mecademic

Programming Manual

Meca500 (R3)

Robot Firmware: 7.0.3

Document Revision: A

March 14, 2018

Contents

1 Introduction **1**

2 Definitions and conventions **1**

2.1 Joint numbering 1

2.2 Reference frames 1

2.3 Joint angles 2

2.4 Pose and Euler angles 3

2.5 Joint set and robot position 4

2.6 Units 4

3 Key concepts **4**

3.1 Homing 4

3.2 Blending 5

3.3 Inverse kinematic configuration 6

3.4 Workspace and singularities 9

4 Communicating with the Meca500 over TCP/IP **10**

4.1 General information 10

4.2 Commands 11

4.2.1 Motion commands 11

4.2.2 Request commands 11

4.3 Responses 11

4.3.1 Joints and pose feedback 12

5 Motion commands **12**

5.1 Delay(t) 13

5.2 GripperOpen/GripperClose 13

5.3 MoveJoints($\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$) 13

5.4 MoveLin($x, y, z, \alpha, \beta, \gamma$) 14

5.5 MoveLinRelTRF($x, y, z, \alpha, \beta, \gamma$) 15

5.6 MoveLinRelWRF($x, y, z, \alpha, \beta, \gamma$) 15

5.7 MovePose($x, y, z, \alpha, \beta, \gamma$) 15

5.8 SetAutoConf(e) 16

5.9 SetBlending(p) . . . 16
5.10 SetCartAcc(p) . . . 17
5.11 SetCartAngVel(ω) . . . 17
5.12 SetCartLinVel(v) . . . 18
5.13 SetConf(c_1, c_3, c_5) . . . 18
5.14 SetGripperForce(p) . . . 18
5.15 SetGripperVel(p) . . . 18
5.16 SetJointAcc(p) . . . 19
5.17 SetJointVel(p) . . . 19
5.18 SetTRF($x, y, z, \alpha, \beta, \gamma$) . . . 19
5.19 SetWRF($x, y, z, \alpha, \beta, \gamma$) . . . 20

6 Request commands **20**

6.1 ActivateRobot . . . 20
6.2 ActivateSim . . . 21
6.3 ClearMotion . . . 21
6.4 DeactivateRobot . . . 21
6.5 DeactivateSim . . . 21
6.6 BrakesOn/BrakesOff . . . 22
6.7 GetConf . . . 22
6.8 GetJoints . . . 22
6.9 GetPose . . . 22
6.10 GetStatusGripper . . . 23
6.11 GetStatusRobot . . . 23
6.12 Home . . . 24
6.13 PauseMotion . . . 24
6.14 ResetError . . . 25
6.15 ResumeMotion . . . 25
6.16 SetEOB(e) . . . 25
6.17 SetEOM(e) . . . 26
6.18 SetOfflineProgramLoop(e) . . . 26
6.19 StartProgram(n) . . . 27
6.20 StartSaving(n) . . . 27

6.21 StopSaving 28

7 Error handling **28**

7.1 Command errors 28

7.2 Server errors 28

7.3 Motion errors 29

8 Status messages and command responses **31**

8.1 List of status messages 31

8.2 List of command responses 32

This page is intentionally left blank

1 Introduction

This document describes key theoretical concepts related to industrial robots, one of the communication protocols used for interfacing the Meca500 with a *network client* (PC, Mac, PLC, etc.), all the commands that can be sent to the robot, and the responses generated by the robot. Before reading this programming manual, you must first read the User Guide.

2 Definitions and conventions

You must read this section very carefully, even if you have prior experience with robot arms.

2.1 Joint numbering

The joints of the Meca500 are numbered in ascending order, starting from the base (Fig. 1).

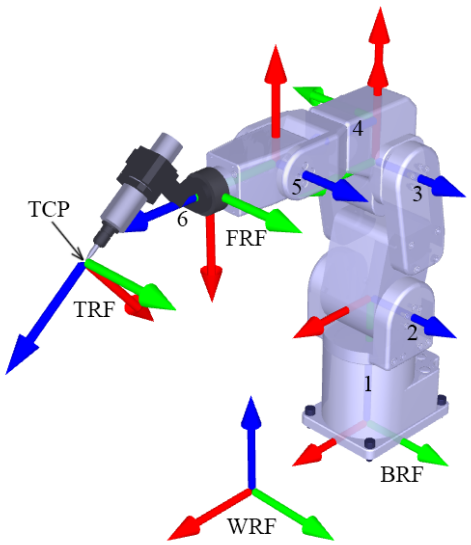


Figure 1: The joint numbering and the reference frames for the Meca500.

2.2 Reference frames

At Mecademic, we use only right-handed Cartesian coordinate systems (*reference frames*). The reference frames associated with the Meca500 are shown in Fig. 1. The x axes are in red, the y axes are in green, and the z axes are in blue. The key terms related to the reference frames that you need to be very familiar with are:

- *BRF*: *Base Reference Frame*. Static reference frame fixed to the robot base. Its z axis coincides with the axis of joint 1 and points upwards, while its origin lies on the

bottom of the robot base. The x axis of the BRF is perpendicular to the front edge of the robot base and points forward. The BRF cannot be reconfigured.

- *WRF*: *World Reference Frame*. The robot main static reference frame. By default, it coincides with the BRF. It can be reconfigured with respect to (w.r.t.) the BRF using the SetWRF command.
- *FRF*: *Flange Reference Frame*. Mobile reference frame fixed to the robot flange (*mechanical interface*). Its z axis coincides with the axis of joint 6, and points outwards. Its origin lies on the surface of the robot flange. Finally, when all joints are at zero, the x axis of the FRF points downwards.
- *TRF*: *Tool Reference Frame*. The robot end-effector's reference frame. By default, it coincides with the FRF. It can be reconfigured with respect to the TRF using the SetTRF command.
- *TCP*: *Tool Center Point*. Origin of the TRF. (Not to be confused with the Transmission Control Protocol acronym, which is also used in this document.)

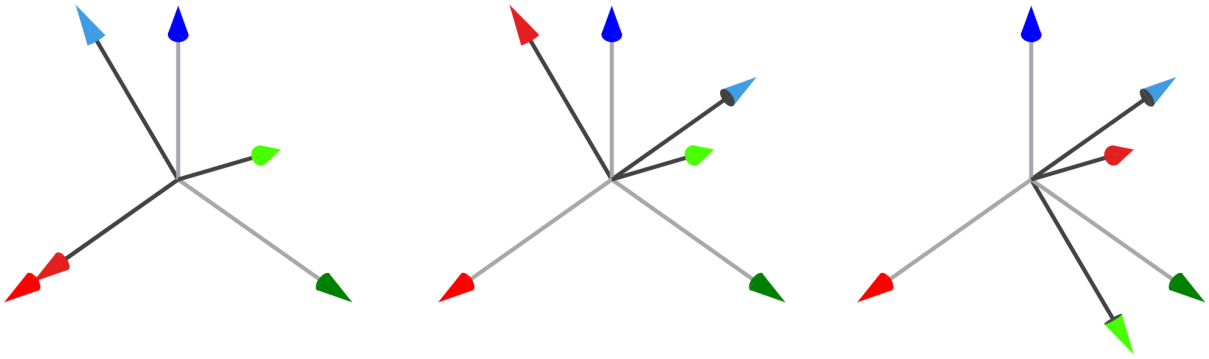
2.3 Joint angles

The angle associated with joint i ($i = 1, 2, ..., 6$), θ_i, will be referred to as *joint angle* i. Since joint 6 can rotate more than one revolution, you should think of a joint angle as a *motor angle*, rather than as the angle between two consecutive robot links. A joint angle is measured about the z axis associated with the given joint using the right-hand rule. Note that the direction of rotation for each joint is indicated on the robot body. All joint angles are zero in the configuration shown in Fig. 1. Note, however, that unless you attach an end-effector with cabling to the robot flange, there is no way of telling the value of θ_6 just by observing the robot. Thus, in Fig. 1, θ_6 might as well be equal to 360°.

