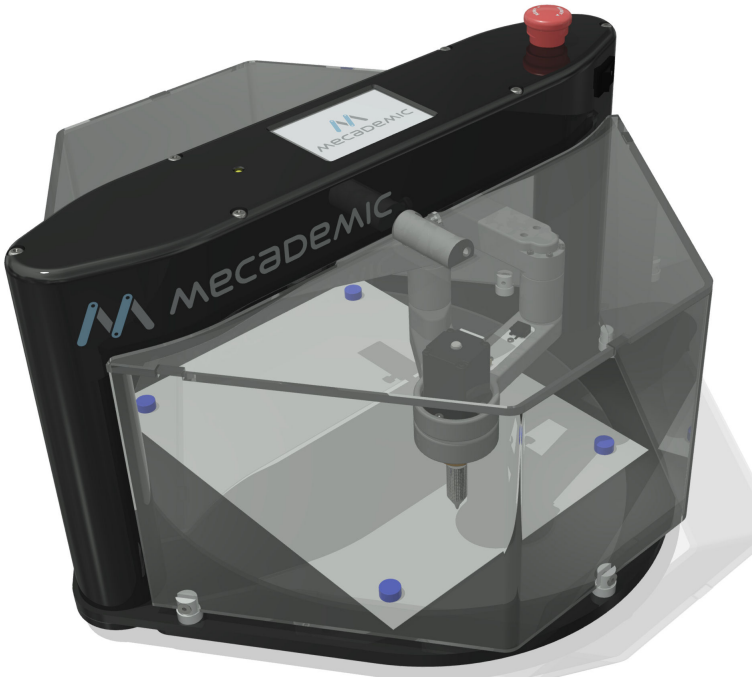
DexTAR



User's Manual

Version 1.0

Contents

Chapter 1

Introduction

DexTAR stands for *Dextrous Twin-Arm Robot*, but dexterity is not what makes DexTAR one of the best robot manipulators for pedagogical purposes. DexTAR is by far the most illustrative example of a *parallel robot**. Understanding how DexTAR functions requires mostly basic knowledge of trigonometry and vector algebra. But robots like DexTAR are also the subject of academic works, because there is so much to learn from studying them. Yet DexTAR is not some kind of academic toy. Robots very similar to DexTAR are used in industry, and DexTAR has nothing to envy them in terms of speed and accuracy.

1.1 The Origins of DexTAR

DexTAR (Fig. 1.1) consists of a so-called *five-bar* planar mechanism, sometimes called a *pantograph*, the base joints of which are actuated and swivel the *proximal links*, and a linear actuator mounted on one of the two *distal links*, which moves normal to the plane of the five-bar. DexTAR is therefore a *3-degree-of-freedom* (3-DOF) parallel robot.

Figure 1.1: Mecademic's DexTAR with its electromagnetic end-effector.

*For more information, visit www.mecademic.com/What-is-a-parallel-robot.html.

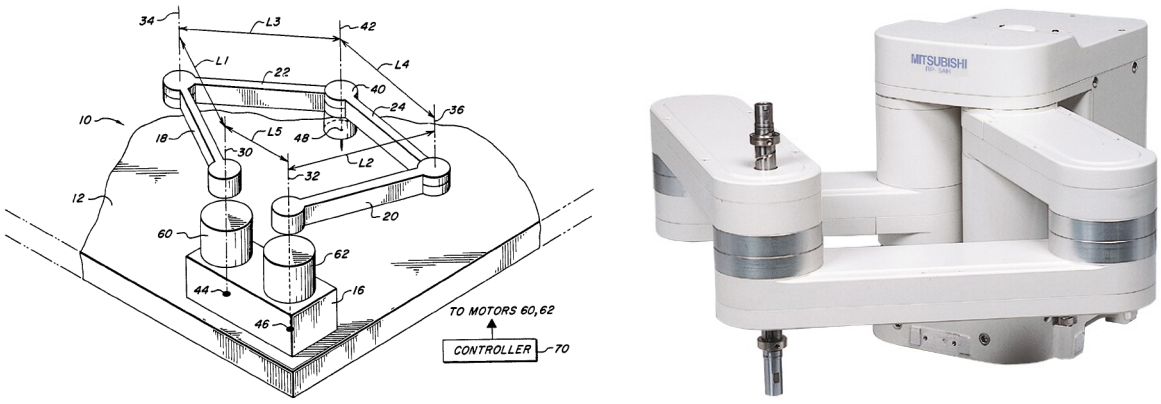
(a) (b) (c)

Figure 1.2: (a) A schematic of the very first industrial robot design, proposed in 1934 [1], (b) a photo of one of the first SCARA robots, built by Prof. Hiroshi Makino in 1980, and (c) a schematic of the dual-arm SCARA robot invented by Prof. Makino in 1979 [4].

DexTAR's *end-effector* is mounted at the lower end of its linear actuator. Two different end-effectors are supplied with our DexTAR: an electromagnet for picking up small (⌀11 mm) ferromagnetic balls (Fig. 1.1) and a ballpoint pen. Various other end-effectors can be custom designed (e.g., a printing head or a suction cup), but you should know that DexTAR cannot control the orientation of its end-effector about the vertical axis, and that this orientation is not constant. In other words, you cannot use DexTAR to assemble LEGO® bricks, for example. Therefore, in this manual, we will only be interested in the x, y and z coordinates of the so-called *tool center point* (*TCP*) of the end-effector (e.g., the center of the steel ball when picked by the electromagnet or the tip of the ballpoint pen).

Five-bar robots similar to DexTAR have existed for at least several decades. Probably the first such pantograph robot, shown in Fig. 1.2(a), was described in a US patent in 1934 [1]. In fact, this pantograph is the very first *industrial robot* design (i.e., a robot for industrial use, with a controller). Much later, in 1978, Prof. Hiroshi Makino from the University of Yamanashi (Japan) invented the now hugely popular *SCARA* (serial) robot [2], one of the first prototypes of which is shown in Fig. 1.2(b). Indeed, in 2013 alone, more than 18,000 SCARA serial robots were installed around the world [3].

The SCARA robot has essentially the same motion characteristics as DexTAR, though most frequently it also offers a fourth DOF (rotation about vertical axes). Despite the popularity of SCARA serial robots, parallel SCARA robots like DexTAR have an unquestionable advantage. Indeed, less than two years after developing the first SCARA serial robot, Prof. Makino invented an improved design, shown in Fig. 1.2(c), featuring two arms [4]. The advantage of that parallel robot is that two of its rotary motors are fixed at the base, which results in lighter moving parts. Then, in 1985, Donald C. Fyler, from the Charles Stark Draper Laboratory (USA), came up independently with the idea of using a five-bar mechanism, shown in Fig. 1.3(a), as a robot [5]. He too claimed that the dual-arm robot was a better alternative to the SCARA serial robot, but never constructed his parallel robot.



(a) (b)

Figure 1.3: (a) A schematic of the five-bar robot proposed in US patent No. 4,712,971 [5] and (b) a photo of the RP-5AH industrial robot from Mitsubishi Electric.

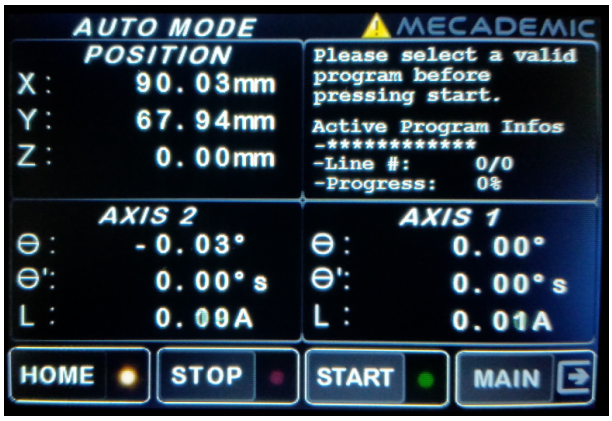
The first company to actually construct and commercialize the dual-arm SCARA robot was Mitsubishi Electric. The company began marketing MELFA RP-1A, the smallest of now three different models of which the largest one is shown in Fig. 1.3(b), in 1998. The robot was designed by Mr. Norio Kodaira, currently president of the Robotics Society of Japan and head of Mitsubishi Electric's robotics technology division. According to Mitsubishi Electric, the robot's high speed and superior accuracy make it ideal for submicron semiconductor processes. Other proposed applications include assembly processes such as mounting, soldering, and painting. Annual production of MELFA RP-1A was set at 1,200 units from the very first year.

The dual-arm SCARA has been popular in academia for more than four decades [6, 7] and is currently a favorite design in the makers community. Probably the most famous such example is the quirky Plotclock (youtu.be/iOLFP90DneY), designed by German programmer Johannes Heberlein, which writes time on a little white board every minute and then erases it. More conventionally, five-bar mechanisms are used as 3D printers and haptic devices. We constantly share new such prototypes on our Twitter account (@mecademic).

DexTAR, however, is very different from all other dual-arm SCARA robots. All of its four links have the same length, which makes its *workspace* free of any holes. Furthermore, in DexTAR, mechanical interferences between the arms have been reduced to a minimum. Finally, by sort of flipping its arms, DexTAR gains access to a very large workspace (the area of the convex lens shaped plate with holes in Fig. 1.1), not as large as that of a one-armed SCARA robot, but larger than the workspace of any other similarly sized parallel SCARA. In some configurations, for example, a base motor can rotate infinitely. All of these features make DexTAR much more fascinating than an ordinary dual-arm SCARA robot.

The very first DexTAR was developed by Prof. Ilian Bonev's team at the École de technologie supérieure, in Montreal, Canada (youtu.be/dnixuCu49o4) [8]. This research prototype is actuated with huge direct-drive motors and a pneumatic linear actuator, and its controller is implemented on a PC running MathWorks' xPC Target™. Our DexTAR is much more compact and with all-electric actuation. It uses two 90 W maxon® servomotors and one stepper motor linear actuator, and its controller is inside the robot's base.

(a) Main menu



(b) Auto Mode menu

Figure 1.4: The most frequently seen screens on DexTAR's touchscreen.

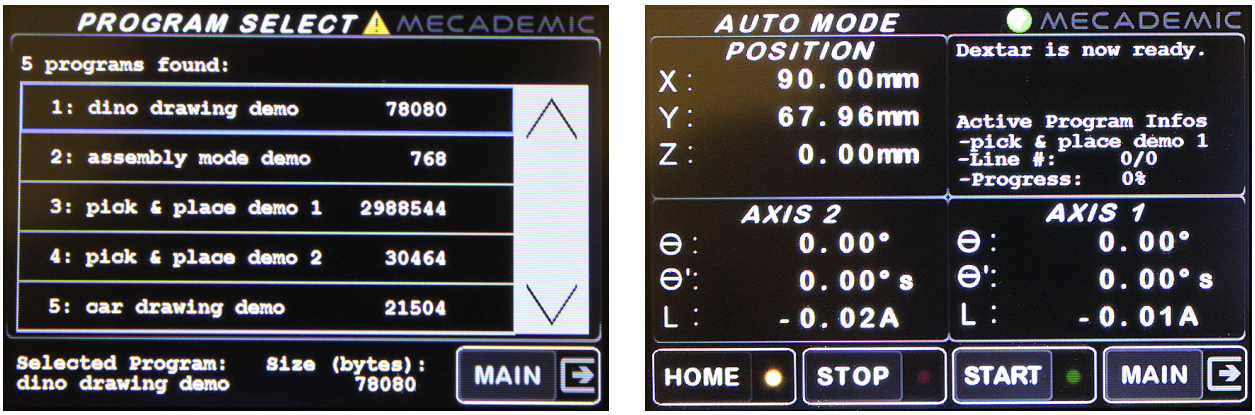
1.2 Getting Started with DexTAR

All right, we know that you are eager to try your new DexTAR, so let's get started. Put your robot on a very solid horizontal work surface. The robot is shipped with the electromagnetic tool already installed and with the lens-shaped perforated acrylic plate in place. Make sure the four steel pins underneath the plate are well inserted into the holes of the robot's base, where neodymium magnets will keep the plate fixed. Then, place nine of the ⌀11 mm balls in the holes of the acrylic plate in a 3×3 pattern, in the very front portion of the plate. Next, move the robot arms manually so that the end-effector is close to the front of the robot, as in Fig. 1.1. Make sure the angle between the distal links is between 30° and 150°! Finally, carefully place both safety covers, make sure the emergency stop (E-stop) button is reset (twist it clockwise) and, of course, that the power supply is connected.

