	*Description*
pose_name	String or Integer	This argument contains the name of a pose as a string or the ID of the pose saved in the database as an integer number (e.g. **'myPose'** or **7**).
velocity	Float	This argument contains the angular maximum velocity (in **degrees per second**) that is used during the motion (e.g. **45**).
acceleration	Float	This argument contains the angular maximum acceleration (in **degrees per second squared**) that are assigned to the joints (e.g. **60**).
block	Boolean	If **True**, the next command will be executed once this motion has finished. If **False**, the next command will be executed without waiting for this motion to finish.

Example:

```
# poses with names 'start' (ID 1) and 'end' (ID 2) have been previously saved in the database
move_to_pose('start')
move_to_pose(2, 45, 60, block=False)
```

Moving to the pose 'start' (ID 1) with default velocity and acceleration, then moving to the pose 'end' (ID 2) with a maximum velocity of 45°/s and maximum acceleration of 60°/s^2 for each joint and awaiting new commands while moving.

Play Path

```
play_path(path_id, velocity_ratio = 1.0)
```

This function plays a path exactly as it has been previously recorded. Please note that not only the location, but also the velocity and the time of the path are reproduced. If the robot is not located at the starting pose of the path when playing the path, the joints will first move to this pose and then the path will be played.

Parameter	*Types*	*Description*
path_id	String or Integer	This argument contains the ID of the path to run (e.g. **3** or **'myPath'**, if this name is compatible with the name of a previously saved path).
velocity_ratio	Float	This argument defines the speed ratio with which the path is played (e.g. **0.8**) in relation to the original recorded speed.

Example:

```
# the pose 'myStartPose' (ID 1) has been previously saved in the database
# the path 'myPath' (ID 3) has been previously recorded and saved in the database
play_path('myPath')
move_to_pose('myStartPose', 30, 30)
play_path(3, 1.2)
```

This example plays the previously recorded path 'myPath' (ID 3) twice, once with the velocity it was recorded, and once with velocity increased by 20%.

2. General Gripper Functions

Open Gripper

```
open_gripper(position = None, velocity = None, acceleration = None)
```

This function opens the gripper of P-Grip with optional arguments such as opening angle, velocity or acceleration. By default, the gripper will open up to its upper angular limit.

Parameter	*Types*	*Description*
position	Float	This argument contains the opening angle (in **degrees**) of the gripper (e.g. **15**).
velocity	Float	This argument contains the angular velocity (in **degrees per second**) of the gripper (e.g. **200**).
acceleration	Float	This argument contains the angular acceleration (in **degrees per second squared**) of the gripper (e.g. **500**).

Example:

```
open_gripper()
```

Open gripper to its upper angular limit using default values for velocity and acceleration.

```
open_gripper(15, 200, 500)
```

Open gripper to an opening angle of 15° by moving with a velocity of 200°/s and an acceleration of 500°/s^2.

Close Gripper

```
close_gripper(position = None, velocity = None, acceleration = None)
```

This function closes the gripper of P-Grip with optional arguments for position, velocity and acceleration. By default, the gripper will close until either its lower angular limit is reached or an object is grabbed.

Parameter	*Types*	*Description*
position	Float	This argument contains the opening angle (in **degrees**) of the gripper (e.g. **15**).
velocity	Float	This argument contains the angular velocity (in **degrees per second**) of the gripper (e.g. **200**).
acceleration	Float	This argument contains the angular acceleration (in **degrees per second squared**) of the gripper (e.g. **500**).

Example:

```
close_gripper()
```

Close gripper by using default values for velocity and acceleration to grab an object (if there is one).

```
close_gripper(200, 500)
```

Close gripper and trying to grab an object by moving the fingers with a velocity of 200°/s and an acceleration of 500°/s^2.

```
close_gripper(15, 10, 10)
```

Close gripper very slowly down to an opening angle of 15° and trying to grab an object meanwhile.

Get Gripper Angle

```
get_gripper_angle()
```

This function reads the current gripper angle of P-Grip.