The mechanical limits of the first five robot joints are as follows:

$$\begin{aligned} -175^\circ &\leq \theta_1 \leq 175^\circ, \\ -70^\circ &\leq \theta_2 \leq 90^\circ, \\ -135^\circ &\leq \theta_3 \leq 70^\circ, \\ -170^\circ &\leq \theta_4 \leq 170^\circ, \\ -115^\circ &\leq \theta_5 \leq 115^\circ. \end{aligned}$$

Joint 6 has no mechanical limits, but its (current) software limits are ±100 revolutions. Note, however, that the normal working range of joint 6 is $-180^\circ \leq \theta_6 \leq 180^\circ$, as will be explained in Section 3.3. This closed interval will be denoted as $[-180^\circ, 180^\circ]$ and also referred to as a ±180° range.



(a) rotation of 45° about the x axis (b) rotation of $-90°$ about the new y axis (c) rotation of 90° about the new z axis

Figure 2: Example showing the three consecutive rotations associated with the Euler angles $\{45°, -90°, 90°\}$.

2.4 Pose and Euler angles

Some of Meca500's commands take a *pose* (position and orientation of one reference frame with respect to another) as an input. In these commands, and in Meca500's web interface, a pose consists of a Cartesian position, $\{x, y, z\}$, and an orientation specified in *Euler angles*, $\{\alpha, \beta, \gamma\}$, according to the mobile XYZ convention. In this convention, if the orientation of a frame F_1 with respect to a frame F_0 is described by the Euler angles $\{\alpha, \beta, \gamma\}$, it means that if you align a frame F_m with frame F_0, then rotate this frame about its x axis by α degrees, then about its y axis by β degrees, and finally about its z axis by γ degrees, the final orientation of this frame F_m will be the same as that of frame F_1.

An example of specifying orientation using the mobile XYZ Euler angle convention is shown in Fig. 2. In the third image of this figure, the orientation of the black reference frame with respect to the gray reference frame is represented with the Euler angles $\{45°, -90°, 90°\}$.

It is crucial to understand that there are infinitely many Euler angles that correspond to a given orientation. For your convenience, the various motion commands that take position and Euler angles as arguments accept any values for the three Euler angles (e.g., the set $\{378°, -567°, 745°\}$). However, we output only the equivalent Euler angle set $\{\alpha, \beta, \gamma\}$, for which $-180° \leq \alpha \leq 180°$, $90° \leq \beta \leq 90°$ and $-180° \leq \gamma \leq 180°$. Furthermore, if you specify the Euler angles $\{\alpha, \pm 90°, \gamma\}$, the controller will always return an equivalent Euler angles set in which $\alpha = 0$. In other words, it is perfectly normal that the Euler angles that you have used to specify an orientation are not the same as the Euler angles returned by the controller, once that orientation has been attained. To understand why, we urge you to read our tutorial on *Euler angles*, available on our web site.

2.5 Joint set and robot position

As we will explain later, for a desired pose of the robot end-effector with respect to the robot base, there are several possible solutions for the values of the joint angles, i.e., several possible sets $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$. Thus, the best way to describe where the robot is, is by giving its set of joint angles. We will refer to this set at the *joint set*.

For example, in Fig. 1, the joint set of the robot is $\{0°, 0° 0°, 0°, 0°, 0°\}$, although, it could have been $\{0°, 0° 0°, 0°, 0°, 360°\}$, and you wouldn't be able to tell the difference from the outside.

A joint set defines completely the relative poses, i.e., the "arrangement", of the seven robot links (a six-axis serial robot is typically composed of a series of seven links, starting from the robot base and ending with the robot end-effector). We will call this arrangement the *robot position*. Do not confuse this term with the position of the robot TCP. Thus, the joint sets $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$ and $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6 + n360°\}$, where n is an integer, correspond to the same robot position. Generally (but not always), a robot position can also be defined by the pose of the TRF with respect to the WRF, the definitions of both frames, and the configuration type (to be discussed in Section 3.3). As we will explain in Section 3.4, in a so-called singular robot position, there may be infinitely many other robot positions corresponding to the same end-effector pose.

2.6 Units

We use the International System of Units (SI). Distances that are displayed to or entered by the user are in millimeters (mm) and angles in degrees (°).

3 Key concepts

3.1 Homing

At power-up, the Meca500 knows the approximate angle of each of its joints, with a couple of degrees of uncertainty. To find the exact joint angles with very high accuracy, each motor must make one full revolution. This process is the essential part of a procedure called *homing*.

Thus, during homing, all joints rotate slightly. First, all joints rotate simultaneously in one direction. Specifically, each of joints 1, 2 and 3 rotates 3.6°, joints 4 and 5 rotate 7.2° each, and joint 6 rotates 12°. Then, all joints rotate back to their initial angles. The whole back and forth motion lasts approximately 4 seconds. Make sure there is nothing that restricts the above-mentioned joint movements, or else the homing process will fail.

Finally, if your robot is equipped with Mecademic's gripper (MEGP 25), the robot controller will automatically detect it, and the homing procedure will end with a homing of the gripper. The gripper will fully open, then fully close. Make sure there is nothing that restricts the full 6-mm range of motion of the gripper, while the latter is being homed.

It is very important to understand that the absolute encoder of joint 6 works only in the range ±360°. Thus, imagine that you use an end-effector that has cabling allowing joint 6 to rotate ±365°. Then imagine that prior to activating the robot, you turn joint 6 so that the cable is almost fully stretched, e.g., $\theta_6 = 361°$. After homing, the robot will think that $\theta_6 = 1°$, which would obviously be a problem. Therefore, if you use an end-effector that restricts the movement of joint 6 (most probably because of cabling), but allows joint 6 to rotate more than 360° in one direction, always make sure that joint 6 is positioned close to its zero angle before homing.

3.2 Blending

All multi-purpose industrial robots function in a similar manner when it comes to moving around. You either ask the robot to move its end-effector to a certain pose or its joints to a certain joint set. When your target is a joint set, you have no control over the path that the robot will follow. When the target is a pose, you can either leave it to the robot to choose the path or require that the TCP follows a linear trajectory. Thus, if you need to follow a complex curve (as in a gluing application), you need to decompose your curve into multiple linear segments. Then, instead of having the robot stop at the end of each segment and make a sharp change in direction, you can blend these segments using what we call *blending*. You can think of blending as the action of taking a shortcut.

Blending is a feature that allows Meca500's trajectory planner to keep the Cartesian velocity of the robot end-effector as constant as possible between two joint-domain movements (MoveJoints, MovePose) or two Cartesian movements (MoveLin, MoveLinRelTRF, MoveLinRelTRF) in a queue. When blending is activated, the trajectory planner will transition between the two trajectories using a rounded (blended) curve. The higher the speed, the more rounded the transition will be. In other words, you cannot control directly the radius of the blending. Also, note that even if blending is enabled, the robot will also come to a full stop after a joint-mode movements followed by a Cartesian-mode movement, or vice-versa.

Figure 3 shows an example of blending. When blending is disabled, each path will begin from a full stop and end to a full stop. Blending is enabled by default. It can be disabled completely or enabled only partially with the SetBlending command.

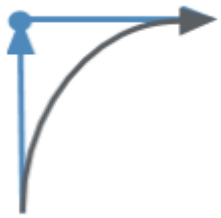


Figure 3: TCP path between two consecutive movements (in this case, two MoveLin commands). In blue, the blending is disabled. In gray, it is enabled.