⚠ Although DexTAR is intrinsically safe, you must always keep both safety covers installed when the yellow LED on DexTAR is lit. Not using the safety covers will expose you to possible hazards such as impacts or pinching.

Now, turn on the robot using the on/off switch on DexTAR's body. After a couple of seconds, the user interface shown in Fig. 1.4(a) appears on the robot's LCD touchscreen and the yellow light on DexTAR's body is illuminated. The robot's motors are now active. Remember that each time you power DexTAR, you need to make it go to its home configuration. To home the robot, press the Auto Mode icon on the touchscreen. This will bring you to the screen shown in Fig. 1.4(b). Press the HOME button, and DexTAR's arms and end-effector will move, but the latter must remain in the front portion. Otherwise, turn the robot off and repeat the procedure described in this paragraph.

Now, as the updated Auto Mode menu says, you need to load a valid program. We've preloaded a few sample programs, some for the electromagnetic tool, others for the ballpoint pen tool. Let's load the one that manipulates the nine balls that you have already placed. Press the MAIN button in the bottom right corner to get back to the main menu. Then,



(a) Program selection menu (b) Auto Mode menu

Figure 1.5: After these two steps, you will be ready to execute a program.

press the Select Program icon. A list of preloaded programs will show up, as in Fig. 1.5(a). Select the one that is called "pick & place demo 1," and then press the MAIN button in the bottom right corner to get back to the main menu. Note that the yellow warning symbol (⚠) on top of the touchscreen is now replaced by a green ✓ symbol (). This means that DexTAR is ready to run a program. Select again the Auto Mode menu, and you will see a confirmation that the robot is "now ready," as shown in Fig. 1.5(b). Note that you can also first load a program and then home the robot.

Well, all that's left to do is hit the START button in the Auto Mode menu and watch DexTAR move the balls. You are supposed to see this demo: youtu.be/jYHawaGL90o. Yes, DexTAR moves swiftly; much faster than in that video on YouTube. In fact, never leave DexTAR unattended as it slides gradually and might fall off your table!

You may rerun the program by pressing START again, but you probably won't understand the magic behind DexTAR's moves. Indeed, the way DexTAR moves to bring its TCP to a given position is somewhat similar to solving a labyrinth, or one of those wire puzzles.

Now, before you continue reading this manual, we strongly suggest that you browse to www.mecademic.com/DexTAR.html and scroll down to the blue and gray image (Fig. 1.6). This, in fact, is a JavaScript applet simulating DexTAR. With your mouse (you do need to use a computer), start dragging the robot's TCP and the two proximal links. Until you don't collect at least four of the five yellow circles, don't read any further.

You've hopefully noticed that for a given TCP position, there are several possible arm arrangements. In theory, there are four possible arrangements, but because of mechanical interferences, for most TCP positions, there are less than four such *configurations*. These four types of configurations are called *working modes* $-+$, $++$, $+-$, and $--$, where each sign corresponds to the sign of the angle between the distal link and the proximal link in the corresponding arm, measured in the counterclockwise sense in the range $[-180°, 180°]$.

Figure 1.7 shows a schematic of DexTAR in one of the rare TCP positions for which all four working modes are feasible. In each of the subfigures, the blue area is the set of TCP positions (i.e., the workspace) for the given working mode. DexTAR can change working modes by simply flipping one or both of its arms. This is fairly simple from a control point

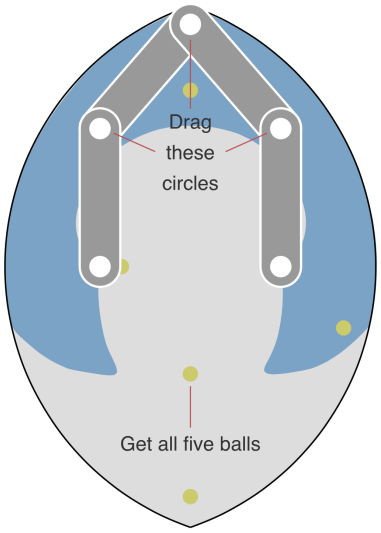
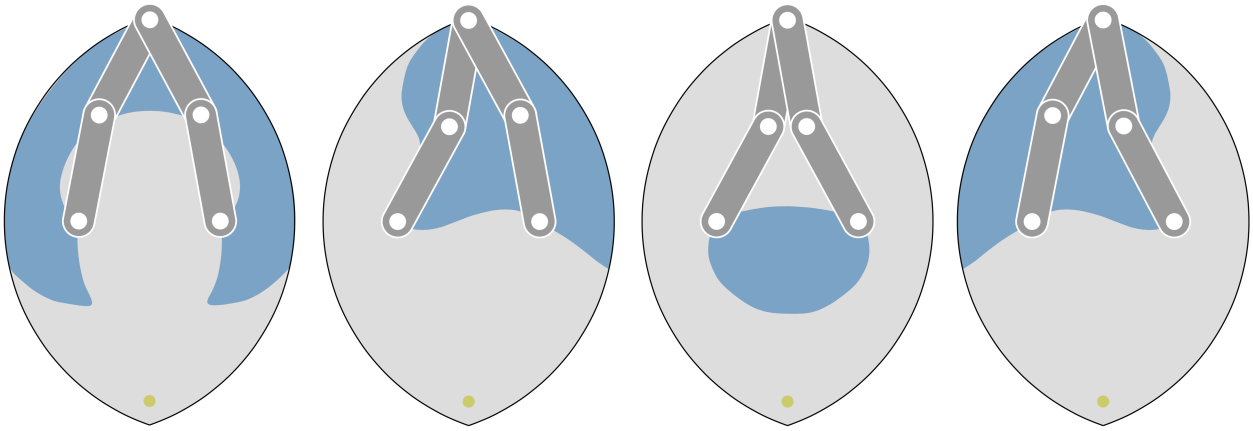


Figure 1.6: A screen capture of the JavaScript applet on Mecademic's web site that lets you understand how DexTAR works.



(a) working mode −+ (b) working mode ++ (c) working mode +− (d) working mode −−

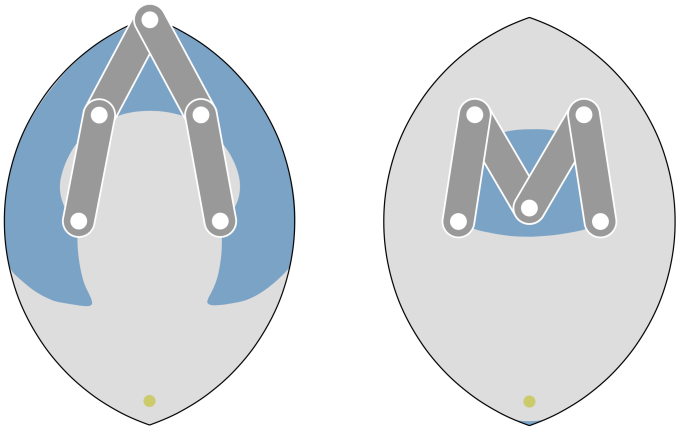
Figure 1.7: A schematic of DexTAR in each of its four working modes.

of view. Thus, DexTAR's workspace is the combination of the four workspace zones shown in Fig. 1.7. But then, how do we get the fifth ball (the yellow circle in Fig. 1.7)?

Similarly, for given base motor angles, there are generally two possible TCP positions (Fig. 1.8). The two possible configurations for the distal links are called *assembly modes* + and −, where the sign is related to the angle between the two distal links.

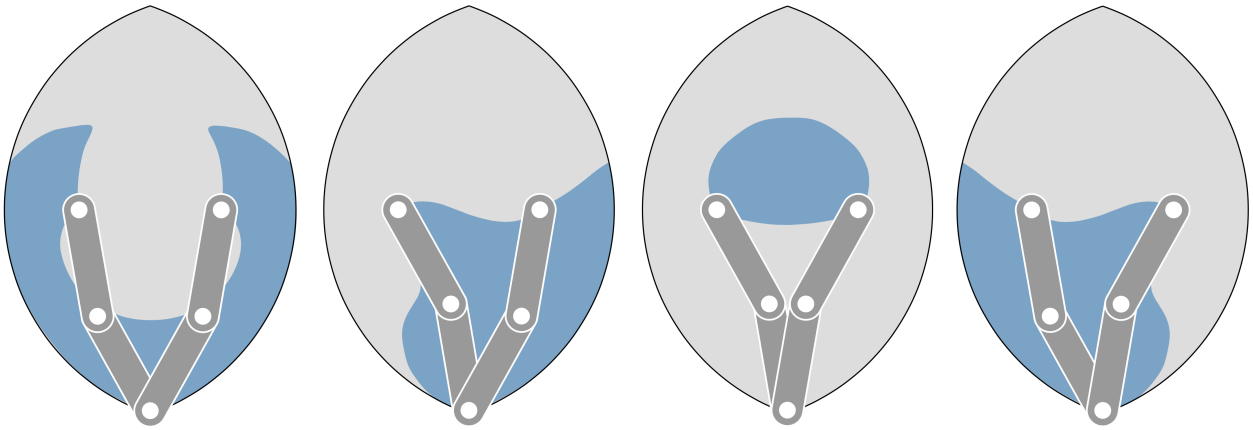
Switching between assembly modes is very difficult from a control point of view and requires passing through a so-called *Type 2 singularity*, where the distal links are aligned. We will address this issue in Chapter 4.

For the negative assembly mode, there are four working modes too. These are illustrated in Fig. 1.9. The workspace zones associated with these working modes are different from the zones associated with the positive assembly mode. Combining all eight zones yields DexTAR's *complete workspace*, which is the lens-shaped area bounded by two circular arcs.



(a) assembly mode + (b) assembly mode −

Figure 1.8: A schematic of DexTAR in each of its two assembly modes.



(a) +− (b) −− (c) −+ (d) ++

Figure 1.9: DexTAR'S four different working modes, for its negative assembly mode.

⚠ DexTAR does not have a sensor that allows its controller to detect the current assembly mode. Thus, the only way to specify that an assembly mode change has occurred is by using a special command that will be presented in Chapter 3.

If you are a graduate student, you most probably have lots of questions: What's the problem with singularities? Where does the complex shape of each workspace come from? All these issues will be addressed in Chapter 4. If you are an undergraduate or college student, you are probably just burning with desire to ask us: Why should I know all this stuff before using DexTAR?

Well, indeed, if you use the right command, DexTAR takes care of all this working mode and assembly switching business. Just specify the position where its end-effector should go, and DexTAR will choose the best arms configuration corresponding to the desired TCP position, and the best path to reach this (arms) configuration. However, you need to know about working modes and assembly modes if you want to use DexTAR as a drawing tool. DexTAR can draw a continuous curve (consisting of line segments and circular arcs)

only in one of the four workspace zones shown in Fig. 1.7. For example, DexTAR cannot draw a continuous line between the midpoints of the two circular arcs bounding DexTAR's workspace. Well, it can, but not without lifting the pen. Furthermore, switching assembly modes requires some tricky programming and knowing what a Type 2 singularity is.

1.3 Working with DexTAR

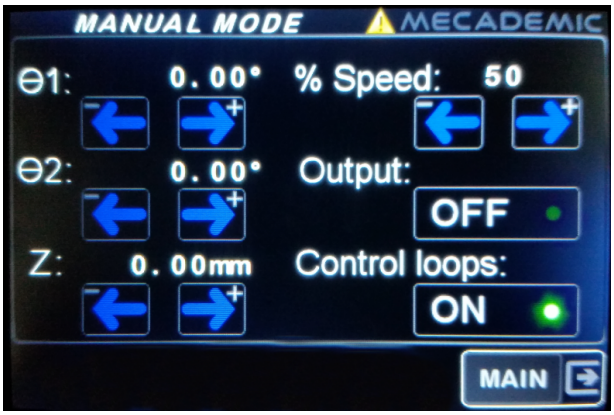
When a program has been launched, the yellow LED on DexTAR's body flashes at 60 Hz. You can pause execution by pressing STOP in the Auto Mode menu. Attention, the robot does not stop instantaneously. While the robot is stopped, the yellow LED continues to flash at 60 Hz. To resume execution of the program, press START again. Alternatively, you can press HOME to bring DexTAR to its home configuration and thus cancel program execution.

