Description to `rob6server` – Version 1.0.9

Sven-Thomas Antoni
german version until 1.01 by Floris Ernst

March 16, 2017

Contents

1 Introduction

This documents gives a list of commands that can be used for communication with the `rob6server` and gives a short introduction on how to use it.

1.1 Units

To control the server, the following units are used:

- **mm** for displacements
- **degree** for angles
- **kg** for weights

2 Starting the server

- Usage:

  ```
  rob6server [params] <robot-type>
  ```

- `robot-types`:

`ur5`	Universal Robots UR5
`ur3`	Universal Robots UR3
`abb`	ABB IRB120
`kr3`	Kuka KR3
`kr16`	Kuka KR16
`ad850`	Adept Viper s850
`kaw_fs`	Kawasaki FS003N

- `params`: (optional)

`-p`	PORTNO	listen on port PORTNO (default: 5005)
`-s`	SERIAL	connect to Kuka on SERIAL (default: /dev/ttyS0)
`-a`	IP	connect to robot on IP (default UR5: 134.28.45.57)
`-A`	PORTNO	connect to robot on PORTNO (default: 30000)
`-b`	IP	robot connects to this IP on the server (only UR3 and UR5; default: 134.28.45.36)
`-B`	PORTNO	robot's joint thread connects to this PORTNO on the server (only UR3 and UR5; default: 30005)
`-d`		dummy mode (default: disabled)
`-h`		print this help message
`-i`		more information

3 Connection

In order to connect to the robot server (`rob6server`) the server needs to be running. This includes initialization of the robot as well as the robot server. Once this is successfully accomplished the `rob6server` waits for a client. Only one client is able to properly connect to the robot server at once.

3.1 Connect to the server as a client

3.1.1 Connect to the robot server

The connection is accomplished via TCP-sockets. The default port for connection is 5005, the server can **not** handle multiple clients.

Example:

start `telnet 127.0.0.1 5005`
receive `Trying 127.0.0.1...`
receive `Connected to 127.0.0.1.`
receive `Escape character is '^]'`
receive `Welcome to rob6server 0.1.01 !`

3.1.2 Initiate communication between server and client

In order to initiate the communication with the server the command `Hello Robot` needs to be passed to the server.

Example:

send `Hello Robot`
receive `accepted`

Now the client can use all functions as stated in the next section.

4 Commands

4.1 Commands for managing the server

- `Help` **Only for UR3, UR5 and ABB**

  Returns a list of possible commands for the connected robot.

  Example:

  send `Help`
  receive list of commands

- `GetRobot`

  Returns the currently connected robot, possible return values are: `ad850`, `kaw_fs`, `kr3rt`, `kr16rt`, `ur3`, `ur5`, `abb` and (outdated) `kr3`, `kr16`

  Example:

send	`GetRobot`
receive	`ad850`

- `Is[Adept|Kuka|Kawa|KR3|KR16|UR|UR3|UR5|ABB]`

  Returns `true` if the connected robot is of given type.

  Example:

send	`GetRobot`
receive	`ad850`
send	`IsAdept`
receive	`true`
send	`IsKawa`
receive	`false`

- `GetVersion`

  Returns the version number of the server.

  Example:

send	`GetVersion`
receive	`0.1.08`

- `Quit`

  Closes the connection.

  Example:

send	`Quit`
receive	`bye!`

- `Shutdown`

  Closes the connection and shuts down the server.

  Example:

send	`Quit`
receive	`bye!`
	shutting down ...

- `GetTimestamp`

  Gets the current timestamp from the server

  Example:

send	`GetTimestamp`
receive	`1285831711.121`

- `PingRobot num wait` **Only Adept-Robot**

  Gets the communication latency between server and robot. Here `num` is the number of pings to do and `wait` the time between two pings in milliseconds.

  Example:

send	`PingRobot 100 50`
receive	`0.003414 3659.8930000000 1285828592.9111907482 -0.00016902738 1.4625943e-05`

  The first number is the mean time for the robot to answer (in seconds), the second number is the neutral point of the robot time, the third number is the neutral point of the server time and the forth and fifth values are the coefficients of degree 2 to calibrate both times.

- `CM_PING`

  Pings the server.

  Example:

send	`CM_PING`
receive	`PONG`

- `IsAlive` **Only UR-Robots**

  Verifies that the robot is still connected and the robot driver running.