3.3 Inverse kinematic configuration

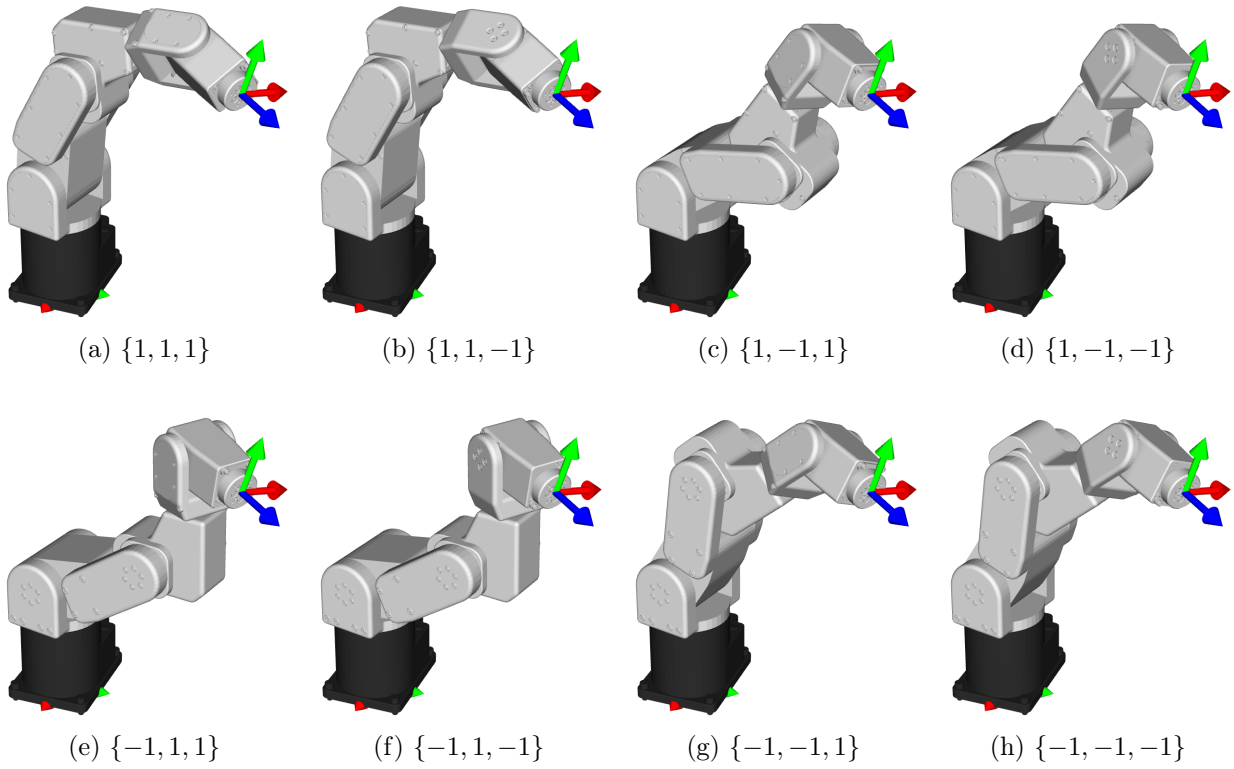
Like virtually all six-axis industrial robot arms available on the market, Meca500's inverse kinematics generally provide up to eight feasible robot positions for a desired pose of the TRF with respect to the WRF (Fig. 4), and many more joint sets (since if θ_6 is a solution, then $\theta_6 \pm n360°$, where n is an integer, is also a solution).

Each of these solutions is associated with one of eight so-called configuration types, or *configurations*, defined by three parameters: c_1, c_3 and c_5. Each parameter corresponds to a specific geometric condition on the robot position:

- c_1 :
  - $c_1 = 1$, if the *wrist center* (where the axes of joints 4, 5 and 6 intersect) is on the positive side of the yz plane of the frame associated with joint 2 (see Fig. 1). This frame is obtained by shifting the BRF upwards and rotating it about the axis of joint 1 at θ_1 degrees. (The condition $c_1 = 1$ is often referred to as "front".)
  - $c_1 = -1$, if the wrist center is on the negative side of this plane ("back" condition).
- c_3:
  - $c_3 = 1$, if $\theta_3 > \tan^{-1}(19/60) - 90° \approx -72.43°$ ("elbow up" condition)
  - $c_3 = -1$, if $\theta_3 < \tan^{-1}(19/60) - 90° \approx -72.43°$ ("elbow down" condition)
- c_5:
  - $c_5 = 1$, if $\theta_5 > 0°$ ("no flip" condition)
  - $c_5 = -1$, if $\theta_5 < 0°$ ("flip" condition)

Figure 5 shows an example of each inverse kinematics configuration parameter, as well as of the limit conditions, which are called singularities.

Note that when we solve the inverse kinematics, we only find solutions for θ_6 that are in the range $\pm 180°$. Thus, if you want to record your current robot position (e.g., after jogging the robot), you can choose between two possible approaches. One approach is to save directly the current joint set by retrieving it first with GetJoints. In this way, you don't have to worry about configurations and you will record the exact value for θ_6, which could be outside the range $\pm 180°$. However, you won't be able to move the robot to this joint set



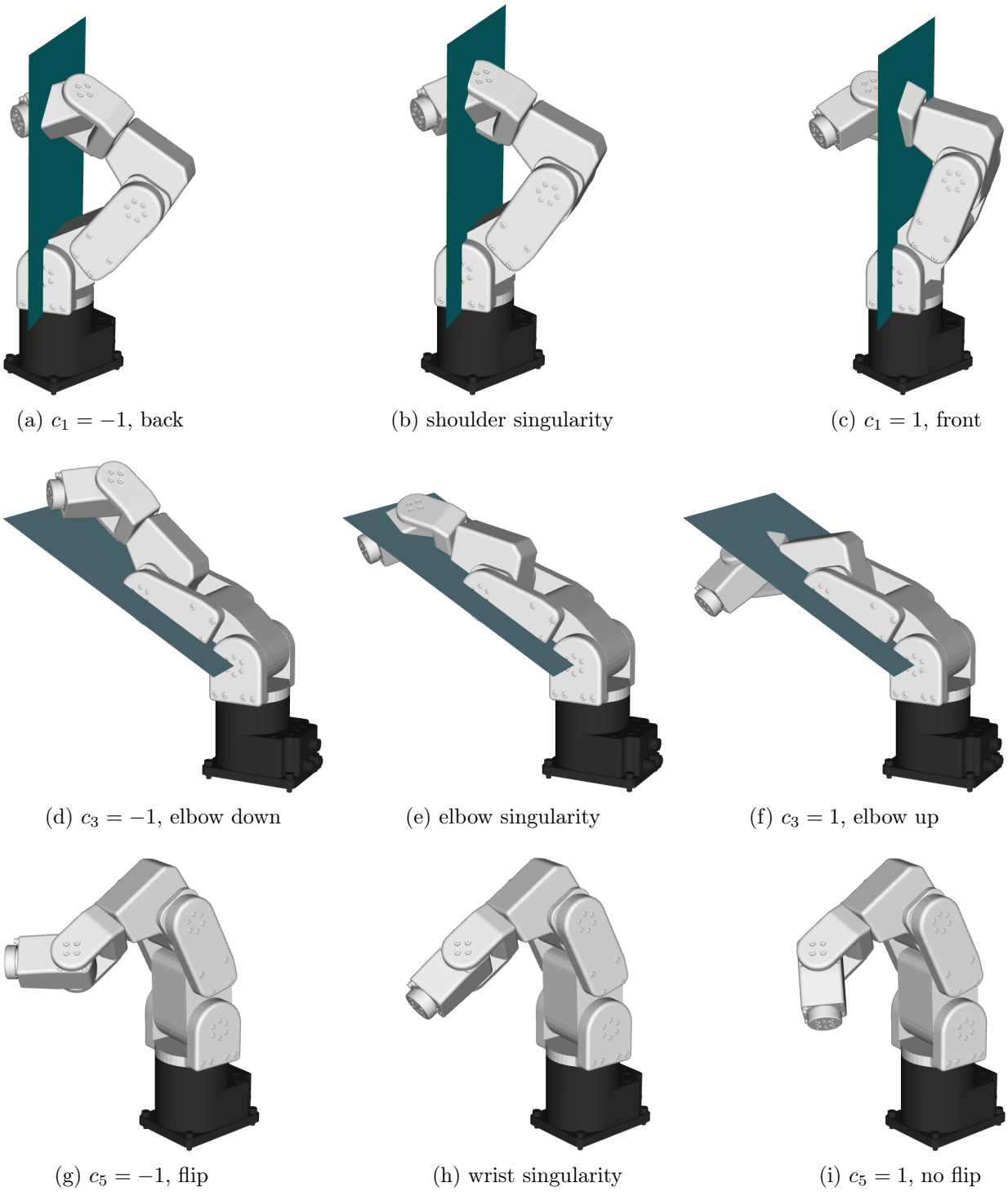
(a) $\{1, 1, 1\}$ (b) $\{1, 1, -1\}$ (c) $\{1, -1, 1\}$ (d) $\{1, -1, -1\}$

(e) $\{-1, 1, 1\}$ (f) $\{-1, 1, -1\}$ (g) $\{-1, -1, 1\}$ (h) $\{-1, -1, -1\}$

Figure 4: An example showing all eight possible configurations $\{c_1, c_3, c_5\}$ for the pose {77 mm, 210 mm, 300 mm, $-103°$, $36°$, $175°$} of the FRF with respect to the BRF.

by forcing the TCP to follow a straight line (i.e., use the MoveLin command). For example, if the recorded joint set corresponds to the robot holding a pin inserted in a hole, you won't be able to insert the pin following a linear path.