At any time, you can press the E-stop button to deactivate immediately the motors. A red × icon () will appear on top of the touchscreen and the yellow LED will turn off. The same will happen if a motion control error occurs. Indeed, because of unmodelled effects such a friction in the belts that are slightly different from one robot to the other, in some extreme situations (e.g., when moving fast close to a Type 2 singularity), DexTAR's controller may not be able to execute certain motions. Similarly, if you somehow manage to bring DexTAR into a collision, the robot will stop, the yellow LED will turn off, and a control error will be displayed. In all of these situations, you'll need to restart DexTAR (turn it off, then turn it on, then home it, then load a program, etc.). And of course, don't forget to release the E-stop button, in case you've pressed it.

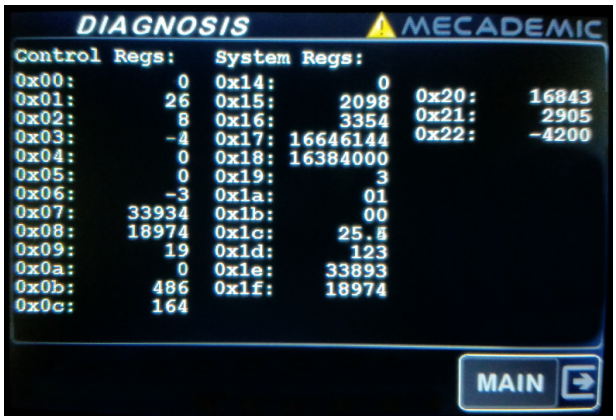
As long as you are using DexTAR's embedded controller, you'll probably never bring the proximal links into a collision. You will, however, occasionally crash the end-effector into a ball (e.g., because you are trying to place a ball in a hole that is already occupied by another ball). Because of the stepper motor used for the vertical displacement of the end-effector, DexTAR cannot detect such collisions. Yet, it's almost certain that your program will no longer work as desired, after such a collision. You will therefore need to restart the robot.

If the robot has been homed and is not executing a program, you can jog it manually via the touchscreen. Select Manual Mode from the main menu. This will bring you the screen shown in Fig. 1.10(a). You can then use the arrow buttons on the left to jog each of the three motors, or the arrow buttons on the right to select the jogging speed. Note that if you jog DexTAR to a Type 2 singularity, its controller will produce an error and you will have to restart the robot. The ON/OFF button in the Manual Mode menu is used to activate/deactivate the electromagnet.

You can deactivate the motors by pressing the OFF button underneath Control loops. The yellow LED will turn off. You can then remove the safety covers and move DexTAR's arms directly with your hands. The active-joint angles and the vertical actuator position will be displayed on the touchscreen. When finished, place back the safety covers and press the ON button underneath Control loops to reactivate the motors.



(a) Manual Mode menu



(b) Diagnosis menu

Figure 1.10: The Manual Mode and Diagnosis screens.

Finally, the Diagnosis menu, Fig. 1.10(b), is reserved for troubleshooting while the Parameters menu is destined for those who use their own external controller. The PC↔DexTAR menu will be described in the next chapter.

1.4 Changing DexTAR's End-Effectors

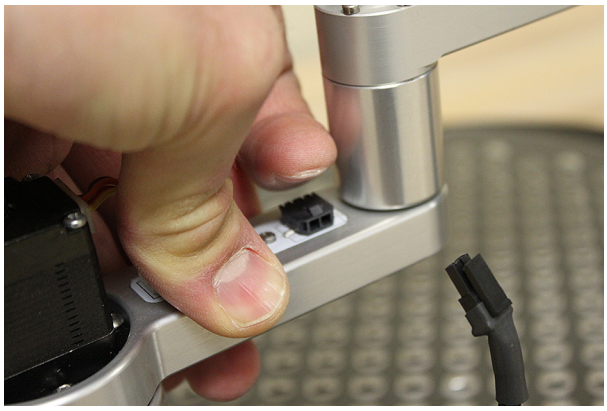
As already mentioned, DexTAR is shipped with the electromagnetic tool already installed. To remove it, you must turn the robot off. The most delicate operation is then unplugging the small black connector. You must NOT pull the black cable away. Instead, you must gently lift the tiny plastic latch and push it away, as shown in Fig. 1.11. Once the connector is unplugged, simply unscrew the electromagnetic tool with your hand.

Installing the pen tool is even trickier, because of the length of the standard D1 ballpoint pen refill (67 mm long, 2.3 mm in diameter). The bright side is that you can find such refills of different colours at any office supplies store for less than $2 apiece (e.g., the Zebra 4C-0.7 refill). These refills are typically used in multi-colour and mini ballpoint pens.

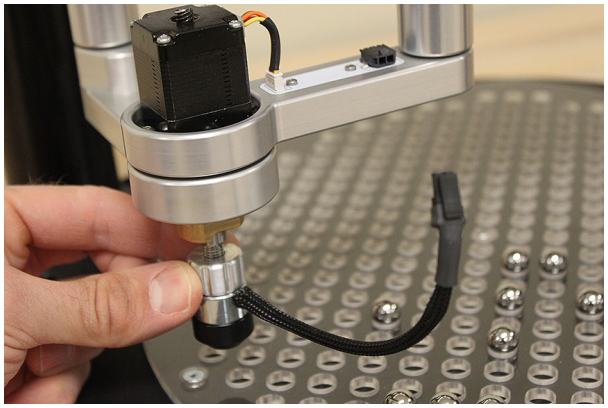
To install the pen tool, first turn off the robot, remove the electromagnet tool and manually push the linear actuator all the way up. You also need to remove the acrylic plate. Because four neodymium magnets embedded in the robot's base retain the plate, you might need to use some wedge shaped tool. Then, bring the hollow shaft of DexTAR's linear actuator directly over the central through hole on DexTAR's base. Next, slightly lift DexTAR's base at one end and insert the pen tool inside the central through hole on DexTAR's base, as shown in Fig. 1.12(a). Then insert the ballpoint pen refill in the hollow shaft of the linear actuator. Note that the spring should be mounted on the ballpoint pen refill. Finally, screw the pen tool in place using your hand, as shown in Fig. 1.12(b).

For drawing applications, you need to use the acrylic plate that has no holes in it. Place it firmly onto DexTAR's base by pressing upon the four screws to make sure the pins are fully inserted into the robot's base. Finally, place a sheet of paper (A4 or Letter format) and use the blue magnets to retain the sheet. Obviously, an A4 or Letter sheet of paper does not cover DexTAR's complete workspace, so for some drawings you might need to place your

(a) lift the connector latch



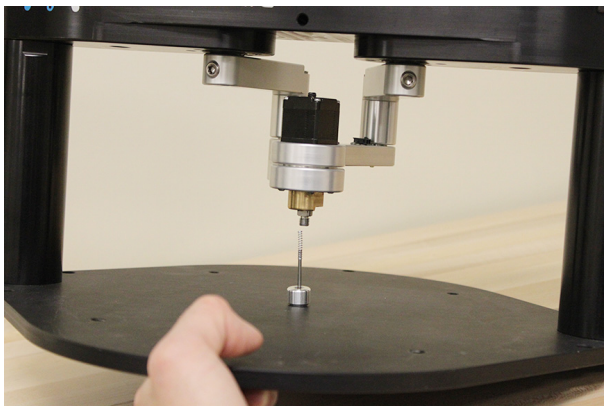
(b) push the latch away



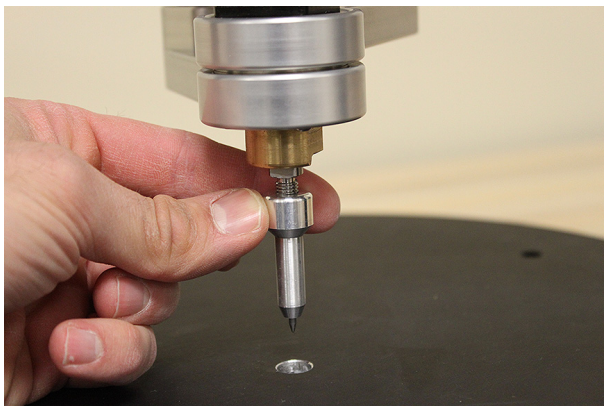
(c) unscrew the electromagnetic tool

Figure 1.11: Removing the electromagnetic tool.

sheet of paper closer to one end, in which case you can only use two magnets. Of course, it is best to use our laser-cut paper which has the exact shape of DexTAR's complete workspace.



(a) insert the pen tool into the base hole



(b) insert the refill and screw the pen tool

Figure 1.12: Installing the ballpoint pen tool.

Chapter 2

DexTAR Sim

Most industrial robots are programmed by the *drive-through* (jogging) method, and there are increasingly more robot manipulators that can be programmed by the *lead-through* method. Both of these methods are called *online programming*. However, industrial robots are programmed most efficiently offline, using proprietary robot programming languages and *simulation and offline programming software*. DexTAR Sim is such a software application, which allows you to program DexTAR using our own programming language called *Mecaprol*, simulate and visualize in 3D the robot motions, transfer compiled code to the actual robot, and even update DexTAR's controller. DexTAR can only be programmed offline, through DexTAR Sim.

2.1 Installing DexTAR Sim

Currently, DexTAR Sim runs only on Windows. The installer can be downloaded from www.mecademic.com/DexTAR.html. It is a relatively small program that requires less than 32 Mb of disk space, and the best thing about DexTAR Sim is that it is completely free. You can install it on as many PCs as you want and you don't even need to register.

Installing DexTAR Sim is pretty straightforward. Just download the .exe file, execute it, and follow the installation steps. During the installation process, a second installation window will pop up which will prompt you to install the USB drivers necessary to connect to DexTAR. You need to install these two drivers, and this takes at least a couple of minutes.

Figure 2.1 shows the composite window of DexTAR Sim, just after installation. You are now ready to jog, program and simulate DexTAR and—if the two USB drivers were correctly installed—to upload your programs to the real DexTAR.

Unfortunately, at this time, there is no online Help, so you need to read this chapter thoroughly in order to learn how to take full advantage of DexTAR Sim.

2.2 Viewing Options

As in any 3D software, the mouse buttons to orbit, zoom or pan the 3D view of DexTAR. Position your mouse cursor over the white background 3D viewer, press and hold the left mouse button while dragging to orbit. Similarly, position your mouse cursor and press and

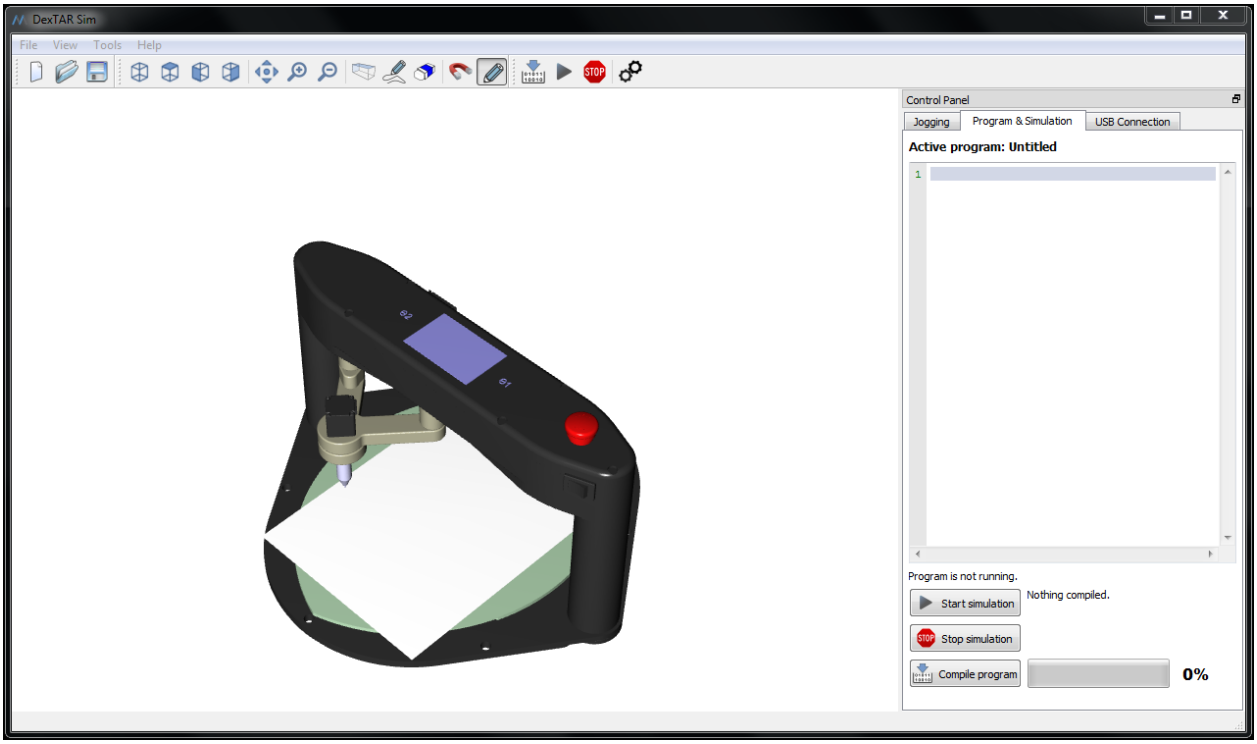


Figure 2.1: DexTAR Sim after installation.

hold the middle mouse button while dragging to pan. Finally, use the mouse scroll button to zoom in and out. Alternatively, press and hold Ctrl while pressing + to zoom in or - to zoom out, or simply use the two magnifying glass icons in the toolbar or in the View menu. You can also use the four-arrows icon to reset pan and zoom to default.

