F&P
Personal Robotics

Motion Control Module

User Manual

MCM v0.18.0

F&P Robotics AG 15.11.2017

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.

Contents

1. Introduction

The Motion Control Module provides the control interface of PRob2. It aims at an easy integration into different existing software frameworks using JSON over TCP/IP as a means of communication. It is intended to be used for Machine-to-Machine (M2M) communication and provides also an optional text-based Human-to-Machine interface (HMI).

Features

- Calibration (using mechanical stops or light barriers, if available)
- Position control loop up to 100Hz providing smooth movement
- Release mode allowing positioning the robot by hand with friction and gravity compensation
- Sophisticated safety checks in all modes
- Well tested (coverage estimated at 95% of the code base)
- TCP JSON-based M2M interface

Options

- TCP text-based HMI interface
- C++ API
- Python API
- ZMQ interface

Getting started

1. Power up the robot according to the PRob2 User Manual.
2. Establish a TCP (Transmission Control Protocol) connection to the robot on **port 18010**. (Factory IP Address is 192.162.80.203, but might set differently by your administrator). It is recommended to use the TCP_NODELAY option on your connecting socket.
3. Send your desired commands in valid JSON-Format (JavaScript Object Notation) as specified in the following chapter. The JSON string must be:
    a. ASCII encoded
    b. The entire string must be on one line
4. Parse the results which are sent as a reply (depends on the commands sent).

Basic Python3 Example

```
import json

import socket

import time

mcm_host = (“192.168.80.203”, 18010)

sock = socket.create_connection(mcm_host)

sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
```

```
command_string = “{"command":"release"}”  # JSON command
sock.send(command_string.encode(“ascii”))
time.sleep(0.1)
data_bytes = sock.recv(4096)
result_string = data_bytes.decode(“ascii”)
result_json = json.loads(result_string)
print(result_json[“result_type”])
```

See the ‘examples’ folder in the MCM distribution you received for more sophisticated examples.

Operating modes

The MCM hasthe following operating modes as documented in the State Diagram in [1]:

- **OFF/ERROR:** Axes are kept at their current positions. Axes powered OFF or in error state
- **HOLD**: Axes are kept at their current positions
- **RELEASE**: Axes can be moved by hand
- **TORQUE_CONTROL**: Torques provided by external controller
- **PVT**: Smooth position control

The **OFF/ERROR** mode is equivalent to **HOLD** with an error state (issuing a ‘hold’ command clears the error, see State transitions below).

All mode transitions are via the **HOLD** mode, i.e. there is no direct transition between **RELEASE** and **PVT**.

A **VELOCITY_CONTROL** mode is planned for the future.

Safety checks

The following safety checks are active in **RELEASE**, **TORQUE_CONTROL** and **PVT** mode:

- Check that robot is calibrated on entering mode
- Check of position and velocity limits
- Dynamic check of position and velocity. If an axis is moving towards a hard stop at high speed, it is stopped if the current position plus braking distance with maximum deceleration exceeds the position limit.

The following safety checks apply in **ALL** modes:

- Monitor Emergency switch (implemented in hardware)
- Monitoring of motor controller connections

- Check that actual position is within a defined band of the commanded position (applies to **HOLD** and **PVT**)
- i2t protection: Check that the energy dissipated by each motor stays within safe limits. If these are exceeded, a reduced maximum motor current applies.

Violation of any safety check will put the robot in the **OFF / ERROR** mode which is equivalent to **HOLD**. Issuing a ‘hold’ command in **OFF / ERROR** tries to clear the error and re-enable power.

2. Commands

All commands must be sent in valid JSON format **on a single line** i.e. **no newline** characters must be contained within the JSON object. The end of a command must be marked by a newline character.

For readability in the present manual, JSON messages are displayed on several lines.

Values and units

- Axes (any <axis> parameter) are numbered from 1 to 6 (lowest to highest axis when the robot is upright)
- Axis angles are in degrees and zero when the robot is in upright position.
- Axis velocities are in degrees/s, accelerations in degrees/s^2.
- Torques are in Newton-meters.
- Motor currents are in Amperes.
- The velocity_factor parameter is dimensionless and specifies a ratio of the maximum velocity for any given axis. It is optional and defaults to 0.33333.

General

Calibrate

Must be executed after each start up of the robot and before the robot is moved in any other way.