Therefore, if you want to follow a linear path to a desired robot position, you must follow a different approach. You must record the corresponding pose of the TRF (by retrieving it with GetPose) with respect to the WRF, but also the corresponding configuration (by retrieving it with GetConf). Then, when you later want to approach this robot position with MoveLin from a starting robot position, you need to make sure the robot is already in this configuration. For example, you can set the desired configuration (using SetConf) and use MovePose to get to the starting robot position. Then, you can use MoveLin to get to the target robot position. Of course, you also need to use the same TRF and WRF. The only problem with this approach is that you can only use it if $-180° \leq \theta_6 \leq 180°$ in the robot configuration of interest. Thus, for example, if you want to retrieve a prismatic peg from a prismatic hole you must absolutely start from a joint position in which $-180° \leq \theta_6 \leq 180°$.



(a) $c_1 = -1$, back (b) shoulder singularity (c) $c_1 = 1$, front

(d) $c_3 = -1$, elbow down (e) elbow singularity (f) $c_3 = 1$, elbow up

(g) $c_5 = -1$, flip (h) wrist singularity (i) $c_5 = 1$, no flip

Figure 5: Examples of the inverse kinematic configuration parameters and of the three types of singularities.

In many cases, however, you simply want to move the robot end-effector to a given pose, and do not care about the path followed by the end-effector. In these situations, you can

enable the automatic configuration selection feature (using SetAutoConf) and not worry about configurations. (In fact, this feature is activated by default.) If you simply want the TRF to move to a certain pose you can use the command MovePose and the robot will automatically select the optimal configuration for this pose (the one that corresponds to the robot position that is faster to reach, but in which $-180° \leq \theta_6 \leq 180°$).

Note, however, that when using a Cartesian-mode movement, it is impossible to change a configuration. (The command SetAutoConf has effect only on MovePose). This is because a change of configuration can only be accomplished by passing through a so-called singularity. In general, this is physically impossible while having the TCP follow a specific path (as is the case with the MoveLin, MoveLinRelTRF and MoveLinRelWRF commands).

3.4 Workspace and singularities

Many users mistakenly oversimplify the workspace of a six-axis robot arm as a sphere of radius equal to the *reach* of the robot (the maximum distance between the axis of joint 1 and the center of the wrist). The truth is that the Cartesian *workspace* of a six-axis industrial robot is a six-dimensional entity: the set of all attainable end-effector poses. Therefore, the workspace of a robot depends on the chosen TRF. Worse yet, as we saw in the preceding section, for a given end-effector pose, we can generally have eight different robot positions (Fig. 4). Thus, the Cartesian workspace of a six-axis robot arm is the combination of eight workspace subsets, one for each the eight configuration types. These eight workspace subsets have common parts, but there are also parts that belong to only one subset (i.e, there are end-effector poses accessible with only one configuration, because of joint limits). Therefore, in order to make optimal use of all attainable end-effector poses, the robot must often pass from one subset to the other. These passages are called *singularities*, and are very problematic when the robot end-effector is to follow a specific Cartesian path (e.g., a line).

Any six-axis industrial robot arm has *singularities*. However, the advantage of robot arms like the Meca500, where the axes of the last three joints intersect at one point (the center of the robot wrist), is that these singularities are very easy to describe geometrically (see Fig. 5). In other words, it is very easy to know whether a robot position is close to singularity in the case of the Meca500.

In a singular robot position, some of the joint set solutions corresponding to the pose of the TRF may coincide, or there may be infinitely many joint sets. The problem with singularities is that at a singular robot position, the robot end-effector may not move in certain directions. This is a physical blockage, not a controller problem. Thus, singularities are one type of workspace boundary (the other type occurs when a joint is at its limit, or when two links interfere mechanically).

For example, consider the Meca500 at its zero robot position (Fig. 1). At this robot position, the end-effector cannot be moved laterally (i.e., along the y axis of the BRF); it is physically blocked. Thus, singularities are not some kind of purely mathematical problem. They represent actual physical limits. That said, because of the way a robot controller is programmed, at a singular position (or at a robot position that is very close to a singularity), the robot cannot be moved in any direction using a Cartesian-mode motion command (MoveLin, MoveLinRelTRF, MoveLinRelWRF).

There are three types of singular robots positions, and these correspond to the conditions under which the configuration parameters c_1, c_3 and c_5 are not defined. The most common singular position is called *wrist singularity* and occurs when $\theta_5 = 0°$ (Fig. 5h). In this singularity, joints 5 and 6 can rotate in opposite directions at equal velocities while the end-effector remains stationary. You will run into this singularity very frequently. The second type of singularity is called *elbow singularity* (Fig. 5e). It occurs when the arm is fully stretched, i.e., when the wrist center is in one plane with the axes of joints 2 and 3. In the Meca500, this singularity occurs when $\theta_3 = -\tan^{-1}(60/19) \approx -72.43°$. You will run into this singularity when you try to reach poses that are too far from the robot base. The third type of singularity is called *shoulder singularity* (Fig. 5b). It occurs when the center of the robot wrist lies on the axis of joint 1. You will run into this singularity when you work too close to the axis of joint 1.

As already mentioned, you can never pass through a singularity using a Cartesian-mode motion command. However, you will have no problem with singularities when using the command MoveJoints, and to a certain extent, the command MovePose. Finally, you need to know that when using the commands MoveLin, MoveLinRelTRF and MoveLinRelWRF, problems occur not only when crossing a singularity, but also when passing too close to a singularity. When passing close to a wrist or shoulder singularity, some joints will move very fast (i.e., 4 and 6, in the case of a wrist singularity, and 1, 4 and 6, in the case of a shoulder singularity), even though the TCP speed is very low. Thus, you must avoid moving in the vicinity of singularities in Cartesian mode.

4 Communicating with the Meca500 over TCP/IP

4.1 General information

To operate the Meca500, the robot must be connected to a computer or a PLC over Ethernet. Commands may be sent through Mecademic's web interface, or through a custom computer program using either the TCP/IP protocol, which is detailed in the remainder of this manual, or EtherCAT, which is explained in another manual. In the case of TCP/IP, the Meca500

communicates using null-terminated ASCII strings, which are transmitted over TCP/IP. The robot default IP address is 192.168.0.100, and its default TCP/IP port is 10000. This port is referred to as the control port. Additionally, after homing, the robot will continuously send pose feedback over TCP/IP port 10001.

4.2 Commands

The Meca500 can interpret two kinds of commands: motion commands and request commands. All commands must end with the ASCII NUL character, commonly denoted as \0. Commands are not case-sensitive. However, no whitespace characters are permitted before or after a command; only inside the arguments part. Empty lines are not permitted either.

4.2.1 Motion commands

Motion commands are used to construct a trajectory for the robot. When the Meca500 receives a motion command, it places it in a queue. The command will be run once all preceding commands have been executed. The arguments for all motion commands, except SetAutoConf and SetConf, are IEEE-754 floating-point numbers, separated with commas and, optionally, spaces. The list of motion commands are presented in Section 5.

4.2.2 Request commands

Request commands are used mainly to get status updates from the robot. Once received, the robot will immediately execute the command (e.g., return the requested information). For example, if you send a MoveJoints command, followed by a GetPose command, you will not get the final pose of the TRF but an intermediate one. To get the final pose, you need to make sure that the robot has completed its movement, before sending the GetPose.

While some of these request commands make the robot move or stop (e.g., StartProgram or PauseMotion), request commands do not have a lasting effect on subsequent motions. The complete list of request commands are presented in Section 6. The format of the robot response is summarized in Section 8.2.

4.3 Responses

The Meca500 sends responses (also referred to as messages) when it encounters an error, when it receives a request command, and when its status changes. All responses from the Meca500 consist of an ASCII string in the following format:

[4-digit code][message string OR return values.]