Additionally, you can set the viewport to isometric, top, front or right by clicking the corresponding cube-in-wireframe icons in the toolbar or in the View menu. In some views, DexTAR's body hides too much of the links. That is why, we have added the prism-in-perspective icon which toggles the transparency of the base.

Finally, since DexTAR comes with two tools, we have made available two icons, the magnet and the pencil, which toggle between both setups: (1) the pen tool and a letter-format sheet of paper and (2) the electromagnet and the perforated acrylic plate. Note that you cannot change the location of the sheet of paper. Furthermore, it goes without saying that the actual ink deposition is not simulated. Similarly, note that DexTAR Sim does not simulate the displacement of the balls. (In other words, the ball in the electromagnet and the one in the center of the acrylic plate always remain there during simulations.)

To compensate for the lack of more realistic simulation, we have added the pencil-with-trace icon, which toggles the display of the trace left by the TCP (Fig. 2.2). This trace is left only when running programs and not when jogging the robot. Furthermore, two different trace colours are used by default, as a function of the z coordinate of the TCP. This option is in the Graphical settings tab of the Settings window, which can be accessed via the Settings menu item in the Tools menu. In that window (Fig. 2.2), the "Trace Z level limit" field is the z threshold level, with respect to the world coordinate system (WCS, see Chapter 3).

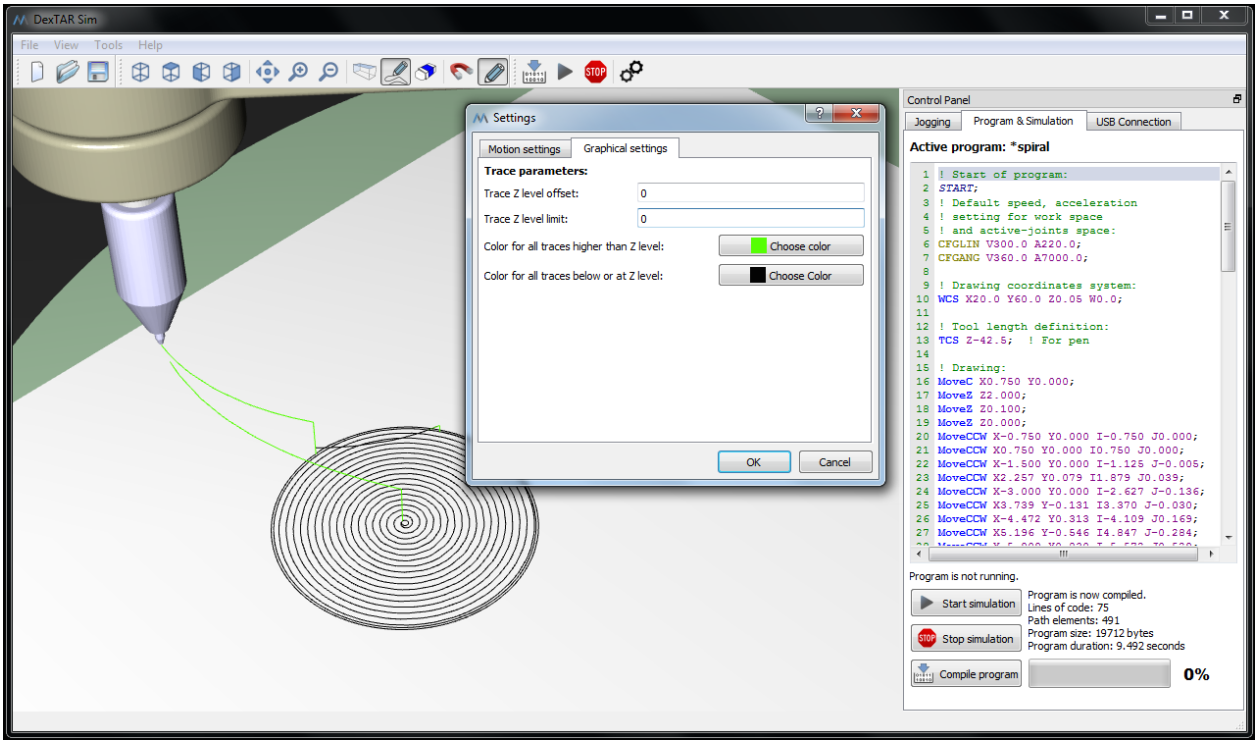


Figure 2.2: DexTAR's TCP can leave a two-coloured trace.

TCP positions with a z coordinate (with respect to the WCS) smaller or equal than the threshold level will leave a trace with one colour, while the rest will leave a trace with the second color. Finally, if you do not wish that the whole trace is vertically offset, you can specify the offset in the "Trace Z level offset" field.

Note that the trace is erased every time you execute a program. The trace can also be removed using the rubber eraser icon.

⚠ The trace left by the TCP does not correspond precisely to the path of the real robot. The trace smoothness depends on the Cartesian speed of the TCP and on the power of your computer. If, for example, your program makes DexTAR draw a circle at maximum speed and you run DexTAR Sim on a very old computer, your TCP trace might look like a hexagon.

2.3 Jogging

DexTAR Sim's jogging menu, located in the Control Panel (Fig. 2.3), is by far the most innovative aspect of the software and is quite different from the jogging menu of any other robot simulation software. Of course, we offer the most basic feature, which is axis-by-axis jogging, though instead of using the typical slider bars, we use dials. Indeed, these are absolutely necessary for the active joints since they are unlimited. However, we also offer the possibility to jog the robot using the maps of its active-joint space and Cartesian workspace.

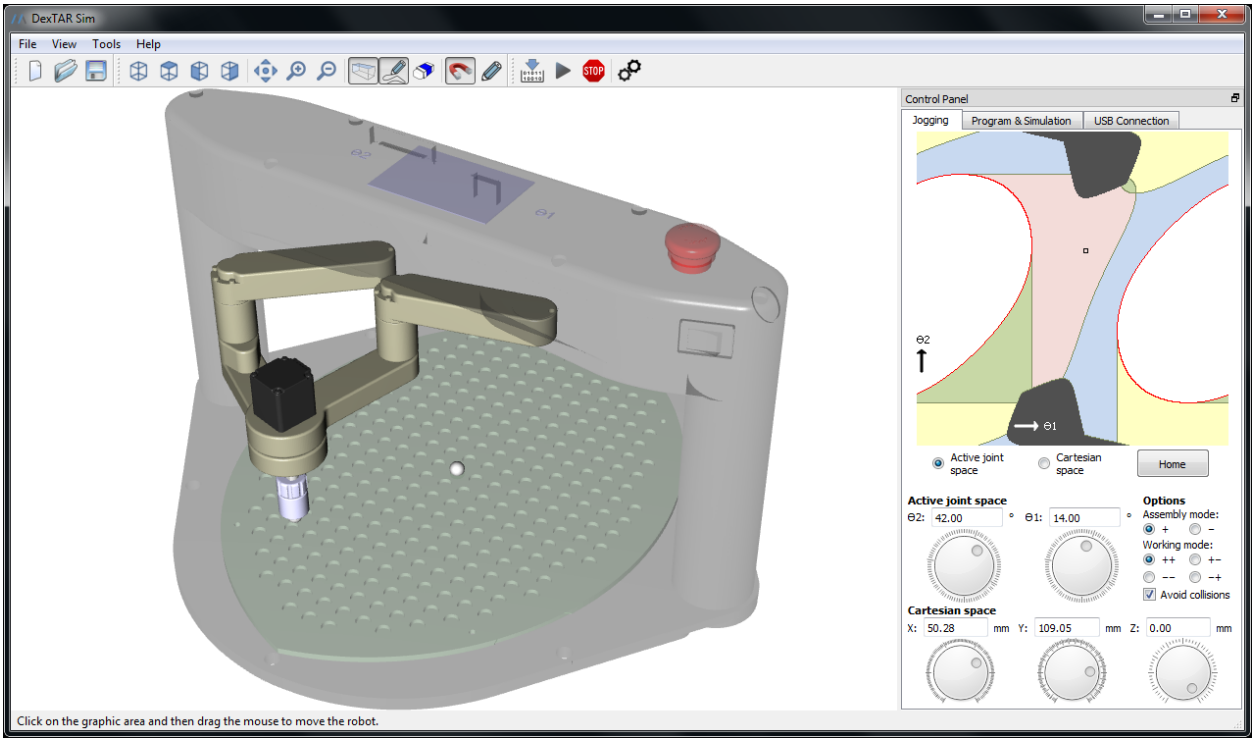


Figure 2.3: DexTAR Sim's jogging menu.

To modify an active-joint variable or one of the Cartesian coordinates of the TCP, simply rotate the dial by clicking and dragging it in circular manner or by positioning your cursor over the dial and rotating the mouse wheel. You can also enter directly the desired value, using the corresponding text field. Note that DexTAR Sim does not accept just any value. If the value that has been entered is unfeasible, its will not be accepted. Note that by default, the "Avoid collisions" checkbox is checked. If for some reason, you don't want to worry about reality, you can uncheck this box, and then you will be able to jog into configurations that are not feasible in practice.

Unchecking the "Avoid collisions" checkbox brings DexTAR to its home configuration, where both active-joint angles are zero and the robot is in its positive assembly mode. Pressing the Home button does the same thing.

While jogging, the assembly mode and working mode radio buttons change accordingly. Alternatively, you can change the assembly and working mode manually. Note, however, that if the "Avoid collisions" checkbox is checked, for some TCP positions, some assembly modes will not be feasible and, therefore, cannot be selected.

Finally, you can display the active-joint space or the Cartesian workspace using the two radio buttons on the left of the Home button. To learn more about these, you must read Chapter 4. For now, note that when DexTAR moves, a small circle indicates the current active-joint configuration of Cartesian position (x and y coordinates only). The most interesting feature, however, is that you can drag this circle with your mouse and thus move the robot. While this feature has limited practical utility, it has a great pedagogical value, which can be appreciated after reading Chapter 4.