Parameter	*Types*	*Description*
return	Float	Returns the angle (in **degrees**) of the gripper.

Example:

```
close_gripper()
angle = get_gripper_angle()
if angle >= 10:
    print('big object has been grabbed')
elif 0 < angle < 10:
    print('small object has been grabbed')
else:
    print('no object has been grabbed')
```

By reading the gripper angle you can e.g. determine the size of an object. This example prints a message showing if a big or small object has been grabbed, by checking if the gripper angle is bigger or smaller than 10°.

3. Robot Calibration Functions

Calibrate Joint

```
calibrate_joint(actuator_id, direction)
```

This function calibrates one joint of the robotic arm into a desired direction. This function is useful to write custom calibration scripts, e.g. if the default calibration cannot be executed due to lack of space.

Parameter	*Types*	*Description*
actuator_id	String or Integer	This argument contains the joint ID of the joint to calibrate (e.g. **4** or **'joint4'**, if this name is compatible with the name of this kinematic joint defined in the configuration file).
direction	Integer	This argument contains the direction of the joint to calibrate. If the direction is a positive number, the joint will be calibrated in its positive rotation direction (e.g. **1** will cause a wrist joint to rotate around the positive z-axis or an elbow joint to rotate around the positive y-axis).

Finalize Calibration

```
finalize_calibration()
```

This function is used to set the status of the robot to 'calibrated' after a manual calibration has been executed.

Example:

```
calibrate_joint(4, 1)
calibrate_joint(1, -1)
move_joint([1, 4], 0)
calibrate_joint(3, 1)
calibrate_joint(2, -1)
move_joint([2, 3], 0)
finalize_calibration()
```

This example describes the default calibration procedure of a P-Rob 1U. First, the wrist joints (1 & 4) move into their opposite mechanical stops one after the other and move to their calibrated zero position. Then this procedure is repeated for the elbow joints (2 & 3). Finally the status of the robot is set to 'calibrated' and the robot thereafter is ready to execute further applications.

4. Learning Functions

Recognize Object

```
recognize_object(lesson_name = 0)
```

This function closes the gripper of P-Grip and uses its sensors to recognize an object that has been previously trained in a lesson. The object recognition compares all trained objects and returns the parameters of the object with the

Please note that you have to define multiple objects for P-Grip to recognize an object. You can also save and train P-Grip when it has no object

Parameter	*Types*	*Description*
lesson_name	String of Integer	This argument contains the ID or name of the lesson the object has been trained with (e.g. **4** or **'myLesson'**).
return	List	Returning all the following parameters of the recognized object in a list: - ID - name - weight (in **kg**) - color (as a list of **[R,G,B]** values) - size (as a list of **[x,y,z]** values) - force (in **N**)

Example:

```
# the objects 'smallObject' (ID 1) and 'bigObject' (ID 2) have been previously saved in the database
# additionally, 'noObject' (ID 3) has also been saved by training P-Grip without gripping any object at all
# a lesson 'myLesson' has been trained with all three predefined objects
object_parameters = recognize_object('myLesson')
if object_parameters[1] == 'smallObject':
    print('the small object has been recognized')
elif object_parameters[0] == 2:
    print('the big object has been recognized')
else:
    print('no object has been recognized')
print('the size of the recognized object is: ' + str(object_parameters[4]))
```

This example uses P-Grip to grab and its sensors to recognize one of two predefined objects. The program will then print which object has been recognized and the size of the recognized object.

5. Other Helpful Functions

Say

```
say(message, language = 'en')
```

This function converts a custom message to an audio file and executes this file. This function requires an active internet connection and we recommend to install the VLC player (with set environment variable) in order to use this function. The VLC player can be downloaded from http://www.videolan.org/vlc.

Parameter	*Types*	*Description*
message	String	This argument contains the message to say in the desired language.
language	String	This argument contains the language code of the message according to ISO 639-1 (e.g. **'en'** for English, **'de'** for German, **'fr'** for French or **'zh-CN'** for simplified Mandarin).