Calibrates the robot
<which>: “ALL”, “ELBOW”, “WRIST” or 1..6
<elbow_direction>: +-1; if -1, reverse elbow direction
<wrist_direction>: +-1; if -1, reverse wrist direction
<force_mechanical_stop>: If 1, forces using the mechanical stops even if light barriers are installed
After a power cycle the robot needs to be re-calibrated.

```
{
  "command"  :"calibrate",
  "arguments": (optional)
  {
    "which":<id>,                  (optional)
    "elbow_direction":<direction>, (optional)
    "wrist_direction":<direction>, (optional)
    "force_mechanical_stop":<bool> (optional)
  }
}
```

Enter Hold

Enters **HOLD** mode and clears any errors.

```
{
"command":"hold"
}
```

Request Status

detail: Whether or not a detailed status report is requested

```
{
  "command"   : "status",
  "arguments" : (optional)
  {
    "detail"  : <bool>
  }
}
```

Notes:

- In **PVT** mode, the reported motor currents are measured values and axis positions and velocities are the commanded values.
- In **RELEASE** and **TORQUE_CONTROL** modes, the reported axis positions and velocities are measured values and currents are the commanded values.

Sleep

duration: time period in seconds before execution of next command

```
{
  "command":"sleep",
  "arguments":
  {
    "duration":<seconds>
  }
}
```

Hold mode

Enter PVT

dt: specifies the interval between evenly spaced PVT points in seconds (default: 0.01s, i.e. 10ms)

```
{
  "command"   : "pvt",
  "arguments" : (optional)
  {
    "dt":<seconds>
  }
}
```

Enter Release

Enters **RELEASE** mode in which the robot arm can be positioned manually.

```
{

  "command":"release"

}
```

Enter Torque Control

timeout: maximum allowed interval between “set_torque” commands in seconds. The robot will enter **HOLD** if more time passed between the commands. No timeout applies iff the value is negative.

dt_status: the robot will send a status result approx. in this interval (Actually, in the next cycle after *dt_status* passed e.g. dt_status=0.01, if the cycle time is however 8ms, the status will be sent every 0.016 seconds).

```
{
  "command"   : "torque_control"
  "arguments" :
  {
    "timeout"   : <seconds>,
    "dt_status" : <seconds>
  }
}
```

PVT mode

Moves the given axis from current position to <angle> using <velocity_factor>.

```
{
  "command"   : "move_axis",
  "arguments" :
```

```
  {
    "axis_index"      : <axis> ,
    "target_position" : <angle>,
    "velocity_factor" : <velocity_factor> (optional)
  }
}
```

Release mode

Possible commands are “hold” to switch back to **HOLD** mode and “status” to request status info.

Torque Control mode

Set torques

Sets torques for all axes to the provided values [Nm].

Notice: If a positive timeout was given in the “torque_control”, “set_torques” commands must be sent in rapid succession so that the timeout isn’t exceeded. You’ll typically want to wait for a status message (see parameter “dt_status” in “torque_control”) and then compute and send the new torque settings immediately.

```
{
  "command"   : "set_torques",
  "arguments" :
  {
    "torques" : [ <torques> ]
  }
}
```

Other commands

Use “hold” to switch back to **HOLD** mode and “status” to request status info.

3. Results

This chapter documents the “result” data structures the robot sends on the TCP/JSON channel.

General remarks

There is no 1:1 mapping between commands and results. Many commands yield a generic result indicating success or failure. Mode change commands and the status command yield a status result containing robot axis state and supplementary information.

The communication pattern is roughly request/reply. However, in the following cases results are sent “spontaneously” by the robot:

- Errors
- Status updates in **TORQUE_CONTROL** mode if requested by the dt_status parameter

Generic result

This result type is provided unless another type is specified for a command.

The “error_status” can be “OK” or “ERROR”.

If “error_status” is not “OK”, “error_message” contains detailed error information.

```
{
  "result_type" : "generic",
  “error_status”: “<error status>”,
  “error_message”: “<detailed information on any error condition>” (optional)
}
```

Status result

```
{
  "result_type"    : "status",
  “operating_mode” : “<operating mode>”,
  “axis_states” : {
    “time”         : [ <measurement times>  ],
    “position”     : [ <axis positions>     ],
    “velocity”     : [ <axis velocities>    ],
    “acceleration” : [ <axis accelerations> ],
    “current”      : [ <axis currents>      ]
  }
  “hardware_status” : “<detailed hardware status>” (optional)
}
```

Move_all result

The move_all result is issued immediately following a move_all command and contains information about the movement that is about to commence. Values are based on the linear movement to the target positions.

TODO: Explain the reference generator and interpretation of these values.

```
{
  "result_type" : "move_all",
  “velocities”  : [ <Axis velocities used> ],
  “longest”     : <Index of axis that takes the longest time [1..6]>
  “max_time_to_target” : <Maximum time to target over all axes [s]>
}
```

Initialization result

This result is issued in response to an “info” command (to be implemented).

```
{
  "result_type" : "initialization",
  “num_axes”    : <Number of axes on the robot (typically: 6)>
  // TODO: Add more robot information like serial number, hardware version etc.
}
```

References

[1] MCM diagrams: https://drive.google.com/drive/folders/0B6JAIXgHqOgfc29LV0lVa1NvbGM

kk   kk