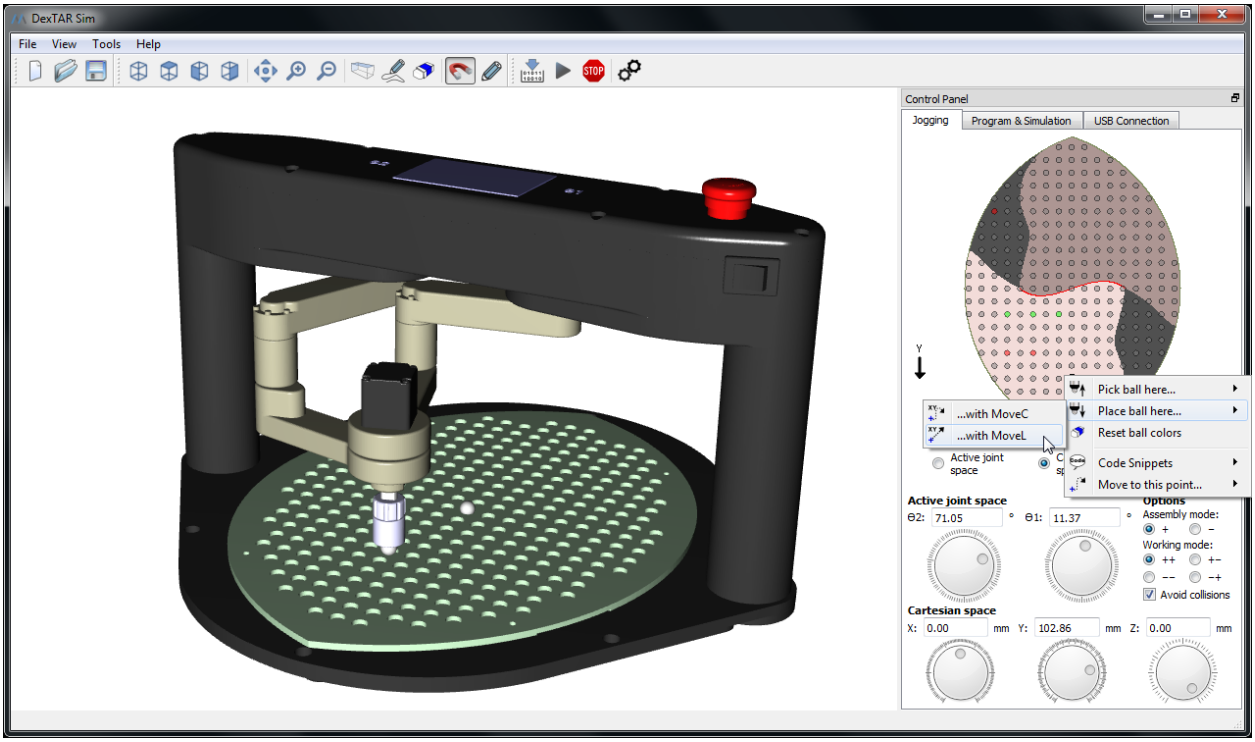


Figure 2.4: You can insert code snippets or commands by right clicking in the active-joint space or Cartesian space maps.

2.4 Programming

Programming in Mecaprol (see Chapter 3) is mainly done in the Program & Simulation tab of the Control Panel (Fig. 2.2). Note that you can detach the Control Panel by clicking at the miniature stacked windows icon in the upper right corner of the panel.

You can write a Mecaprol program from scratch in the program field of the "Program & Simulation window. Alternatively, you can right click in that field and select "Insert program template," which will insert the common part of any Mecaprol program. Of course, you can also load a program from the File menu. You can either select Open Example, which will automatically bring you to a folder with several preloaded Mecaprol programs, or select Open, which will bring you to the folder where you last saved a Mecaprol program.

Once you have created your own program, or modified an existing program, you can save it and declare its name. The name will automatically appear on top of the program text field. Once the program modified, an asterisk (*) will appear in front of the name of the program, which means that the program needs to be saved.

The text field where the Mecaprol program is written offers syntax highlighting and line numbering by not advanced editing features. Therefore, if you are writing a lengthy program, you might consider using a more sophisticated third-party text editor.

You can use the jogging window to insert code snippets or basic motion commands. Firstly, position your cursor in the program to the place where you want the code or the command to be inserted. Then switch to the jogging menu and right click in the active-joint space map or in the Cartesian space map (Fig. 2.4).

Currently, there are only two code snippets: one for bringing DexTAR to its home configuration, and another that demonstrates an assembly mode. In addition, there are three motion commands that can be inserted from the jogging menu You right-click at a point in the active-joint space or the Cartesian space and then select one of the commands. Finally, if you use the Cartesian workspace, you can right-click at one of the tiny circles (or close to it), which represent the holes in the perforated acrylic plate, and select either "Pick ball here" or "Place ball there." This will insert five lines of code, which correspond to the requested action. Additionally, the circle will be coloured green, if a ball is picked from the corresponding hole, or red if a ball is placed at the corresponding hole.

Finally, if you want to test your program, you need to compile it first using the Compile program button in the Program & Simulation window, the compile icon in the toolbar, or the Compile item in the Tools menu. If you have any errors, these will be indicated at the bottom of the Program & Simulation window. If the compilation was successful, you will see some data regarding the compiled program (number of lines, program size, etc.).

To run the program, simply click on the "Start simulation" button in the Program & Simulation window, the Start icon in the toolbar, or the Start item in the Tools menu. During the simulation, you can click on "Stop simulation," which cancels the simulation. You can also switch to the jogging menu and observe the displacement of the robot in its active-joint space or Cartesian space.

2.5 Program Upload

The final and most important task to perform on DexTAR Sim is to upload your program to the actual robot. This is done entirely through the USB Connection tab of the Control Panel. To upload your program, you must first successfully compile it. Then, of course, you must connect your PC to the actual robot through the USB cable provided. Next, you must turn the robot on (if it is not already turned on) and select the PC↔Dextar icon from the Main menu on DexTAR's touchscreen. If the USB connection was successful, you will receive according messages on both DexTAR's touchscreen and your PC.

The next step is to click on the "List all program" button in the USB Connection tab. This will list all the programs found on the actual DexTAR. You can now upload your program by clicking on "Write program" button. The new program (called "spiral") will eventually appear in the list, which means that it is already uploaded to DexTAR (Fig. 2.5). You can now press the MENU button on DexTAR's touchscreen, and follow the necessary steps to select the new program and execute it.

While in the PC↔Dextar menu (and with the USB connection still active) you can rename a program on DexTAR by selecting it in the program list of the USB Connection tab and pressing the "Rename selected program" button. You can also delete a program by pressing the "Delete selected program" button. You will need to confirm this action, since there is no turning back (no backup). Finally, you can delete all programs by pressing the "Format file system" button. Note that you will need to do this if during the upload of a program, the USB connection is broken (e.g., you accidentally unplug the USB cable). It is therefore strongly recommended that you keep a backup of all programs that are on DexTAR.

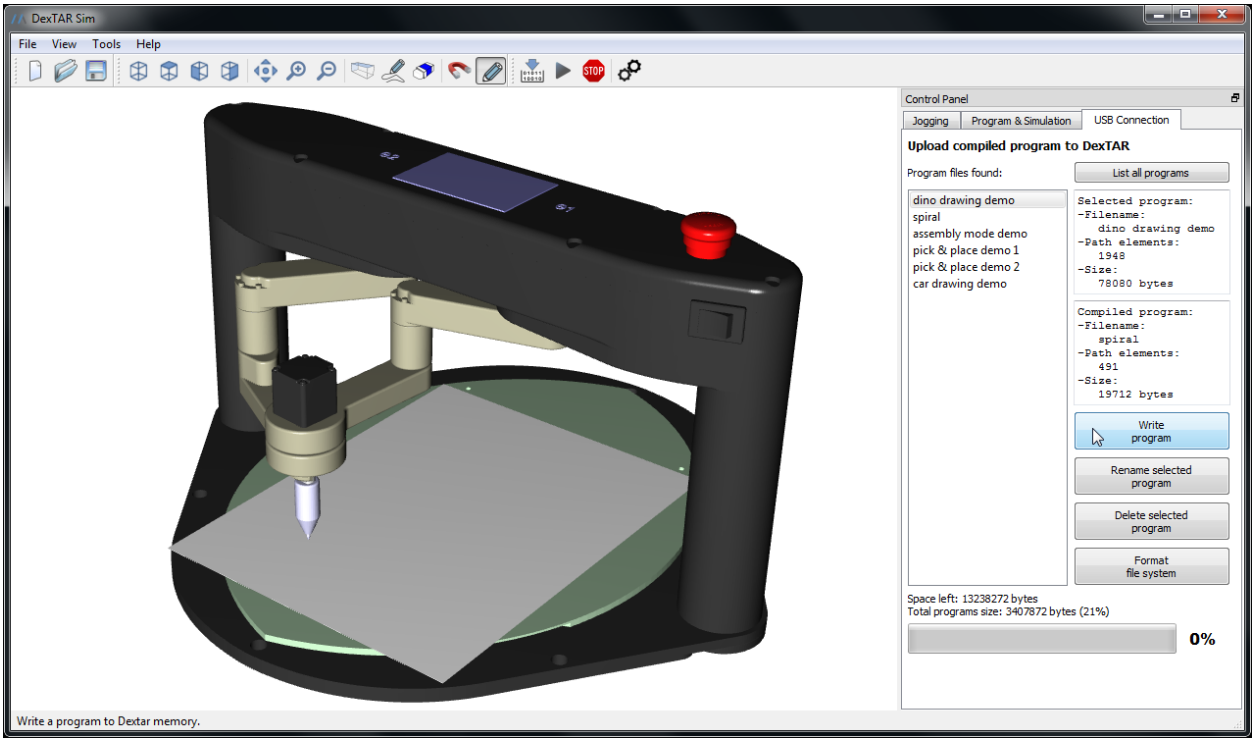


Figure 2.5: DexTAR Sim's USB Connection menu.

Finally, note that if you want to upload a program that has the same name as a program that is already on DexTAR, you will first need to delete the latter. This is true even if you have just uploaded the program, made a few changes, recompiled the program, and now want to upload the new program.

Chapter 3

Mecaprol

Most industrial robots are programmed using proprietary robot programming languages. For example, robots manufactured by ABB use the RAPID language, while those built by KUKA are programmed in KRL. Some of these programming languages are similar to each other, but unfortunately, there is no standard common programming language. Mecademic's robots are programmed in yet another proprietary programming language that we call *Mecaprol*. Mecaprol is much more elementary than sophisticated languages like RAPID and KRL, but follows the same philosophy. Furthermore, Mecaprol is very similar to *G-code*, the standard common language for programming CNC machine tools. This might be particularly useful when using DexTAR as a drawing tool, because there are plenty of software tools for converting vector drawings to G-code.

3.1 Structure of a Program and Syntax

Mecaprol is a compiled language. A Mecaprol program consists of a single non-compiled text (ASCII) file with an .mp extension. The name of the file must contain only alphanumeric characters and spaces and its length must be no more than 20 characters.

Each command should be followed by its arguments and end with a semi-colon, all of these on a single line.

Comments can be included in a program at any place. A comment starts with an exclamation mark and ends with a new-line character.

Mecaprol is not case sensitive. For example, the command `MoveL` can be written `MOVEL` or `movel`. Furthermore, Mecaprol is a free format language, meaning that spaces can be used anywhere except in commands, arguments and numerical values. Tab and form-feed characters can be used wherever a space can be used.

Numerical values can be expressed as an integer (for example 5, −30 or 3E2) or a decimal number (for example 2.56, −0.340, .03, or −256E−2). The value must be in the range specified by the ANSI IEEE 754 Standard for Floating-Point Arithmetic.

The header of a program can only contain comments or whitespace characters. The body of the program starts with the command `START` (followed by a semi-colon) and ends with the command `END` (followed by a semi-colon). No commands are allowed after the command `END`.

Example 3.1 shows a Mecaprol program that illustrates nearly all currently available commands. Note that the line numbers, the italic formatting on START and END and the highlighting are as automatically shown in the DexTAR Sim software editing panel. They are not part of the program, which we recall is a simple text file.

Currently, there are no constants or variables, no arithmetic or logical expressions, no IFs, no FORs, no WHILEs, no user messaging. These and many new features and commands will be implemented in the near future.

Exemple 3.1: A simple Mecaprol program illustrating a pick and place operation.

```
1  ! The electromagnetic tool and the acrylic plate with holes must be installed.
2  ! Three 11 mm steel balls must be placed at the front extremity of the plate.
3
4  START;
5  CFGLIN V40.0 A10000.0; ! Speed and acceleration settings for MoveL, etc.
6  CFGANG V360.0 A10000.0; ! Speed and acceleration settings for MoveJ and MoveC.
7
8  ! Definition of the world coordinate system
9  WCS X0.0 Y0.0 Z-2.50 W0.0; ! Z-2.50, because of the holes
10 ! Definition of the tool coordinate system
11 TCS Z-39.00;
12
13 ! Start manipulating the three balls
14 MoveC X0.00 Y162.25 M1; ! pick the first ball in the ++ working mode
15 MoveZ Z0.00;
16 Output ON; !turn the electromagnet on
17 MoveZ Z12.00; !lift the tool at least the height of a ball
18 MoveL X0.00 Y14.75;
19 MoveZ Z0.00;
20 Output OFF; ! turn the electromagnet off
21 MoveZ Z12.00;
22 WaitTime T1000; ! Pause for 1 second, DexTAR is almost in a Type 2 singularity
23 MoveC X14.75 Y162.25;
24 MoveZ Z0.00;
25 Output ON;
26 MoveZ Z12.00;
27 MoveC X-118.00 Y14.75;
28 MoveZ Z0.00;
29 Output OFF;
30 MoveZ Z12.00;
31 MoveC X-14.75 Y162.25;
32 MoveZ Z0.00;
33 Output ON;
34 MoveZ Z12.00;
35 MoveC X118.00 Y14.75;
36 MoveZ Z0.00;
37 Output OFF;
38
39 ! Move DexTAR to its home configuration
40 MoveZ L0.0; ! it's always best to start with the linear actuator
41 MoveJ A0.0 B0.0;
42 END;
```

Table 3.1: List of currently available Mecaprol commands.