Example:

```
say('Hello world!')
say('Hallo Welt!', 'de')
```

Saying 'Hello world!' in English and German.

Wait

```
wait(seconds)
```

This function waits for time to pass before myP continues executing the next script command.

Parameter	*Types*	*Description*
seconds	Float	This argument contains the message to print.

Example:

```
print('Hello world!')
```

This example prints 'Hello world!' into the myP output terminal.

Run User Script

```
run_user_script(script_id)
```

This function executes a previously saved script within the running script. The chosen script is executed entirely before the main script continues.

Parameter	*Types*	*Description*
script_id	String or Integer	This argument contains the ID of the script to run (e.g. **4** or **'myScript'**, if this name is compatible with the name of a previously saved script).

Example:

```
# scripts named 'myFirstScript' (ID 1) and 'mySecondScript' (ID 2) have been previously saved
print('start main script')
run_user_script('myFirstScript')
print('continue executing main script')
run_user_script(2)
print('exit main script')
```

This example runs code of a main script and two other scripts named 'myFirstScript' (ID 1) and 'mySecondScript' (ID 2). The main script will wait for those two scripts to finish before continuing its execution.

6. Sensor Reading Functions

Get Sensor Data

```
get_sensor_data(sensor_name)
```

This function reads the data of a sensor available in myP by an open socket. Available are only sensors that are defined in the sensor configuration file (…/myP/config/sensor.config) and sensors that are currently implemented in myP.

Parameter	*Types*	*Description*
sensor_name	String	This argument contains the name of the sensor (e.g. **'mySensor'**, if this name is compatible with the name of a sensor defined in the sensor configuration file).
return	Various	Returns the data of a sensor. This data will vary for different types of sensors and depends on how this type of sensor is implemented in myP. (E.g. for the infrared sensors included in the fingers of P-Grip, this function will return a list of integers, one of each referring to one particular sensor. For further information on the finger sensors, please refer to the **P-Grip User Manual**.)

Example:

```
# for this example, a custom sensor 'mySensor' has been defined in the sensor configuration file
sensor_data = get_sensor_data('mySensor')
```

This example saves the current sensor data of a custom sensor into a variable.

```
# for this example, a P-Grip with finger sensors is required
# the sensors are defined with the name 'PGripSensors' in the sensor configuration file
open_gripper()
data_open = get_sensor_data('PGripSensors')
close_gripper()
data_close = get_sensor_data('PGripSensors')
print('infrared measurements for open gripper: ' + str(data_open))
print('infrared measurements for closed gripper: ' + str(data_closed))
```

This example prints the current infrared measurements of the P-Grip sensors in open and closed gripper position.

Get Joint Position

```
get_position(actuator_ids)
```

This function reads the position measurements of one or multiple joints.

Parameter	*Types*	*Description*
actuator_ids	Various	This argument contains the IDs of the joints of whose the position should be measured. It can be - the ID of a single joint (e.g. **6**), - a list of joint IDs for multiple joints (e.g. **[1,4,6]**), - the global constant **ALL_KIN_JOINTS**, which is equal to the list of all kinematic joints, - the global constant **ALL_ACTUATORS**, which is equal to the list of all actuators including end effectors.
return	List of Floats	Returns a list of position measurements (in **degrees**), one for each of the joints specified in **actuator_ids**.

Example:

```
position_data = get_position(ALL_KIN_JOINTS)
print('joint angles of all kinematic joints: ' + str(position_data))
```

This example prints the current angles of all kinematic joints.

Get Joint Current

```
get_current(actuator_ids)
```

This function reads the current measurements of one or multiple joints.

Parameter	*Types*	*Description*
actuator_ids	Various	This argument contains the IDs of the joints of which the current should be measured. It can be - the ID of a single joint (e.g. **6**), - a list of joint IDs for multiple joints (e.g. **[1,4,6]**), - the global constant **ALL_KIN_JOINTS**, which is equal to the list of all kinematic joints, - the global constant **ALL_ACTUATORS**, which is equal to the list of all actuators including end effectors.
return	List of Floats	Returns a list of current measurements (in **A**), one for each of the joints specified in **actuator_ids**.