  Example:

send	`IsAlive`
receive	`true`

- `SetVerbosity num`

  Sets the verbosity of the robot server. The following values for `num` are possible:

  0 — nothing
  1 — only errors
  2 — errors & warnings
  3 — errors, warnings and communication data
  4 — everything

  Example:

send	`SetVerbosity 4`
receive	`true`

4.2 Commands to alter movement

- `EnableAlter`

  Activates the realtime mode. Depending on the connected robot and the server software on the robot this mode is more or less hard. Proper realtime control is working for `kr3rt` and `kr16rt`.

  Example:

send	`EnableAlter`
receive	`true`

- `EnableAdeptAlter` **Only Adept-Robot**

  Activates the hard realtime mode for the `ad850` if `serveralter` is running.
  Example:

send	`EnableAdeptAlter`
receive	`true`

- `EnableURAlter` **Only UR-Robots**

  Activates the hard realtime mode for the `ur3` and `ur5`.
  Example:

send	`EnableURAlter`
receive	`true`

- `EnableFreedrive` **Only UR-Robots**

  Activates the freedrive mode. This allows manual manipulation of the robot. To avoid damage to the robot the Payload needs to be set correctly. No movement commands can be executed while in this mode. Not available in realtime mode.

  Example:

send	`EnableFreedrive`
receive	`true`

- `EnableBlocking` **Only for UR3, UR5 and ABB**

  Disables the waypoint queue. The server's command line will block until the robot has reached the designated position. Not available in realtime mode.

  Example:

send	`EnableBlocking`
receive	`true`

- `EnableLin`

  Activates the linear movement mode.

  Example:

send	`EnableLin`
receive	`true`

- `EnableKeepAlive` **Only UR-Robots**

  If enabled, the server will detect connection losses and automatically recover. For this mode, movements using the panel are not advised. (Default on)

  Example:

send	`EnableKeepAlive`
receive	`true`

- `DisableAlter` **Not UR3, UR5 and ABB**

  Deactivates the realtime mode. Movement is done in a point-to-point sense. This means the server waits after every movement command until the robot has reached its destination.

  Example:

send	`DisableAlter`
receive	`true`

- `DisableAlter` **Only for UR3, UR5 and ABB**

  Deactivates the realtime mode. Movement is done in a point-to-point sense. All sent movement commands are saved in a queue on the robot and are executed in order.

  Example:

send	`DisableAlter`
receive	`true`

- `DisableFreedrive` **Only UR-Robots**

  Deactivates the freedrive mode. Allowing control of the robot only through the server or the panel.

  Example:

send	`DisableFreedrive`
receive	`true`

- `DisableBlocking` **Only for UR3, UR5 and ABB**

  Enables the wayoint queue. All sent movement commands are saved in a queue on the robot and are executed in order.

  Example:

send	`DisableBlocking`
receive	`true`

- `DisableLin`

  Deactivates the linear movement mode.

  Example:

send `DisableLin`
receive `true`

- `DisableKeepAlive` **Only UR-Robots**

  Disables the automatic recovery from disconnects between server and robot. This allows for using the panel while the server is running but manual recovery of the resulting disconnect is necessary.

  Example:

  send `DisableKeepAlive`
  receive `true`

- `SetRTSpeedControl 0|1` **Only KUKA-Robots**

  Sets the behavior in realtime mode. If this is active the robot will drive ramps in realtime mode, else it will move its joint in a way that they arrive simultaneously.
  Example:

  send `SetRTSpeedControl 1`
  receive `true`

4.3 Commands for movement

4.3.1 Payload

- `SetPayload` w **Only for UR3, UR5 and ABB**

  Sets the TCP payload to w.
  Example:

  send `SetPayload 0.1`
  receive `true`

- `SetPayloadCOG` w cog_x cog_y cog_z **Only for UR3, UR5 and ABB**

  Sets the TCP payload to w and the center of gravity (COG) to $[cog_x, cog_y, cog_z]$ in TCP coordinates.
  Example:

  send `SetPayloadCOG 0.1 10 0 5`
  receive `true`

4.3.2 Precision of movement

- `SetAdeptFine` v **Only Adept-Robot**

  Activates fine positioning with a precision of v percent of the joints default precision.
  Example:

  send `SetAdeptFine 50`
  receive `true`

- `SetAdeptCoarse` v **Only Adept-Robot**

  Activates coarse positioning with a precision of v percent of the joints default precision.