Command	Arguments
START	-
END	-
CFGLIN	V[value] A[value]
CFGANG	V[value] A[value]
TCS	Z[value]
WCS	X[value] Y[value] Z[value] W[value]
MoveL	X[value] Y[value] S[value]
MoveCW	X[value] Y[value] I[value] J[value] S[value]
MoveCCW	X[value] Y[value] I[value] J[value] S[value]
MoveJ	A[value] B[value] S[value]
MoveC	X[value] Y[value] M[value] S[value]
MoveZ	Z[value] L[value] S[value]
Output	[ON / OFF]
WaitTime	T[value]
Halt	-
AssyMode	-

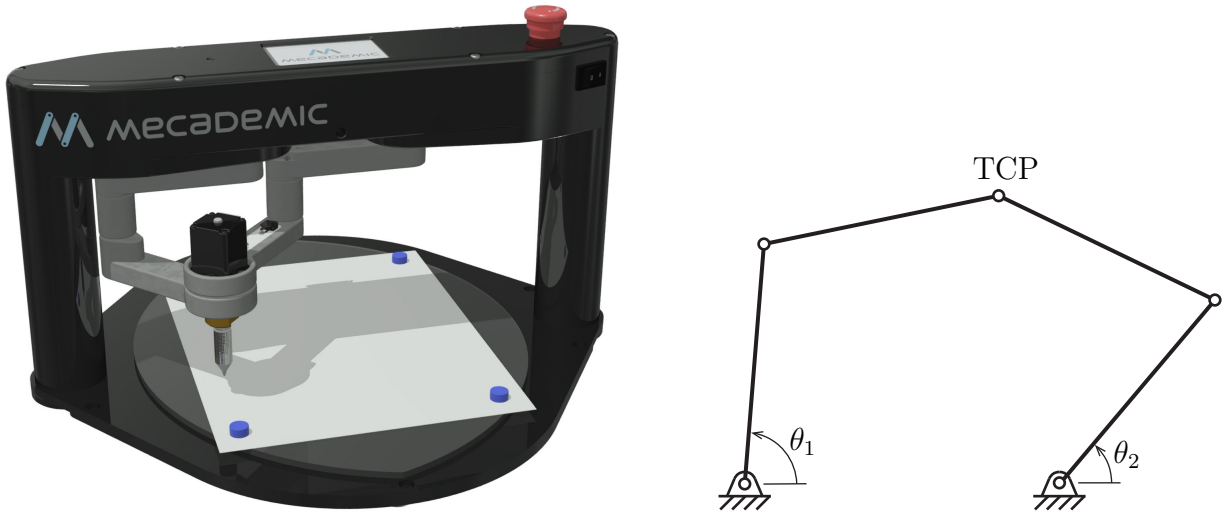
3.2 Commands

Table 3.1 presents most currently available Mecaprol commands and their corresponding arguments. A few of these arguments are optional (e.g., `S`, which stands for speed). Note that each argument consists of a numerical value preceded by a letter (`V`, `A`, `J`, etc.), except for the `Output` command. Some of these letters have completely different meanings from one command to another (i.e., `J` and `A`).

3.2.1 START

As already mentioned, the `START` command is required and must precede all other commands. Furthermore, before executing a program, DexTAR must be brought to its *home configuration*. In this configuration, $\theta_1 = \theta_2 = 0°$, and the end-effector is in the front portion of the robot, as shown in Fig. 3.1(a), i.e., the robot is in its positive assembly mode. The *active-joint angles*, θ_1 and θ_2, are defined as in Fig. 3.1(b). Note that active joint 1 is the one closest to the E-stop button. Since the robot cannot detect its assembly mode, it will run even if started in the negative assembly mode. However, motion commands will not function correctly. Fortunately, it is possible to change the assembly mode later in the program.

Note that if you compile a program in DexTAR Sim, the robot will automatically be moved to its home position, as soon as the command `START` is encountered.



(a) home configuration (b) active-joint angles (top view from behind)

Figure 3.1: Definition of DexTAR's (a) home configuration and (b) active-joint angles (active joint 1 is the one closest to the E-stop).

3.2.2 END

The `END` command is also required and must be the very last command in a Mecaprol program, but first DexTAR must be brought to its home configuration (see Example 3.1).

3.2.3 CFGLIN

The `CFGLIN` command specifies the maximal values for the TCP speed and acceleration in a horizontal plane, for the linear and circular motion commands (`MoveL`, `MoveCW` and `MoveCCW`). Note that the speed of the linear actuator, i.e., the vertical speed of the TCP, can only be modified with the `MoveZ` command. The `CFGLIN` command is optional but can be used as many times as needed. Its two arguments, both of which are required, are:

`V[value]` maximal horizontal speed of the TCP (in mm/s);

`A[value]` maximal horizontal acceleration of the TCP (in mm/s^2);

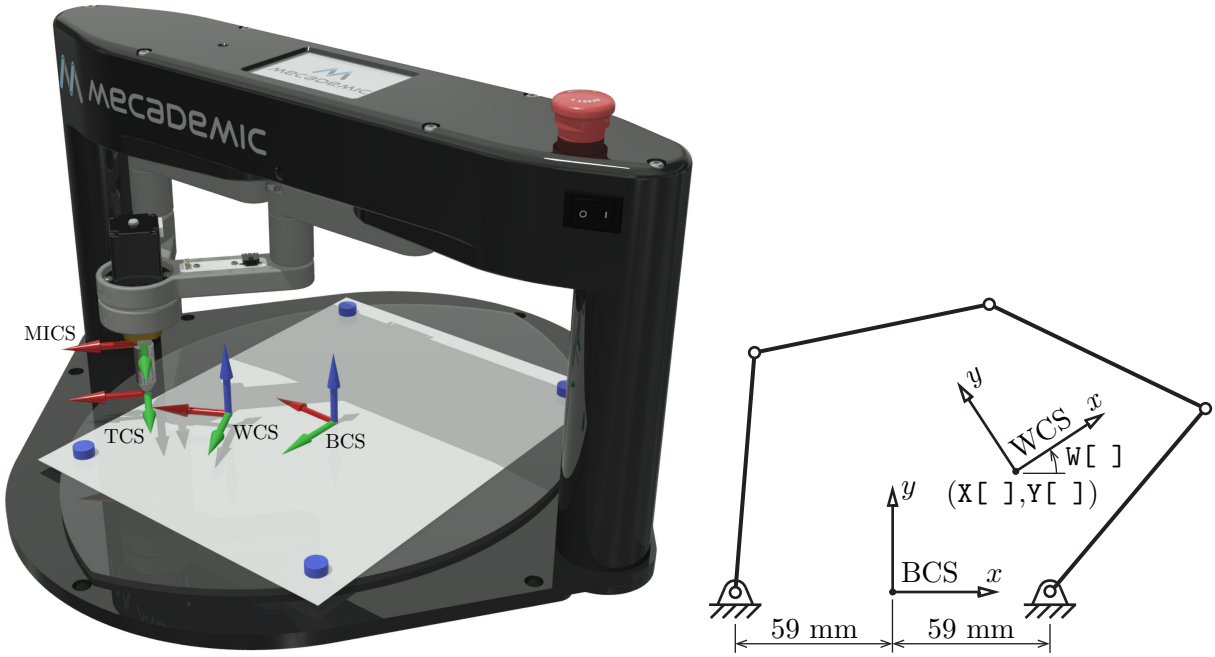
If the `CFGLIN` command is not used, the maximal values for the speed and the acceleration of the TCP in a horizontal plane will only be limited by the performance of the servo motors. These values depend strongly on the robot configuration.

3.2.4 CFGANG

The `CFGANG` command specifies the maximal values for the speed and the acceleration of the TCP in a horizontal plane, for the active-joint space motion commands (`MoveJ` and `MoveC`). The `CFGANG` command is optional but can be used as many times as needed. Its two arguments, both of which are required, are:

`V[value]` maximal horizontal speed of the TCP (in mm/s);

`A[value]` maximal horizontal acceleration of the TCP (in mm/s^2);



(a) Coordinate systems (b) Definition of the WCS

Figure 3.2: (a) The four coordinate systems used in Mecaprol and (b) definition of the WCS with respect to the BCS (top view), using the `WCS` command.

If the `CFGANG` command is not used, the maximal values for the speed and the acceleration of the TCP in a horizontal plane will only be limited by the performance of the servo motors. These values depend on the robot configuration.

3.2.5 TCS

Figure 3.2(a) illustrates the four coordinate systems used in Mecaprol. Since DexTAR is essentially a planar robot, the z axes of all coordinate systems are always parallel. The *tool coordinate system* (*TCS*) is defined by the user with respect to the *mechanical interface coordinate system* (*MICS*). The z axis of the MICS coincides with the axis of the revolute joint between the two distal links and the xy plane of the MICS coincides with the shoulder of the linear actuator's shaft, just before the short threaded end. The TCS is simply offset from the MICS along the negative direction of the MICS's z axis, and the origin of the TCS is what we refer to as the TCP.

The `TCS` command specifies uniquely the z offset of the TCS with respect to the MICS. Its single argument, obviously required, is:

`Z[value]` z coordinate (in mm) of the TCS' origin with respect to the MICS;

By default, the TCS coincides with the MICS, and can only be modified in a program. When you use the electromagnetic tool (tightly screwed) with the ⌀11 mm balls, if you want to define your TCP at the bottom of a ball picked up by the tool, then you need to specify

```
TCS Z-39.00;
```

Similarly, if you want to define the TCP in the case of the ballpoint pen tool at the tip of the pen, the right command would be

```
TCS Z-42.50;
```

Of course, since the pen is mounted on a spring, it would be wiser to use a slightly shorter offset (e.g., `Z-42.00`), so that the pen applies some pressure on the paper.

The TCS command can be used as many times as needed in a program. However, since DexTAR is not equipped with an automatic tool changer, there is little sense in using this command more than once. Perhaps, you might want to manipulate balls of different diameters, but note that our electromagnetic tool was specifically designed for ⌀11 mm balls.

⚠ DexTAR is a 3-DOF robot that is used only for positioning its TCP in three-dimensional space. Therefore, although we define the TCS, we actually care only about the origin of this coordinate system, i.e., the TCP.

3.2.6 WCS

In Mecaprol, TCP coordinates are always expressed in the *world coordinate system* (*WCS*). By default, the latter coincides with the *base coordinate system* (*BCS*), which is illustrated in Fig. 3.2(a). The xy plane of the BCS coincides with the upper surface of the lens-shaped acrylic plate, and the x axis intersects (and is normal to) the axes of DexTAR's actuated joints.

⚠ When the linear actuator is fully retracted (i.e., the TCP is in its highest position), the distance between the xy planes of the MICS and the BCS is 52.50 mm.

The `WCS` command specifies the position and orientation (i.e., the rotation about the z axis) of the WCS with respect to the BCS. Its four arguments, all of which are optional, are:

`X[value]` x coordinate (in mm) of the WCS' origin with respect to the BCS;

`Y[value]` y coordinate (in mm) of the WCS' origin with respect to the BCS;

`Z[value]` z coordinate (in mm) of the WCS' origin with respect to the BCS;

`W[value]` the orientation (in degrees) of the WCS with respect to the BCS.

The WCS command can be used as many times as needed. Imagine, for example, that you have to draw two squares, sharing the same origin, but rotated 45° with respect to each other. This task will be much simpler if you redefine your WCS twice, as shown in Example 3.2.

Note that the `WCS` command could also be very useful if you change the thickness of your drawing paper. You will only need to offset your WCS along the positive direction of the BCS's z axis. Don't forget, however, that the new WCS must be defined with respect to the BCS, and not with respect to the old WCS. Similarly, note that in Example 3.1, we have defined a WCS that is shifter along the negative direction of the BCS's z axis, to account for the sagging of the balls in the holes.