The four-digit code indicates the type of the response:

- [1000] to [1999]: Error message due to a command
- [2000] to [2999]: Response to a command
- [3000] to [3999]: Status update message or general error

The second part of an error message [1xxx] or status update message [3xxx] will always be a string describing the error or status change. The second part of a command response [2xxx] may be a string, or a set of comma-separated return values, depending on the command.

All message strings are intended to communicate information to the user, and are subject to change at any time. Therefore, parsing the message strings with custom software is not recommended. However, the comma-separated return value formats will not change without notification. Return values are either integers, or IEEE-754 floating-point numbers with three decimal places of precision.

4.3.1 Joints and pose feedback

The Meca500 is configured to continuously send position feedback over TCP/IP port 10001. Two kinds of feedback messages are sent over this port: joint feedback and pose feedback. Joint feedback returns the angles of the robot joints (i.e., joint set). Pose feedback returns the pose of the TRF with respect to the WRF. Feedback messages are sent over TCP/IP approximately every 15 ms. To obtain faster response times, you must use EtherCAT.

The format of the responses for the joints and the pose feedback is, respectively

[2102][$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$]
[2103][$x, y, z, \alpha, \beta, \gamma$]

5 Motion commands

In the following, the motion commands are presented in alphabetical order. All motion commands have arguments, but not all have default values (e.g., the command Delay). Also, all motion commands generate an "[3012][End of block.]" message, by default, but this message can be deactivate with SetEOB. Furthermore, most motion commands may also generate an "[3004][End of movement.]" message, if this option is activated with SetEOM.

Note that the motion commands MoveLin, MoveJoints and MovePose are not executed if they cannot be completed because of workspace limitations or singularities. In contrast, the motion commands MoveLinRelTRF and MoveLinRelWRF are executed until a workspace

limit, a singularity or another problem (e.g., a collision) is encountered along the path. Finally, all motion commands can generate errors, but these are explained in Section 7.

5.1 Delay(t)

This command is used to add a time delay after a motion command. In other words, the robot completes all movements sent before the Delay command and stops temporarily. (In contrast, the PauseMotion command interrupts the motion as soon as received by the robot.)

Arguments:

- t: desired pause length in seconds

5.2 GripperOpen/GripperClose

These two commands are used to open or close the gripper, respectively. The gripper will move its fingers until the grip force reaches 40 N. You can reduce this maximum grip force with the SetGripperForce command. In addition, you can control the speed of the gripper with the SetGripperVel command.

It is very important to understand that the GripperOpen and GripperClose commands have the same behavior as a robot motion command, being executed only after the preceding motion command has been completed. Currently, however, if a robot motion command is sent after the GripperOpen or GripperClose command, the robot will start executing the motion command without waiting for the gripper to finish its action. You must therefore send a Delay command after GripperOpen and GripperClose commands.

5.3 MoveJoints($\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$)

This command makes the robot move simultaneously its joints to the specified joint set. All joint rotations start and stop simultaneously The path that the robot takes is linear in the joint space, but nonlinear in the Cartesian space. Therefore, the TCP path is not easily predictable (Fig. 6). With MoveJoints, the robot can cross singularities without a problem.

Arguments:

- θ_i: the angle of joint i, where $i = 1, 2, ..., 6$, in degrees

Note that this is the only command that can make joint 6 move outside its normal range of $\pm 180°$. With this command, you can assign values for θ_6 in the range $\pm 180,000°$.

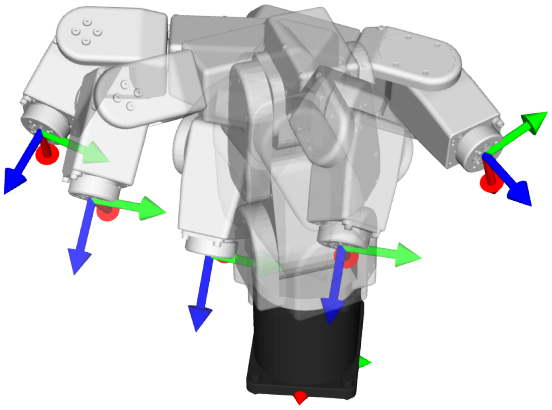


Figure 6: End-effector motion when using the MoveJoints or MovePose commands.

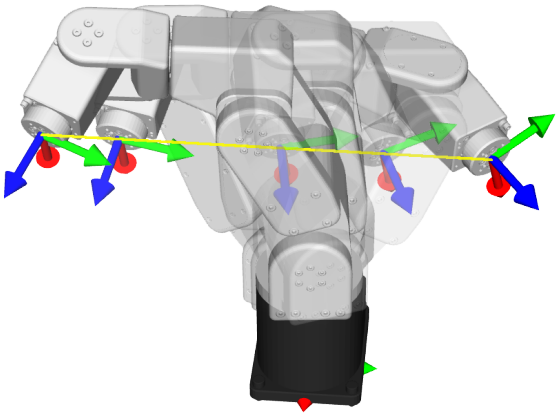


Figure 7: The TCP path when using the MoveLin command.

5.4 MoveLin($x, y, z, \alpha, \beta, \gamma$)

This command makes the robot move its TRF to a specific pose with respect to the WRF while keeping the TCP along a linear path in Cartesian space, as illustrated in Fig. 7. If the final (desired) orientation of the TRF is different from the initial orientation, the orientation will be modified along the path using SLERP interpolation.

Using this command, the robot cannot move to or through a singularity. This command cannot automatically change the robot configuration. Finally, note that this command will always move to a joint set (that corresponds to the desired pose) for which θ_6 is in the range $\pm 180°$. This restriction may lead to large rotations on joint 6.

Arguments:

- x, y, and z: the coordinates of the origin of the TRF w.r.t. the WRF, in mm
- α, β, and γ: the Euler angles representing the orientation of the TRF w.r.t. the WRF, in degrees

5.5 MoveLinRelTRF($x, y, z, \alpha, \beta, \gamma$)

This command is similar to the MoveLin command, but allows a desired pose to be specified relative to the current pose of the TRF. Thus, the arguments x, y, z, α, β, γ represent the desired pose of the TRF with respect to the current pose of the TRF(i.e., the pose of the TRF just before executing the MoveLinRelTRF command).

Arguments:

- x, y, and z: the position coordinates, in mm
- α, β, and γ: the Euler angles, in degrees

Note that this command will make the robot move to a joint set (corresponding to the desired pose) for which θ_6 is in the range $\pm 180°$. This may lead to large rotations on joint 6.

5.6 MoveLinRelWRF($x, y, z, \alpha, \beta, \gamma$)

This command is similar to the MoveLinRelTRF command, but instead of defining the desired pose with respect to the current pose of the TRF it is defined with respect to a reference frame that has the same orientation as the WRF but its origin is at the current position of the TCP. This command is mostly useful for jogging the robot.

Arguments:

- x, y, and z: the position coordinates, in mm
- α, β, and γ: the Euler angles, in degrees

Note that this command will make the robot move to a joint set (corresponding to the desired pose) for which θ_6 is in the range $\pm 180°$. This may lead to large rotations on joint 6.

5.7 MovePose($x, y, z, \alpha, \beta, \gamma$)

This command makes the robot move its TRF to a specific pose with respect to the WRF. Essentially, the robot controller calculates all possible joint sets corresponding to the desired pose and for which θ_6 is in the range $\pm 180°$, except those corresponding to a singular robot position. Then, it either chooses the joint set that corresponds to the one desired configuration or the one that is fastest to reach (see SetConf and SetAutoConf). Finally, it executes a MoveJoints commands with the chosen joint set.

Thus, all joint rotations start and stop at the same time. The path the robot takes is linear in the joint-space, but nonlinear in Cartesian space. Therefore, the path the TCP will follow to its final destination is not easily predictable, as illustrated in Fig. 6.

Using this command, the robot can cross a singularity or start from a singular robot position, as long as SetAutoConf is enabled. However, the robot cannot go to a singular configuration using MovePose. For example, assuming that the TRF coincides with the FRF, you cannot execute the command MovePose(190,0,308,0,90,0), since this pose corresponds only to singular robot positions (e.g., the joint set {0,0,0,0,0,0}). You can only use the MovePose command with a pose that corresponds to at least one non-singular robot position.