  Example:

  send `SetAdeptCoarse 50`
  receive `true`

- `SetBlend num` **Only for UR3, UR5 and ABB**

  Sets the blend radius in mm. Has no effect in realtime mode.

  Example:

  send `SetBlend 5`
  receive `true`

- `SetFine num` **Only for UR3, UR5 and ABB**

  Sets the blend radius to 0mm.

Example:

send	`SetFine`
receive	`true`

4.3.3 Speed and acceleration

- `SetSpeed` v **Only for UR3, UR5 and ABB**

  Sets the speed given in percent of the maximum value. Values up to $v = 120$ are possible.

  Example:

send	`SetSpeed 10`
receive	`true`

- `SetSpeedJoints` v **Only for UR3, UR5 and ABB**

  Sets the maximum speed for joint motion, given in degree per second.

  Example:

send	`SetSpeedJoints 10`
receive	`true`

- `SetSpeedLIN` v **Only for UR3, UR5 and ABB**

  Sets the maximum speed for linear PTP motion, given in millimeters per second.

  Example:

send	`SetSpeedLIN 10`
receive	`true`

- `SetAccel` a **Only for UR3, UR5 and ABB**

  Sets the acceleration in percent of the maximum value. Values up to $a = 150$ are possible.

  Example:

send	`SetAccel 10`
receive	`true`

- `SetAccelJoints` a **Only UR-Robots**

  Sets the maximum acceleration for joint motion, given in degree per seconds square.

  Example:

send	`SetAccelJoints 10`
receive	`true`

- `SetAccelLIN` a **Only UR-Robots**

  Sets the maximum acceleration for linear PTP motion, given in millimeters per seconds square.

  Example:

send	`SetAccelLIN 10`
receive	`true`

- `SetURAccel` a **Only UR-Robots**

  Sets the acceleration in percent of the maximum value. Values up to $a = 150$ are possible.

  Example:

send	`SetUR5Accel 100`
receive	`true`

- `SetABBAccel` a r **Only ABB-Robots**

  Sets the acceleration (a) and acceleration ramp (r) in percent of the maximum value. Values up to $a, r = 150$ are possible.

Example:

send `SetABBAccel 100 50`
receive `true`

- `SetAdeptSpeed` v **Only Adept-Robot**

  Sets the speed given in percent of the maximum value. Values up to $v = 120$ are possible.

  Example:

  send `SetAdeptSpeed 100`
  receive `true`

- `SetAdeptAccel` a_1 a_2 **Only Adept-Robot**

  Sets the acceleration and deceleration given in percent of the maximum value. Values up to $a_1, a_2 = 120$ are possible.

  Example:

  send `SetAdeptAccel 100 50`
  receive `true`

- `SetKukaRTSpeed` v_1 v_2 v_3 v_4 v_5 v_6 **Only KUKA-Robot with RT-Interface**

  Sets the speed for the six different joints.

  Example:

  send `SetKukaRTSpeed 0.3 0.1 0.1 0.1 0.1 0.1`
  receive `true`

- `SetJointsMaxSpeed` v_1 v_2 v_3 v_4 v_5 v_6 **Only KUKA-Robots**

  Sets the speed for the six different joints.

  Example:

  send `SetJointsMaxSpeed 0.3 0.1 0.1 0.1 0.1 0.1`
  receive `true`

- `SetSingleJointMaxSpeed` j a **Only KUKA-Robots**

  Sets the speed for a single joint.

  Example:

  send `SetSingleJointMaxSpeed 3 0.3`
  receive `true`

- `SetJointsMaxAcceleration` a_1 a_2 a_3 a_4 a_5 a_6 **Only KUKA-Robots**

  Sets the acceleration for the six different joints.

  Example:

  send `SetJointsMaxAcceleration 0.005 0.001 0.01 0.01 0.01 0.05`
  receive `true`

- `SetSingleJointMaxAcceleration` j a **Only KUKA-Robots**

  Sets the acceleration for a single joint.

  Example:

  send `SetSingleJointMaxAcceleration 3 0.1`
  receive `true`

- `GetJointsMaxSpeed` **Only KUKA-Robots**

  Returns the set speed of all six joints.

Example:

send	`GetJointsMaxSpeed`
receive	`0.187200 0.187200 0.187200 0.396000 0.396000 0.738000`

- `GetJointsMaxAcceleration` **Only KUKA-Robots**

  Returns the set acceleration of all six joints.