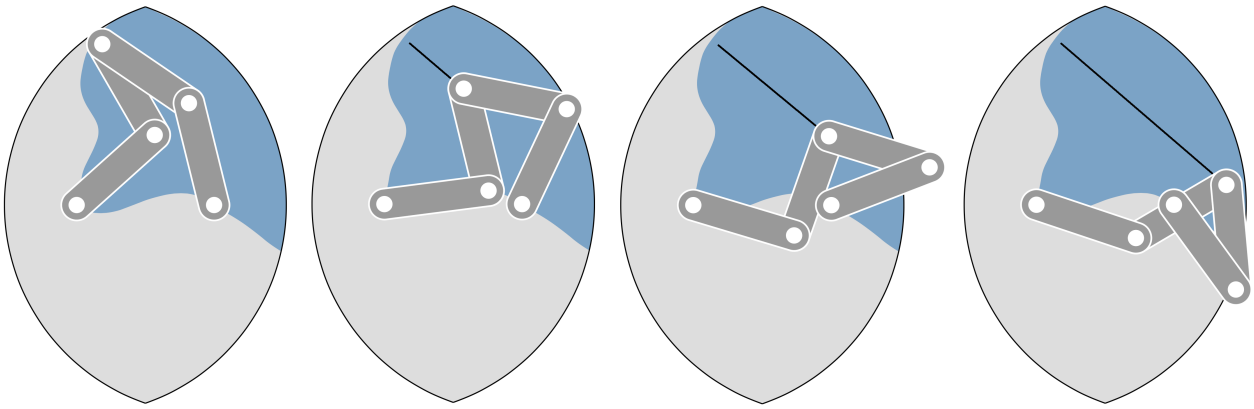
Exemple 3.2: An example illustrating the use of the WCS command.

```
! The pen tool and the unholed acrylic plate with paper should be installed.
! This program makes DexTAR draw two squares on paper of 0.05 mm thickness.

START;

! Definition of the world coordinate system
WCS X24.00 Y71.00 Z0.05 W0.00;
! Definition of the tool coordinate system
TCS Z-42.00;

! Go to initial position in the working mode of the home configuration (++)
MoveC X25.00 Y-25.00 S100 M1; !actuator is already up at the beginning

! Draw a square of side 50 mm
MoveZ Z0.00 S100.0; ! Move pen down
MoveL X-25.00 S100.0;
MoveL Y25.00;
MoveL X25.00;
MoveL Y-25.00;
MoveZ Z2.00; ! Move pen up

! Definition of a new world coordinate system
WCS X25.00 Y71.00 Z0.05 W45.00;

! Draw the same square
MoveC X25.00 Y-25.00;
MoveZ Z0.00; ! Move pen down
MoveL X-25.00;
MoveL Y25.00;
MoveL X25.00;
MoveL Y-25.00;
MoveZ Z2.00; ! Move pen up

! Move DexTAR to its home configuration
MoveZ L0.0;
MoveJ A0.0 B0.0;
END;
```

3.2.7 MoveL

The `MoveL` command makes DexTAR move its TCP horizontally (i.e., the TCP's z coordinate remains constant) from its current position to a specified position (defined with respect to the WCS), along a linear path, provided that DexTAR is in the assembly mode it assumes it is in, and there are no singularities along the trajectory (Fig. 3.3).

This basically means that you can draw linear segments only in one of the workspace zones shown in Figs. 1.7 and 1.9. If you want to draw a linear segment that spans more than one such workspace zone, you need to subdevide your linear segment into and use several `MoveL` commands, as well as do some tricky manipulations involving working mode change or even assembly mode change.



(a) start configuration (b) intermediate config. (c) intermediate config. (d) end configuration

Figure 3.3: Schematic illustrating the command `MoveL`.

The `MoveL` command has three arguments:

`X[value]` x coordinate (in mm) of the final TCP position with respect to the WCS;

`Y[value]` y coordinate (in mm) of the final TCP position with respect to the WCS;

`S[value]` maximum horizontal speed of the TCP (in %), where 100 corresponds to the maximum horizontal speed of the TCP defined by the `CFGLIN` command.

None of these three arguments is required, but you should obviously use at least one of the first two. The default value for the third argument is the value defined in the last `MoveL` command or 100, if the command is used for the first time. The default values for the first and second arguments are the x and y coordinates, respectively, of the current TCP position.

Note that if DexTAR passes through a configuration where the angle between its distal links is almost 180°, and especially if it moves at high speeds, it might stop and you would have to restart the robot. Therefore, avoid drawing lines that are too close to the boundaries of a workspace zone (for a given working mode and assembly mode).

3.2.8 MoveCW and MoveCCW

The `MoveCW` command makes DexTAR move its TCP along a circular arc in the clockwise direction, starting from the initial (starting) TCP position, and keeping the z coordinate of the TCP constant. As with the `MoveL` command, the whole circular arc should lie in the workspace zone corresponding to the working mode and the assembly mode of the starting robot configuration. The `MoveCW` command takes five arguments:

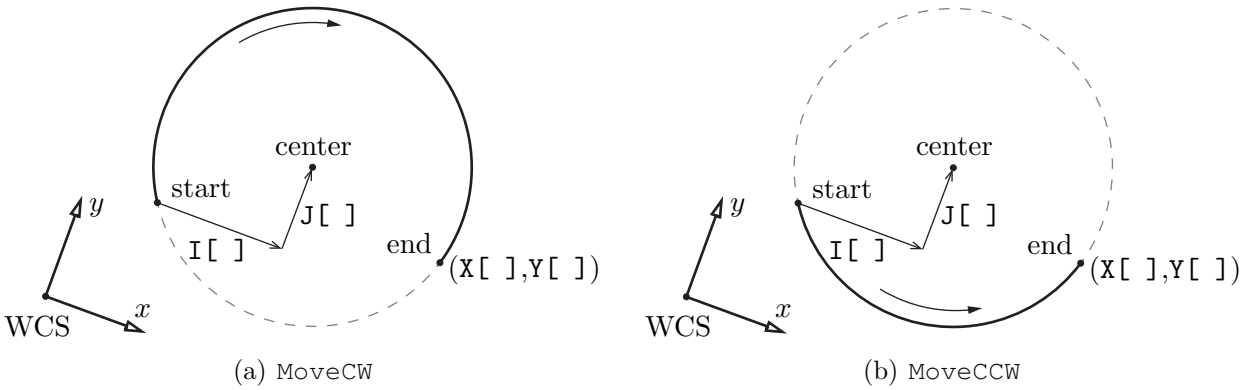
`X[value]` coordinate x (in mm) of the end TCP position with respect to the WCS;

`Y[value]` coordinate y (in mm) of the end TCP position with respect to the WCS;

`I[value]` offset along the WCS' x axis (in mm) with respect to the starting TCP position, defining the center of the circular arc;

`J[value]` offset along the WCS' y axis (in mm) with respect to the starting TCP position, defining the center of the circular arc;

`S[value]` maximum horizontal speed of the TCP (in %), where 100 corresponds to the maximum horizontal speed defined by the `CFGLIN` command.



(a) `MoveCW` (b) `MoveCCW`

Figure 3.4: Schematics illustrating the commands `MoveCW` and `MoveCCW`.

The first four arguments are required, while the last one is optional. The default value for the fifth argument is the value defined in the last `MoveCW` command or 100, if the command is used for the first time.

Similarly, the `MoveCCW` command makes DexTAR move its TCP along a circular arc in the counter clockwise direction, starting from the current TCP position, and keeping the z coordinate of the TCP constant. It is defined in exactly the same way as the `MoveCW` command. Both commands are illustrated in Fig. 3.4, where the start point of the arc is the initial (starting) TCP position, and the end point is the final TCP position. If the final TCP position coincides with the starting position, then a full circle is drawn. Obviously, the `I` and `J` offsets should not be both zero. Finally, for both commands, the center of the circular arc should be equidistant from the starting and final positions.

Clearly, this is not the most intuitive way to define a circular arc. Robot programming languages typically use three points to define a circular arc. However, we stick to it, since this is the way G-code works. Thus, it would be very simple to convert G-code to Mecaprol. Indeed, there exist various software tools for converting vector graphics into G-code, and these could be very useful.

3.2.9 MoveJ

The `MoveJ` command ("J" for active-joint) moves DexTAR's arms from their current configuration to a target configuration defined by the active-joint angles, θ_1 and θ_2.

Understanding that command requires full grasp of the kinematics of DexTAR. This command is only for advanced users, but not understanding this command will prevent you from using DexTAR in an optimal manner. For example, you have to use this command if you want to change your working mode and your assembly mode and draw in each one of the working zones illustrated in Figs. 1.7 and 1.9.

With the `MoveJ` command, the trajectory followed by the TCP is complex and almost certainly not linear. For those who are more curious, it should be mentioned that the trajectory followed by the DexTAR in the $\theta_1\theta_2$ active-joint space is not linear either. It is not necessarily the fastest trajectory possible, but it is quite close to it. Finally, it is possible that DexTAR will change its working mode (probably even more than once) during that trajectory.

The `MoveJ` command has four arguments:

`A[value]` angle θ_1 (in degrees);

`B[value]` angle θ_2 (in degrees);

`S[value]` maximum angular speed (in %) for both rotary motors, where 100 corresponds to 720°/s.

None of these three arguments is required, but you should obviously use at least one of the first two. The default values for the first and second arguments are the θ_1 and θ_2 angles, respectively, of the current robot configuration. The default value for the third argument is the value defined in the last `MoveJ` command or 100, if the command is used for the first time. Note, however, that even if you specify S100, it is possible that the maximum speed of the motors will be much lower than 720°/s, if the `CFGANG` command has been used.

You may rightfully ask why should the `MoveJ` command be used, when you are exclusively interested in the position of the TCP. Well, firstly, the only reliable way to move DexTAR to its home configuration, before ending a program, is by using the `MoveJ` command (see any of the examples in this chapter). Secondly, the most reliable way to make DexTAR switch to a given working mode is by using the `MoveJ` command. To do this, you must use DexTAR Sim, jog DexTAR to the desired TCP position, check the desired working mode, copy the resulting θ_1 and θ_2, and then use the `MoveJ` command. Finally, the only way to move DexTAR to a Type 2 singular configuration is to by using the `MoveJ` command, and the procedure just described.

3.2.10 MoveC

The `MoveC` command ("C" for Cartesian) is one of the most sophisticated commands in Mecaprol and does not have an equivalent in other robot programming languages. Yet, this command is extremely easy to use. Just specify the desired TCP position (x and y coordinates only) with respect to the WCS and DexTAR will automatically choose the optimal path to get to this position (if possible), in the optimal working mode, and while keeping the current assembly mode. Indeed, for a given TCP position, there may be up to four feasible configurations, some corresponding to one assembly mode, and some possibly corresponding to the other assembly mode. The configuration that is in the current assembly mode, both closest to reach and sufficiently far from Type 2 singularities will be automatically selected as a destination. However, it is possible to force the use a particular working mode.

The `MoveC` command has four arguments:

`X[value]` x coordinate (in mm) of the final TCP position with respect to the WCS;

`Y[value]` y coordinate (in mm) of the final TCP position with respect to the WCS;

`M[value]` the desired working mode for the final configuration, where 1 corresponds to the $++$ working mode, 2 to the $+-$ one, 3 to the $-+$ one, and 4 to the $--$ working mode;

`S[value]` maximum horizontal speed of the TCP (in %), where 100 corresponds to the maximum horizontal speed of the TCP defined by the `CFGANG` command.

None of these four arguments is required, but you should obviously use at least one of the first two. The default values for the first two arguments are the x and y coordinates of

the current TCP position. The default value for the fourth argument is the value defined in the last `MoveC` command or 100, if the command is used for the first time. Finally, if the fourth argument is not used, the optimal working mode will be automatically chosen.

3.2.11 AsyMode

The `AsyMode` command is used to indicate to DexTAR's controller that the movement that was just performed by DexTAR is supposed to have changed the robot's assembly mode. Recall that DexTAR is not equipped with an additional sensor that detects its current assembly mode. In other words, you have to indicate that an assembly mode change is about to occur. The `AsyMode` command has no arguments. Example 3.3 illustrates the use of the AssyMode command.