Once again, note that this command will make the robot move to a joint set (corresponding to the desired pose) for which θ_6 is in the range $\pm 180°$. This restriction may lead to large rotations on joint 6.

Arguments:

- x, y, and z: the coordinates of the origin of the TRF w.r.t. the WRF, in mm
- α, β, and γ: the Euler angles representing the orientation of the TRF w.r.t. the WRF, in degrees

5.8 SetAutoConf(e)

This command enables or disables the automatic robot configuration selection and has effect only on the MovePose command. This automatic selection allows the controller to choose the "closest" joint set corresponding to the desired pose (the arguments of the MovePose command) and for which θ_6 is in the range $\pm 180°$ (recall Section 3.3).

Arguments:

- e: enable (1) or disable (0) automatic configuration selection

Default values:

SetAutoConf is enabled by default. If you disable it, the new desired inverse kinematic configuration will be the one corresponding to the current robot position, i.e., the one after all preceding motion commands have been completed. Note, however, that if you disable the automatic robot configuration selection in a singular robot position, the controller will automatically choose one of the two, four or eight boundary configurations. For example, if you execute SetAutoCon(0) while the robot is at the joint set {0,0,0,0,0,0}, the new desired configuration will be {1,1,1}.

5.9 SetBlending(p)

This command enables/disables the robot's blending feature. Note that the commands MoveLin, MovePose, and MoveJoints will only send "[3004][End of movement.]" responses

when the robot comes to a complete stop and the command SetEOM(1) was used. Therefore, enabling blending may suppress these responses.

Also, note that there is blending only between consecutive movements executed with the commands MoveJoints and MovePose, or between consecutive movements executed with the commands MoveLin, MoveLinRelTRF and MoveLinRelTRF. In other words, there will never been blending between the paths of a MovePose command followed by a MoveLin command.

Arguments:

- p: percentage of blending, ranging from 0 (blending disabled) to 100%

Default values:

Blending is enabled at 100% by default.

5.10 SetCartAcc(p)

This command limits the Cartesian acceleration (both the linear and the angular) of the end-effector. Note that this command makes the robot come to a complete stop, even if blending is enabled.

Arguments:

- p: percentage of maximum acceleration of the end-effector, ranging from 1% to 100%

Default values:

The default end-effector acceleration limit is 100%.

5.11 SetCartAngVel(ω)

This command limits the angular velocity of the robot TRF with respect to its WRF. It only affects the movements generated by the MoveLin, MoveLinRelTRF and MoveLinRelWRF commands.

Arguments:

- ω: TRF angular velocity limit, ranging from 0.001°/s to 180°/s

Default values:

The default end-effector angular velocity limit is 45°/s.

5.12 SetCartLinVel(v)

This command limits the Cartesian linear velocity of the robot's TRF with respect to its WRF. It only affects the movements generated by the MoveLin, MoveLinRelTRF and MoveLinRelWRF commands.

Arguments:

- v: TCP velocity limit, ranging from 0.001 mm/s to 500 mm/s

Default values:

The default TCP velocity limit is 150 mm/s.

5.13 SetConf(c_1, c_3, c_5)

This command sets the desired robot inverse kinematic configuration to be observed in the MovePose command.

The robot inverse kinematic configuration (see Fig. 5) can be automatically selected by using the SetAutoConf command. Using SetConf automatically disables the automatic configuration selection.

Arguments:

- c_1: first inverse kinematics configuration parameter, either -1 or 1
- c_3: second inverse kinematics configuration parameter, either -1 or 1
- c_5: third inverse kinematics configuration parameter, either -1 or 1

5.14 SetGripperForce(p)

This command limits the grip force of the gripper.

Arguments:

- p: percentage of maximum grip force ($\sim$40 N), ranging from 0 to 100%

Default values:

By default, the grip force limit is 50%.

5.15 SetGripperVel(p)

This command limits the velocity of the gripper fingers (with respect to the gripper).

Arguments:

- p: percentage of maximum finger velocity (∼100 mm/s), ranging from 1% to 100%

Default values:

By default, the finger velocity limit is 50%.

5.16 SetJointAcc(p)

This command limits the acceleration of the joints. Note that this command makes the robot stop, even if blending is enabled.

Arguments:

- p: percentage of maximum acceleration of the joints, ranging from 1% to 100%

Default values:

The default joint acceleration limit is 100%.

5.17 SetJointVel(p)

This command limits the angular velocities of the robot joints. It affects the movements generated by the MovePose and MoveJoints commands.

Arguments:

- p: percentage of maximum joint velocities, ranging from 1 to 100%

Default values:

By default, the limit is set to 25%.

Note that it is not possible to limit the velocity of only one joint. With this command, the maximum velocities of all joints are limited proportionally. In other words, the maximum velocity of each joint will be reduced to a percentage p of its top velocity. The top velocity of joints 1 and 2 is $150°/s$, of joint 3 is $180°/s$, of joints 4 and 5 is $300°/s$, and of joint 6 is $500°/s$.

5.18 SetTRF($x, y, z, \alpha, \beta, \gamma$)

This command defines the pose of the TRF with respect to the FRF.

Arguments:

- x, y, and z: the coordinates of the origin of the TRF w.r.t. the FRF, in mm
- α, β, and γ: the Euler angles representing the orientation of the TRF w.r.t. the FRF, in degrees

Default values:

By default, the TRF coincides with the FRF.

5.19 SetWRF($x, y, z, \alpha, \beta, \gamma$)

This command defines the pose of the WRF with respect to the BRF.

Arguments:

- x, y, and z: the coordinates of the origin of the WRF w.r.t. the BRF, in mm
- α, β, and γ: the Euler angles representing the orientation of the WRF w.r.t. the BRF, in degrees

6 Request commands

In the following, the request commands are presented in alphabetical order. Most request commands do not have arguments. Note that the commands PauseMotion ClearMotion, and ResumeMotion have direct effect on the movement of the robot, but they are considered request commands, since they are executed as soon as received.

6.1 ActivateRobot

This command activates all motors and disables the brakes on joints 1, 2, and 3. It must be sent before homing is started. This command only works if the robot is idle.

Responses:

[2000][Motors activated.]
[2001][Motors already activated.]

The first response is generated if the robot was not active, while the second one is generated if the robot was already active.

6.2 ActivateSim

The Meca500 supports a simulation mode in which all of the robot's hardware functions normally, but none of the motors move. This mode allows you to test programs with the robot's hardware (i.e., hardware-in-the-loop simulation), without the risk of damaging the robot or its surroundings. Simulation mode can be activated and deactivated with the ActivateSim and DeactivateSim commands.

Responses:

[2045][The simulation mode is enabled.]

6.3 ClearMotion

This command stops the robot movement, in the same fashion as the PauseMotion command (i.e., by decelerating). However, if the robot is stopped in the middle of a trajectory, the rest of the trajectory is deleted. As is the case with PauseMotion you need to send the command ResumeMotion to make the robot ready to execute new motion commands.

Responses:

[2044][The motion was cleared.]
[3004][End of movement.]

6.4 DeactivateRobot

This command disables all motors and engages the brakes on joints 1, 2, and 3. It must not be sent while the robot is moving. Deactivating the robot while in motion could damage the joints. This command should be run before powering down the robot.

When this command is executed, the robot loses its homing. The homing process must be repeated after reactivating the robot.

Responses:

[2004][Motors deactivated.]

6.5 DeactivateSim

This command deactivates the simulation mode. See the description of the ActivateSim command, for an overview of Meca500's simulation mode.

Responses:

[2046][The simulation mode is disabled.]

6.6 BrakesOn/BrakesOff

These commands enables or disable the brakes of joints 1, 2 and 3, if and only if the robot is powered but deactivated. The command returns no message.

Responses:

[2010][All brakes set.]
[2008][All brakes released.]

6.7 GetConf

This command returns the robot current inverse kinematic configuration (see Fig. 5).