  Example:

send	`GetJointsMaxAcceleration`
receive	`0.005000 0.005000 0.005000 0.005000 0.005000 0.005000`

- `ResetJointsMaxSpeed` **Only KUKA-Robots**

  Resets speed to default.

  Example:

send	`ResetJointsMaxAcceleration`
receive	`true`

- `ResetJointsMaxAcceleration` **Only KUKA-Robots**

  Resets acceleration to default.

  Example:

send	`ResetJointsMaxAcceleration`
receive	`true`

4.3.4 Joint limits

- `GetJointsMaxChange`

  Returns the maximal rotation angle of the joints for a single movement.

  Example:

  send `GetJointsMaxChange`
  receive `370.000000 175.000000 284.000000 700.000000 250.000000 700.000000`

- `SetJointsMaxChange` α_1 α_2 α_3 α_4 α_5 α_6

  Sets the maximal rotation angle of the joints for single movements. Six angles (degree) are expected.

  Example:

  send `SetJointsMaxChange 10 10 10 10 10 10`
  receive `true`

- `SetSingleJointMaxChange` j α

  Sets the maximal rotation angle per movement of the joint j to α.

  Example:

  send `SetSingleJointMaxChange 3 20`
  receive `true`

- `GetJointsMaxTurnMax`

  Returns the maximal rotation angle of the joints.

  Example:

  send `GetJointsMaxTurnMax`
  receive `185.000000 20.000000 154.000000 350.000000 125.000000 350.000000`

- `SetJointsMaxTurnMax` α_1 α_2 α_3 α_4 α_5 α_6

  Sets the maximal rotation angle of the joints.

  Example:

  send `SetJointsMaxTurnMax 10 10 10 10 10 10`
  receive `true`

- `SetSingleJointMaxTurnMax` j α

  Sets the maximal rotation angle of the joint j to α.

  Example:

  send `SetSingleJointMaxTurnMax 3 40`
  receive `true`

- `GetJointsMaxTurnMin`

  Return the minimal rotation angle of the joints.

  Example:

  send `GetJointsMaxTurnMin`
  receive `-185.000000 -155.000000 -130.000000 -350.000000 -125.000000 -350.000000`

- `SetJointsMaxTurnMin` α_1 α_2 α_3 α_4 α_5 α_6

  Sets the minimal rotation angle of the joints. Six angles (degree) are expected.

  Example:

  send `SetJointsMaxTurnMin -10 -10 -10 -10 -10 -10`
  receive `true`

- `SetSingleJointMaxTurnMin` j α

  Sets the minimal rotation angle of the joint j to α.

Example:

send `SetSingleJointMaxTurnMin 3 -40`
receive `true`

- `ResetJointsMaxChange` j α

  Resets the maximal and minimal rotation angle of the joints per movement to default.

  Example:

  send `ResetJointsMaxChange`
  receive `true`

- `ResetJointsMaxTurn` j α

  Resets the maximal and minimal rotation angle of the joints to default.

  Example:

  send `ResetJointsMaxTurn`
  receive `true`

4.3.5 Movement

- `MoveMinChangeRowWiseStatus` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$ `flip|noflip|toggleHand|noToggleHand` `up|down|toggleElbow|noToggleElbow` `lefty|righty|toggleArm|noToggleArm`

  This command is used to execute a point-to-point (PTP) movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) and the robot configuration is submitted. If the target pose can be reached with different allowed configurations the configuration is chosen where the change in joint angles is minimal.

  Example:

  send `MoveMinChangeRowWiseStatus 0 0 -1 1768 0 -1 0 0 -1 0 0 640 flip toggleElbow toggleArm`
  receive `true`

- `MovePTPJoints` j_1 j_2 j_3 j_4 j_5 j_6

  Moves the robot in PTP-mode to a new joint position.
  Example:

  send `MovePTPJoints 10 0 0 0 0 0`
  receive `true`

- `MoveRTHomRowWise` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$

  Only KUKA (RT), Adept, UR3, UR5 and ABB in soft-RT-mode

  This command is used to execute a realtime (RT) movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) is submitted. The current configuration is kept.

  Example:

  send `MoveRTHomRowWise 0 0 -1 1768 0 -1 0 0 -1 0 0 640`
  receive `true`

- `MoveRTHomRowWise` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$

  Only UR3 and UR5 in hard-RT-mode

  This command is used to execute a realtime (RT) movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) is submitted. The pose is directly send to the robot and neither configuration nor possibility or accordance to set joint turn limits are checked.