Example:

```
current_data = get_current(4)
print('current of joint 4: ' + str(current _data))
```

This example prints the current velocities of the joints 2 & 3

7. Basic Python Programming

Basic Python Standards

commentary

```
# this is a comentary and will not be considered during code execution
```

variable definitions and declarations

```
myBoolean = True          # here, myBoolean is defined as a Boolean with value True
myInteger = 3             # here, myInteger is defined as an integer number with value 3
myFloat = 3.1416          # here, myFloat is defined as a floating point number 3.1416
myString = “Hello World!” # here, myString is defined as a string with value “Hello World!”
```

printing messages

```
print(myString)           # prints the value of myString
```

list (array) definitions and usage

```
myList = [1,2,3,4,5]      # create a list containing some numbers
myList[0] = 6             # sets the first list element to be 6
```

converting variables

```
x = int(x)                # converts x to an integer
x = float(x)              # converts x to a floating point
x = str(x)                # converts x to a string
```

time statements

```
time.sleep(x)             # sleep for x seconds
x = time.time()           # save current system time in variable x
```

Example: calculate the duration of a code segment.

```
start_time = time.time()          # get current system time
…                                 # insert your code here
stop_time = time.time()           # get current system time again
duration = stop_time - start_time # get the duration (in seconds)
```

Basic Python Operators

Calculation Operators

```
x = 3 + 5          # addition:        x has now value 8
x = 7 - 2          # subtraction:     x has now value 5
x = 4 * 8          # multiplication:  x has now value 32
x = 54 / 6         # division:        x has now value 9
x = 16 % 7         # modulo:          x has now value 2
x = 2 ** 3         # exponential:     x has now value 8
```

Comparison Operator

```
x == y             # returns true if x is equal to y
x != y             # returns true if x is not equal to y
x > y              # returns true if x is greater than y
x < y              # returns true if x is smaller than y
x >= y             # returns true if x is greater than or equal to y
x <= y             # returns true if x is smaller than or equal to y
```

Logical Operators

Here x and y are Booleans.

```
x and y            # logical AND operator:   returns true if x and y are true
x or y             # logical OR operator:    returns true if either x or y is true
not x              # logical NOT operator:   returns true if x is false
```

Conditional Programming

"if", "elif" (else if) and "else" conditions can be combined according to the following rules:

If Condition

The "if" scope is executed if and only if its condition returns the value True.

```
if condition:
    …
```

Elif Condition

An "elif" (else if) scope is executed if and only if previous "if" or "elif" conditions returned the value False and its own condition returns the values True.

```
if firstCondition:
    …
elif secondCondition:
    …
elif thirdCondition:
    …
```

Else Condition

An "else" scope is executed if and only if all the previous "if" or "elif" conditions returned the value False.

```
if condition:
    …
else:
    …
```

Example: check if the variable myInteger is equal to 5, 10, or neither one of those numbers.

```
if myInteger == 5:
    print "your integer is 5."
    print "have a nice day, sir."
elif myInteger == 10:
    print "your integer is 10."
    print "have a nice day, sir."
else:
    print "your integer is neither 5 nor 10."
    print "have a nice day anyway."
```

Loops

While Loop

A while loop is executed and repeated as long as its condition returns the value True.

```
while condition:
    …
```

Example: printing all numbers from 1 to 10.

```
i = 1
while(i <= 10):
    print(i)
    i = i + 1
```

For Loop

A for loop iterates over the items of any sequence, such as a list or a string.

```
for item in sequence:
    …
```

Example: printing all numbers from 1 to 5.

```
for i in range(5):
    print(i)
```

Example: printing my favorite fruits.

```
for fruit in [“apple”, “banana”, “orange”]:
    print “one of my favorite fruits is ” + str(fruit)
```

Example: counting out the letters in ‘python’.

```
i = 1
for letter in “python”:
    print(“letter number “ + str(i) + “ in ‘python’ is “ + str(letter))
    i = i + 1
```