Responses:

[2029][c_1, c_3, c_5]

- c_1: first inverse kinematic configuration parameter, either -1 or 1
- c_3: second inverse kinematic configuration parameter, either -1 or 1
- c_5: third inverse kinematic configuration parameter, either -1 or 1

Note that, currently, if you request the inverse kinematic configuration, while the robot is in a robot singularity, the controller will automatically choose one of the two, four or eight boundary configurations. For example, if you execute GetConf while the robot is at the joint set {0,0,0,0,0,0}, the robot will return the configuration {1,1,1}.

6.8 GetJoints

This command returns the robot joint angles in degrees.

Responses:

[2026][$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$]

- θ_i: the angle of joint i, in degrees, where $i = 1, 2, ..., 6$

Note that the values for θ_6 returned are in the range $[-180,000°, 180,000°]$.

6.9 GetPose

This command returns the current pose of the robot TRF with respect to the WRF.

Responses:

[2027][$x, y, z, \alpha, \beta, \gamma$]

- x, y, and z: the coordinates of the origin of the TRF w.r.t. the WRF, in mm;
- α, β, and γ: the Euler angles representing the orientation of the TRF w.r.t. the WRF, in degrees.

6.10 GetStatusGripper

This command returns the gripper's status.

Responses:

[2079][ge, hs, ph, lr, es, fo]

- ge: gripper enabled, i.e., present (0 for disabled, 1 for enabled)
- hs: homing state (0 for homing not performed, 1 for homing performed)
- ph: holding part (0 if the gripper does not hold a part, 1 otherwise)
- lr: limit reached (0 if the fingers are not fully open or closed, 1 otherwise)
- es: error state (0 for absence of error, 1 for presence of error)
- fo: force overload (0 if there is no overload, 1 if the gripper is in force overload)

6.11 GetStatusRobot

This command returns the status of the robot.

Responses:

[2007][$as, hs, sm, es, pm, eob, eom$]

- as: activation state (0 for not activated, 1 for activated)
- hs: homing state (0 for homing not performed, 1 for homing performed)
- sm: simulation mode (0 for simulation mode disabled, 1 for simulation mode enabled)
- es: error status (0 for robot not in error mode, 1 for robot in error mode)
- pm: pause motion status (0 for robot not in pause motion, 1 for robot in pause motion)
- eob: end of block status (0 for end of block disabled, 1 if enabled)
- eom: end of movement status (0 for end of movement disabled, 1 if enabled)

Note that $pm = 1$ if and only if a PauseMotion or a ClearMotion was sent, or if the robot is in error mode.

6.12 Home

This command starts the robot and gripper homing process (Section 3.1). While homing, it is critical to remove any obstacles that could hinder the robot and gripper movements. This command takes about four seconds to execute.

Responses:

[2002][Homing done.]
[2003][Homing already done.]
[1014][Homing failed.]

The first response is sent if homing was completed successfully, while the second one is sent if the robot is already homed. The third response is sent if the homing procedure failed.

6.13 PauseMotion

This command stops the robot movement. The command is executed as soon as received (within approximately 5 ms from it being sent, depending on your network configuration), but the robot stops by decelerating, and not by applying the brakes. For example, if a MoveLin command is currently being executed when the PauseMotion command is received, the robot TCP will stop somewhere along the linear path. If you want to know where exactly did the robot stop, you can use the GetPose or GetJoints commands.

Strictly speaking, the PauseMotion command pauses the robot motion; the rest of the trajectory is not deleted and can be resumed with the ResumeMotion command. The PauseMotion command is useful if you develop your own HMI and need to implement a pause button. It can also be useful if you suddenly have a problem with your tool (e.g., while the robot is applying an adhesive, the reservoir becomes empty).

The PauseMotion command normally generates the following two responses. The first one is always sent, whereas the second one is sent only if the robot was moving when it received the PauseMotion command.

Finally, if a motion error occurs while the robot is at pause (e.g., if another moving body hits the robot), the motion is cleared and can no longer be resumed.

Responses:

[2042][Motion paused.]
[3004][End of movement.]

6.14 ResetError

This command resets the robot error status. The command can generate one of the following two responses. The first response is generated if the robot was indeed in an error mode, while the second one is sent if the robot was not in error mode.

Responses:

[2005][The error was reset.]
[2006][There was no error to reset.]

6.15 ResumeMotion

This command resumes the robot movement, if it was previously paused with the command PauseMotion. More precisely, the robot TCP resumes the rest of the trajectory from the pose where it was brought to a stop (after deceleration).

This command will not work if the robot was deactivated after the last time the command PauseMotion was used, if the robot is in an error mode, or if the rest of the trajectory was deleted using the ClearMotion command.

Note that it is not possible to pause the motion along a trajectory, have the end-effector move away, then have it come back, and finally resume the trajectory. If you send motion commands while the robot is paused, they will simply be placed in the queue.

Responses:

[2043][Motion resumed.]

6.16 SetEOB(e)

When the robot completes a motion command or a block of motion commands, it can send the message "[3012][End of block.]". This means that there are no more motion commands in the queue and the robot velocity is zero. The user could enable or disable this message by sending this command.

Arguments:

- e: enable (1) or disable (0) message

Default values:

By default, the end-of-block message is enabled.

Responses:

[2054][End of block is enabled.]
[2055][End of block is disabled.]

6.17 SetEOM(e)

The robot can also send the message "[3004][End of movement.]" as soon as the robot velocity becomes zero. This can happen after the commands MoveJoints, MovePose, MoveLin, MoveLinRelTRF, MoveLinRelWRF, PauseMotion and ClearMotion commands, as well as after the SetCartAcc and SetJointAcc commands. Recall, however, that is blending is enabled (even only partially), then there would be no end-of-movement message between two consecutive Cartesian-mode commands (MoveLin, MoveLinRelTRF, MoveLinRelWRF) or two consecutive joint-mode commands (MoveJoints, MovePose).

Arguments:

- e: enable (1) or disable (0) message

Default values:

By default, the end-of-movement message is disabled.

Responses:

[2052][End of movement is enabled.]
[2053][End of movement is disabled.]

6.18 SetOfflineProgramLoop(e)

This command is used to define whether the program that is to be saved must later be executed a single time or infinitely many times.

Arguments:

- e: enable (1) or disable (0) the loop execution

Default values:

By default, looping is disabled.

Responses:

[1022][Robot was not saving the program.]

This command does not generate an immediate response. It is only when saving a program, that a message indicates whether loop execution was enabled or disabled. However, if the command is sent while no program is being saved, the above message is returned.

6.19 StartProgram(n)

To start the program that has been previously saved in the robot memory, the robot must be activated prior to using the StartProgram command. The number of times the program will be executed is defined with the SetOfflineProgramLoop command. Note that you can either use this command, or simply press the Start/Stop button on the robot base (provided that no one is connected to the robot). However, pressing the Start/Stop button on the robot base will only start program 1.

Arguments:

- n: program number, where $n \leq 500$ (maximum number of programs that can be stored)

Responses:

[2063][Offline program started.]
[3017][No offline program saved.]

6.20 StartSaving(n)

The Meca500 is equipped with several membrane buttons on its base, of which a Start/Pause button. These can be used to run a simple program (possibly in loop), when no external device is connected to the robot. For example, this feature is particularly useful for running demos. To distinguish this program, which will reside in the memory of the robot controller, from any other programs that will be sent to the robot and executed line by line, it will be referred to as an *offline program*.

To save such an offline program, the robot must be deactivated first. Then you need to use the StartSaving and StopSaving commands. Note that the program will remain in the robot internal memory even after disconnecting the power. To clear a specific program, simply overwrite a new program by using the StartSaving command with the same argument.

The StartSaving command starts the recoding of all subsequent motion commands, as long as the robot is deactivated. Once the robot receives this command, it will generate the first of the two responses given bellow and start waiting for the command(s) to save. If the robot receives commands that are not of motion type (GetJoints, GetPose, etc.), it will generate the second response and clear the whole offline program. Since the robot must be deactivated during the saving process, it will not enter error mode

Note that the maximum number of motion commands that can be saved is 13,000.

Arguments:

- n: program number, where $n \leq 500$ (maximum number of programs that can be stored)

Responses:

[2060][Start saving program.]
[1023][Ignoring command for offline mode. - Command: "..."]

6.21 StopSaving

This command will make the controller save the program and end the saving process. Two responses will be generated: the first and the second or third of the three responses given bellow. Finally, if you send this command while the robot is not saving a program, the fourth response will be returned.