  Example:

  send `MoveRTHomRowWise 0 0 -1 1768 0 -1 0 0 -1 0 0 640`
  receive `true`

- `MoveRTHomRowWiseStatus` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$ `flip|noflip|toggleHand|noToggleHand` `up|down|toggleElbow|noToggleElbow` `lefty|righty|toggleArm|noToggleArm`

  This command is used to execute a RT movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) and the robot configuration is submitted. If the target pose can be reached with different allowed configurations the configuration is chosen where the change in joint angles is minimal.

  Example:

  send `MoveRTHomRowWiseStatus 0 0 -1 1768 0 -1 0 0 -1 0 0 640 flip toggleElbow toggleArm`
  receive `true`

- `MoveRTJoints` j_1 j_2 j_3 j_4 j_5 j_6 **Only KUKA (RT), Adept, UR3, UR5 and ABB in soft-RT-mode**

  Moves the robot in RT-mode to a new joint position.
  Example:

  send `MoveRTJoints 10 20 10 10 10 10`
  receive `true`

- `MoveRTJoints` j_1 j_2 j_3 j_4 j_5 j_6 **Only UR3 and UR5 in hard-RT-mode**

  Moves the robot in RT-mode to a new joint position. The joint positions are directly send to the robot and neither configuration nor possibility or accordance to set joint turn limits are checked.
  Example:

  send `MoveRTJoints 10 20 10 10 10 10`
  receive `true`

- `MoveLINJoints` j_1 j_2 j_3 j_4 j_5 j_6

  Moves the robot in linear-mode to a new joint position.

  Example:

  send `MoveLINJoints 10 0 0 0 0 0`
  receive `true`

- `MoveLINHomRowWise` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$

  Not UR3, UR5 and ABB

  This command is used to execute a linear movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) is submitted. The current configuration is kept.

  Example:

  send `MoveLINHomRowWise 0 0 -1 768 0 -1 0 0 -1 0 0 640`
  receive `true`

- `MoveLINHomRowWise` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$

  Only for UR3, UR5 and ABB

  This command is used to execute a linear movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) is submitted. As the trajectory is calculated on the robot, no guarantees can be given regarding the configuration for the target, configuration while moving and violations of joint turn limits.

  Example:

  send `MoveLINHomRowWise 0 0 -1 768 0 -1 0 0 -1 0 0 640`
  receive `true`

- `MoveLINHomRowWiseDirect` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$

  Only for UR3, UR5 and ABB

  This command is used to execute a linear movement. The target matrix (homogeneous coordinates, line wise, only the first three lines) is submitted. The pose is directly send to the robot and neither configuration nor possibility or accordance to set joint turn limits are checked.

  Example:

  send `MoveLINHomRowWiseDirect 0 0 -1 768 0 -1 0 0 -1 0 0 640`
  receive `true`

- `MoveStop` **Only UR-Robots**

  Stops the robot movement. In RT-mode the robot's motion is stopped immediately. Else, the robot finishes the current motion and then stops. Same as `BRAKE.`

  Example:

  send `MoveStop`
  receive `true`

4.3.6 Queries

- `GetPositionHomRowWise`

  Returns the current position in a homogeneous matrix (linewise, only the first three lines).

  Example:

  send `GetPositionHomRowWise`
  receive `0.000000 -0.173648 -0.984808 1741.140107` ↩
  `0.000000 -0.984808 0.173648 -307.009978` ↩
  `-1.000000 -0.000000 -0.000000 640.000000`

- `GetPositionJoints`

  Returns the current joint angles.

  Example:

  send `GetPositionJoints`
  receive `10.000000 0.000000 0.000000 0.000000 0.000000 0.000000`

- `GetStatus`

  Returns the current configuration. Return arguments are `flip|noflip up|down lefty|righty`

  Example:

  send `GetStatus`
  receive `noflip down lefty`

- `GetQueueLength` **Only for UR3, UR5 and ABB**

  Returns the elements in the robots PTP queue.
  Example:

  send `GetQueueLength`
  receive `8`

- `GetSpeed` **Only for UR3, UR5 and ABB**

  Returns the currently set speed, depending on the movement mode (linear or joint-space) in degree or millimeter per seconds.
  Example:

  send `GetSpeed`
  receive `20 °/s`

- `GetAccel` **Only UR-Robots**

  Returns the currently set acceleration, depending on the movement mode (linear or joint-space) in degree or millimeter per seconds square.
  Example:

  send `GetAccel`
  receive `20 °/s^2`

- `GetAccel` **Only ABB-Robots**

  Returns the currently set acceleration in percent
  Example:

  send `GetAccel`
  receive `20`

4.4 Gripper commands

- `HasGripper` **Only Adept-Robot**

  Checks whether the serial gripper is connected and running.

  Example:

  send `HasGripper`
  receive `true`

- `GripperGoHome` **Only Adept-Robot**

  Moves the gripper to the reference position (approximately 4cm open):
  Example:

  send `GripperGoHome`
  receive `State: 01000000000000010000000000000000`
  `true`

- `GripperMove amp` **Only Adept-Robot**

  Moves the gripper with `amp` ampere. Negative values close the gripper, positives open it.
  Example:

  send `GripperMove -1`
  receive `Start moving with amperage -1.000`
  `true`

- `GripperMoveToPosition pos` **Only Adept-Robot**

  Moves the gripper to the given position `pos` (opening in m).
  Example:

  send `GripperMoveToPosition 0.025`
  receive `State: 01000010010000010000000000000000`
  `true`

4.5 Additional commands

- `ForwardCalc` j_1 j_2 j_3 j_4 j_5 j_6

  Calculates the homogeneous matrix of a given joint position.
  Example:

  send `ForwardCalc 10 0 0 0 0 0`
  receive `0.000000 -0.173648 -0.984808 1741.140107` ↩
  `0.000000 -0.984808 0.173648 -307.009978` ↩
  `-1.000000 -0.000000 -0.000000 640.000000`
  `noflip up lefty`

- `BackwardCalc` $m_{1,1}$ $m_{1,2}$ $m_{1,3}$ $m_{1,4}$ $m_{2,1}$ $m_{2,2}$ $m_{2,3}$ $m_{2,4}$ $m_{3,1}$ $m_{3,2}$ $m_{3,3}$ $m_{3,4}$ ↩
  `flip|noflip up|down`

  Given a given homogeneous matrix and a corresponding configuration the joint positions are calculated.
  Example:

send	`BackwardCalc` ↩ `0.000000 -0.173648 -0.984808 1741.140107` ↩ `0.000000 -0.984808 0.173648 -307.009978` ↩ `-1.000000 -0.000000 -0.000000 640.000000` ↩ `noflip up lefty`
receive	`10.000001 -0.000055 0.000110 0.000000 -0.000055 0.000000`

- `DirectAdeptCmd MSG` **Only Adept-Robot**

  Sends `MSG` to the Adept and executes the command.
  Example:

send	`DirectAdeptCmd jmove 0,0,0,0,0,0`
receive	`true`

5 Outdated commands

Obsolete command	Replacement
`GetJointsRowWise`	`BackwardCalc`
`MovePTPHomRowWise`	`MoveMinChangeRowWiseStatus`
`MovePTPHomRowWiseStatusTurn`	`MoveMinChangeRowWiseStatus`

6 Not documented commands

`DisableRoutetest`, `DoAlterCart`, `DoAlterJoint`, `EnableRoutetest`, `GetAllowedStatus`, `GetMinChangeWeights`, `IsPossible`, `GetJointsRowWise`, `MovePTPJointsStatus`, `ResetAllowedStatus`, `ResetMinChangeWeights`, `RoutetestJoints`, `RoutetestRowWiseStatus`, `SetAllowedStatus`, `SetMinChangeWeights`, `GetRTSpeedControl`, `BRAKE`

History

1.08.1 further improvements to UR RT-mode; fixes for ABB; introduction of `KeepAlive`-Functions controlling the automatic recovery of UR robots; Introduced functions to set speed and acceleration in real world units.
1.08 improved RT-mode for UR3 and UR5; improved performance
1.07 added freedrive and blocking mode; added commands for setting payload of UR3 and UR5; rob6server can now reconnect to UR robots after connection loss
1.06 added UR3; expanded `MoveLINHomRowWiseDirect` to UR3 and UR5; added hard-RT mode for UR robots and changed behavior of `MoveRTHomRowWise` and `MoveRTJoints` in hard-RT mode; various bug fixes; new combined functions for UR3 and UR5 to reduce redundancy
1.05 introduced `MoveLINHomRowWiseDirect` for ABB
1.04 english translation, ABB with various commands added
1.03 fixes and additional commands for UR5
1.02 UR5 added
1.0b = old commands
1.0a error correction (`SetVerbosity` was not documented)
1.0 initial version