Responses:

[2061][n commands saved.]
[2064][Offline program looping is enabled.]
[2065][Offline program looping is disabled.]
[1022][Robot was not saving the program.]

7 Error handling

7.1 Command errors

When the Meca500 encounters an error while executing a command, it goes into error mode. All pending and future commands are canceled. The robot stops and does not accept subsequent commands until it receives a ResetError command. Table 1 lists all possible error messages that the robot may send.

7.2 Server errors

If a user tries to connect to Meca500's control port while another user is already connected, the robot will respond with the following message:

[3001][Another user is already connected, closing connection.]

If a user tries to connect to Meca500's control port and the robot has encountered an initialization problem, the robot will respond with the following message and disconnect:

[3009][Robot initialization failed due to an internal error. Restart the robot.]

7.3 Motion errors

Motion errors can occur for several reasons:

- the robot is improperly installed;
- the robot has entered into a collision;
- the position error of a joint is too large (may indicate a collision);
- there is a hardware problem with a joint.

In all cases, the robot will send the following message and go into error mode (i.e., you will have to send the ResetError command and possibly power-cycle the robot):

[3005][Error of motion.]

The list of messages and corresponding explanations is given in the following table.

Table 1: List of error messages

Message	**Explanation**
[1000][Command buffer is full.]	Maximum number of queued commands reached. Retry by sending fewer commands at a time.
[1001][Empty command or command unrecognized. - Command: "..."]	Unknown or empty command.
[1002][Syntax error, symbol missing. - Command: "..."]	A parenthesis or a comma has been omitted.
[1003][Argument error. - Command: "..."]	Wrong number of arguments or invalid input (e.g., the argument is out of range).
[1005][The robot is not activated.]	Robot is not activated. Send the ActivateRobot command.
[1006][The robot is not homed.]	The robot is not homed. Send the Home command.
[1007][Joint over limit. - Command: "..."]	The robot cannot reach the joint set or pose requested because of its joint limits.
[1011][The robot is in error.]	A command has been sent but the robot is in error mode and cannot process it until a ResetError command is sent.
[1012][Singularity detected.]	The MoveLin command sent requires that the robot pass through a singularity, or the MovePose command sent requires that the robot goes to a singular robot position.
[1013][Activation failed.]	Activation failed. Try again.

Table 1: (continued)

Message	Explanation
[1014][Homing failed.]	Homing procedure failed. Try again.
[1016][Pose out of reach.]	The pose requested in the MoveLin or MovePose commands cannot be reached by the robot (even if the robot had no joint limits).
[1017][Communication failed. - Command: "..."]	Problem with communication.
[1018]['\0' missing. - Command: "..."]	Missing NULL character at the end of a command.
[1020][Brakes cannot be released.]	Something is wrong. The brakes cannot be released. Try again.
[1021][Deactivation failed. - Command: "..."]	Something is wrong. The deactivation failed. Try again.
[1022][Robot was not saving the program.]	The command StopSaving was sent, but the robot was not saving a program.
[1023][Ignoring command for offline mode. - Command: "..."]	The command cannot be executed in the offline program.
[1024][Mastering needed. - Command: "..."]	Somehow, mastering was lost. Contact Mecademic.
[1025][Impossible to reset the error. Please, power-cycle the robot.]	Turn off the robot, then turn it back on in order to reset the error.
[1026][Deactivation needed to execute the command. - Command: "..."]	The robot must be deactivated in order to execute this command.
[1027][Simulation mode can only be enabled/disabled while the robot is deactivated.]	The robot must be deactivated in order to execute this command.
[1028][Network error.]	Error on the network. Resend the command.
[1029][Offline program full. Maximum program size is 13,000 commands. Saving stopped.]	Memory full.
[1038][No gripper connected.]	No gripper was detected.

8 Status messages and command responses

8.1 List of status messages

The status messages can occur without specific action from the network client. These could contain general status (e.g., when the robot has ended its motion) or error messages (e.g., error of motion if the robot entered in collision). Table 2 lists all possible status messages.

Table 2: List of all status messages

Message	**Explanation**
[3000][Connected to Meca500 x_x_x.x.x.]	Confirms connection to robot.
[3001][Another user is already connected, closing connection.]	Another user is already connected to the Meca500. The robot disconnects from the user immediately after sending this message.
[3003][Command has reached the maximum length.]	Too many characters before the NUL character. Almost certainly means that a NUL character was forgotten.
[3004][End of movement.]	The robot has stopped moving.
[3005][Error of motion.]	Motion error. Probably caused by a physical interference.
[3009][Robot initialization failed due to an internal error. Restart the robot.]	Error in robot startup procedure. Contact Mecademic if restarting the Meca500 did not resolve the issue.
[3012][End of block.]	No motion command in queue and robot velocity is zero.
[3013][End of offline program.]	The offline program has finished.
[3014][Problem with saved program, save a new program.]	There was a problem saving the program.
[3016][Ignoring command while in offline mode.]	A non-motion command was sent while executing a program and was ignored.
[3017][No offline program saved.]	There is no program in memory.
[3018][Loop ended. Restarting the program.]	The offline program is being restarted.
[3026][Robot's maintenance check has discovered a problem. Mecademic cannot guaranty correct movements. Please contact Mecademic.]	A problem was detected. Contact Mecademic.

8.2 List of command responses

Table 3 presents a summary of all motion and request commands and the possible responses for each of them.

Table 3: List of all possible command responses

Command	**Possible response(s)**
ActivateRobot	[2000][Motors activated.] [2001][Motors already activated.]
ActivateSim	[2045][The simulation mode is enabled.]
ClearMotion	[2044][The motion was cleared.]
DeactivateRobot	[2004][Motors deactivated.]
DeactivateSim	[2046][The simulation mode is disabled.]
Delay	[3012][End of block.]
BrakesOn	[2008][All brakes released.]
BrakesOff	[2010][All brakes set.]
GetConf	[2029][c_1, c_3, c_5]
GetJoints	[2026][$\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$]
GetStatusRobot	[2007][$as, hs, sm, es, pm, eob, eom$]
GetStatusGripper	[2079][ge, hs, ph, lr, es, fo]
GetPose	[2027][$x, y, z, \alpha, \beta, \gamma$]
GripperClose	[3012][End of block.]
GripperOpen	[3012][End of block.]
Home	[2002][Homing done.] [2003][Homing already done.]
MoveJoints	[3004][End of movement.] [3012][End of block.]
MoveLin	[3004][End of movement.] [3012][End of block.]
MoveLinRelTRF	[3004][End of movement.] [3012][End of block.]
MoveLinRelWRF	[3004][End of movement.] [3012][End of block.]
MovePose	[3004][End of movement.] [3012][End of block.]
PauseMotion	[2042][Motion paused.] [3004][End of movement.]
ResetError	[2005][The error was reset.] [2006][There was no error to reset.]

Table 3: (continued)

Command	**Possible response(s)**
ResumeMotion	[2043][Motion resumed.]
SetAutoConf	[3004][End of movement.] [3012][End of block.]
SetCartAcc	[3004][End of movement.] [3012][End of block.]
SetCartLinVel	[3012][End of block.]
SetCartAngVel	[3012][End of block.]
SetConf	[3012][End of block.]
SetBlending	[3012][End of block.]
SetGripperVel	[3012][End of block.]
SetJointAcc	[3004][End of movement.] [3012][End of block.]
SetJointVel	[3012][End of block.]
SetEOB	[2054][End of block is enabled.] [2055][End of block is disabled.]
SetEOM	[3004][End of movement.] [3012][End of block.]
SetOfflineProgramLoop	[1022][Robot was not saving the program.]
SetTRF	[3004][End of movement.] [3012][End of block.]
SetWRF	[3004][End of movement.] [3012][End of block.]
StartProgram	[3004][End of movement.]
StartSaving	[2060][Start saving program.] [2064][Offline program looping is enabled.] [2065][Offline program looping is disabled.]
StopSaving	[2061][n commands saved.] [2064][Offline program looping is enabled.] [2065][Offline program looping is disabled.]

Mecademic Inc.
1390 Rosemont Blvd
Montreal QC H2G 1V4
CANADA