Exemple 3.3: A Mecaprol program illustrating the use of the AssyMode command

```
! The electromagnetic tool and the acrylic plate with holes must be installed.
! This program makes DexTAR pick a ball from the front of its workspace and
! deposit it on the back, through an assembly mode change.

START;

CFGLIN V40.0 A10000.0; ! Speed and acceleration settings for MoveL, etc.
CFGANG V360.0 A10000.0; ! Speed and acceleration settings for MoveJ and MoveC.

! Definition of the world coordinate system
WCS X0.0 Y0.0 Z-2.50 W0.0; ! Z-2.50, because of the holes
! Definition of the tool coordinate system
TCS Z-39.00;

! Pick the ball
MoveC X0.00 Y162.25 M1; ! pick the ball in the ++ working mode
MoveZ Z0.00 S100;
Output ON;
MoveZ Z12.00 S100;

MoveJ A110.1 B69.9 S100; ! Go to a Type 2 singularity
AssyMode; ! Indicate that you are about to change the assembly mode
MoveJ A90.00 B90.00 S100; ! this is supposed to make the switch
MoveC X0.00 Y-162.25 S100;  ! place the ball
MoveZ Z0.00 S100;
Output OFF;
MoveZ Z12.00 S100;
MoveJ A90.00 B90.00 S100; ! Go to a Type 2 singularity
MoveJ A110.1 B69.9 S100;
AssyMode;
MoveJ A75.00 B95.0;

! Move DexTAR to its home configuration
MoveZ L0.0;
MoveJ A0.0 B0.0;
END;
```

3.2.12 MoveZ

As you have certainly guessed by now, the `MoveZ` command makes DexTAR move its TCP up and down. The `MoveZ` command has three arguments, the first two of which are mutually exclusive, and the third optional:

`Z[value]` coordinate z (in mm) of the end TCP position with respect to the WCS;

`L[value]` coordinate z (in mm) of the end TCP position with respect to the MICS, in the range $[0, -19]$;

`S[value]` maximum linear speed (in %) of the linear actuator, where 100 corresponds to 118 mm/s.

The default value for the third argument is the value defined in the last `MoveZ` command or 100, if the command is used for the first time. Recall that the speed of the linear actuator can only be defined by the `MoveZ` command.

3.2.13 Output

The `Output` command is used to activate or deactivate the electromagnet tool, or any other electric tool that you might decide to develop. The `Output` command has two mutually exclusive arguments:

`ON` current starts to flow to the electromagnet tool;

`OFF` current stops flowing to the electromagnet tool.

3.2.14 WaitTime

The `WaitTime` command pauses temporary the robot for a given period of time. After that period, the program continues automatically. The command has only one argument:

`T[value]` duration of the pause (in ms).

For example, the `WaitTime` command can be used at the beginning of a program, to allow you to remove your hand from the touch screen interface, in case you are filming your demo.

3.2.15 Halt

The `Halt` command "freezes" the robot and allows the interlock covers to be removed. Note that normally removing any of the safety covers (or pressing the E-stop button) not only stops the robot but also deactivates the motors. In the case of the `Halt` command, the robot stops but motors are still under control and the robot is rigid.

The `Halt` command is equivalent to pressing the STOP button in the Auto Mode window on the touch-screen interface. To continue the normal execution of the program, set the interlock covers in place and press the START button in the same window.

Chapter 4

Theory

You don't need to read this chapter in order to be able to use DexTAR. However, if you grasp most of the information that this chapter presents, you will be able to make full use of DexTAR and probably even make your own one. In this chapter, we will present all the equations needed to describe the motion of DexTAR, or more precisely the five-bar planar mechanism part only. In other words, we will ignore DexTAR's linear actuator and study only the motion in the xy plane of the base coordinate system (BCS). We will first present the inverse kinematics, which is concerned with finding the required active-joint angles for a desired TCP position. Then, we will describe the direct kinematics, which is the problem of finding the TCP position for given active-joint angles. Finally, we will study the singularities and the workspace of DexTAR and reveal the most important characteristics of DexTAR.

4.1 Inverse Kinematics

Figure 4.1 shows a schematic of DexTAR that will be used to illustrate the solution of its inverse kinematics. The Oxy reference frame is the base coordinate system (BCS) that was defined in the previous chapter (Fig. 3.2). Points O_1 and O_2 lie on the axes of active revolute joints 1 and 2, respectively. Points A_1 and A_2 lie on the axes of the intermediate passive revolute joints in arm 1 and 2, respectively. Finally, point C lies on the axis of the revolute joint between the distal links and defines the tool center point (TCP). Active-joint angles θ_1 and θ_2 are the angles between the x-axis and distal links 1 and 2, respectively (Fig. 4.1).

The lengths of the distal and proximal links are equal and denoted by ℓ, where $\ell = 90\ mm$. The distance between O_1 and O_2 is denoted by d, where $d = 118\ mm$.* As already mentioned, the origin of the BCS, O, is halfway between O_1 and O_2 and the x axis of the BCS passes through O_1 and O_2. (Recall that active joint 1 is closest to the E-stop button).

In the inverse kinematics we know the position of C, with coordinates x and y with respect to the BCS, and we look for the active-joint angles θ_1 and θ_2. It is easy to see that this problem consists of solving independently the inverse kinematics of arms 1 and 2. Therefore, for brevity, we will only present the solution for arm 2.

*Why 118 mm? This is because the shorter the distance, the larger the workspace of the robot, but this distance should be large enough to minimize the mechanical interferences between the proximal links. Well, we later realized that 120 mm would have been a better choice from a usability point of view...

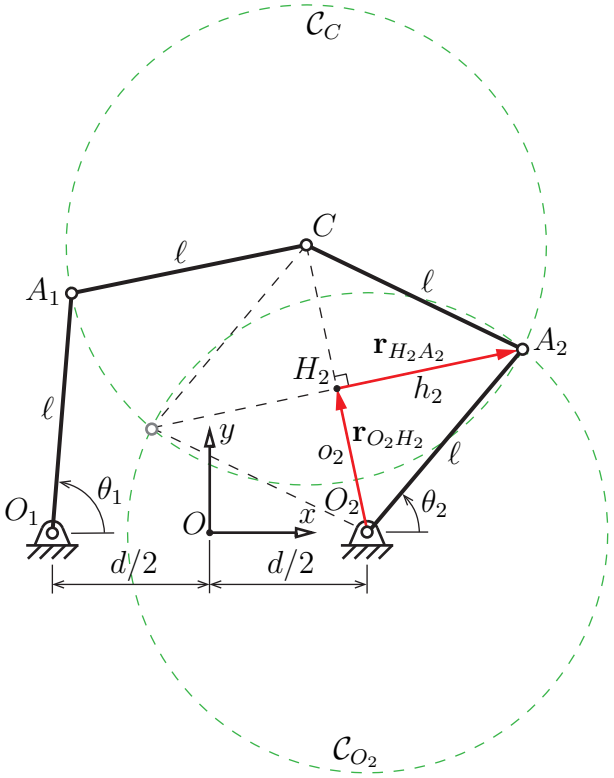


Figure 4.1: Schematic illustrating the solution of the inverse kinematics for arm 2.

In the kinematic analysis of mechanisms, it is very important to interpret each problem geometrically. In our case, the inverse kinematics for arm 2 is identical to the problem of finding the intersection points between circle $\mathcal{C}_C$, centred at C and of radius ℓ, and circle $\mathcal{C}_{O_2}$, centred at O_2 and of radius ℓ. These intersection points are the possible positions for point A_2. In general, there are two such intersection points and these can be found as follows.

Let us denote by H_2 the midpoint between C and O_2, and let o_2 be the length of segments O_2H_2 and H_2C, and h_2 be the length of segment H_2A_2. Obviously, the line segment H_2A_2 is normal to the line segment O_2H_2. Now, let vector $\mathbf{r}_{OC}$ represent the vector connecting point O to point C, whose coordinates are expressed in the BCS (i.e., $\mathbf{r}_{OC}$ represents the position of the TCP with respect to the BCS). Similarly, vector $\mathbf{r}_{OO_2}$ represents the vector connecting point O to point O_2, whose coordinates are expressed in the BCS, vector $\mathbf{r}_{O_2H_2}$ represents the vector connecting point O_2 to point H_2, whose coordinates are expressed in the BCS, and so on.

Referring to Fig. 4.1, we have

$$\mathbf{r}_{O_2A_2} = \mathbf{r}_{O_2H_2} + \mathbf{r}_{H_2A_2}, \tag{4.1}$$

where

$$\mathbf{r}_{O_2H_2} = \frac{1}{2}\left(\mathbf{r}_{OC} - \mathbf{r}_{OO_2}\right) = \frac{1}{2}\left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} d/2 \\ 0 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} x - d/2 \\ y \end{bmatrix}. \tag{4.2}$$

In general, there are two possible positions for point A_2. In other words, vector $\mathbf{r}_{H_2A_2}$ can be obtained by rotating vector $\mathbf{r}_{O_2H_2}$ 90° clockwise or 90° counterclockwise, normalizing it,

and then multiplying the resulting vector by the length of segment H_2A_2 (i.e., h_2). Thus,

$$\mathbf{r}_{H_2A_2} = -\delta_2 \frac{h_2}{o_2} \mathbf{E}\, \mathbf{r}_{O_2H_2}, = -\delta_2 \frac{h_2}{o_2} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_{O_2H_2}, \tag{4.3}$$

where $\mathbf{E}$ is the matrix that rotates vectors 90° counterclockwise and

$$h_2 = \sqrt{\ell^2 - o_2^2} \tag{4.4}$$

and δ_2 is the *branch index* of arm 2, with $\delta_2 = +1$ if A_2 is on the right side of vector $\mathbf{r}_{O_2C}$ and $\delta_2 = -1$ if A_2 is on the left side of vector $\mathbf{r}_{O_2C}$.

Therefore, we have

$$\theta_2 = \operatorname{atan2}\left(y + \delta_2 \frac{\sqrt{\ell^2 - o_2^2}}{o_2}\big(d/2 - x\big), x - d/2 + \delta_2 \frac{\sqrt{\ell^2 - o_2^2}}{o_2} y \right). \tag{4.5}$$

Similarly, we can show that

$$\theta_1 = \operatorname{atan2}\left(y - \delta_1 \frac{\sqrt{\ell^2 - o_1^2}}{o_1}\big(d/2 + x\big), x + d/2 + \delta_1 \frac{\sqrt{\ell^2 - o_1^2}}{o_1} y \right). \tag{4.6}$$

where δ_1 is the *branch index* of arm 1, with $\delta_1 = +1$ if A_1 is on the right side of vector $\mathbf{r}_{O_1C}$ and $\delta_1 = -1$ if A_1 is on the left side of vector $\mathbf{r}_{O_1C}$.

The set $\{\operatorname{sign}(\delta_1), \operatorname{sign}(\delta_2)\}$ defines that we call the *working mode*: $++$, $+-$, $-+$, or $--$. For example, DexTAR is in the $-+$ working mode in Fig. 4.1, while its home configuration, illustrated in Fig. 3.1(a), is in the $++$ working mode.

To conclude this section, let's see if there are always two solutions for each arm. Indeed, what if $o_2 = 0$ or $o_2 \geq \ell$? If $o_2 > \ell$, this means that circles $\mathcal{C}_C$ and $\mathcal{C}_{O_2}$ do not intersect, because the distance between points C and O_2 is larger than 2ℓ. In other words, the desired TCP position is outside DexTAR's workspace.

We have $o_2 = 0$ if and only if points C and O_2 coincide, which can physically occur in DexTAR, when arm 2 is fully folded. In that case, circles $\mathcal{C}_C$ and $\mathcal{C}_{O_2}$ coincide. When this happens, arm 2 can rotate indefinitely about point O_2, but the position of the TCP does not change. (Well, in practice it can't make a full rotation because of mechanical interferences). This means that any angle is a solution for θ_2.

Similarly, we have $o_2 = \ell$ if and only if the distance between points C and O_2 is equal to 2ℓ, which can occur in DexTAR when arm 2 is fully stretched. In that case, circles $\mathcal{C}_C$ and $\mathcal{C}_{O_2}$ are tangent, and we have a single solution for θ_2.

The same reasoning goes for arm 1. In conclusion, if an arm is fully stretched, there is only one solution for the corresponding active-joint angle, while if an arm is fully folded, any values is a solution for the corresponding active-joint angle (at least in theory). Configurations in which an arm is fully stretched or fully folded are called *Type 1 singularities*. As we already saw in Section 1.2, and as we will see in Section 4.3, crossing Type 1 singularities, allows DexTAR to switch working mode and thus access a much larger workspace.

Finally, let's just specify for the sake of completeness that at a Type 1 singularity, we can't say that DexTAR is in one working mode or in another. It is actually on the boundary between two, or even four, working modes.

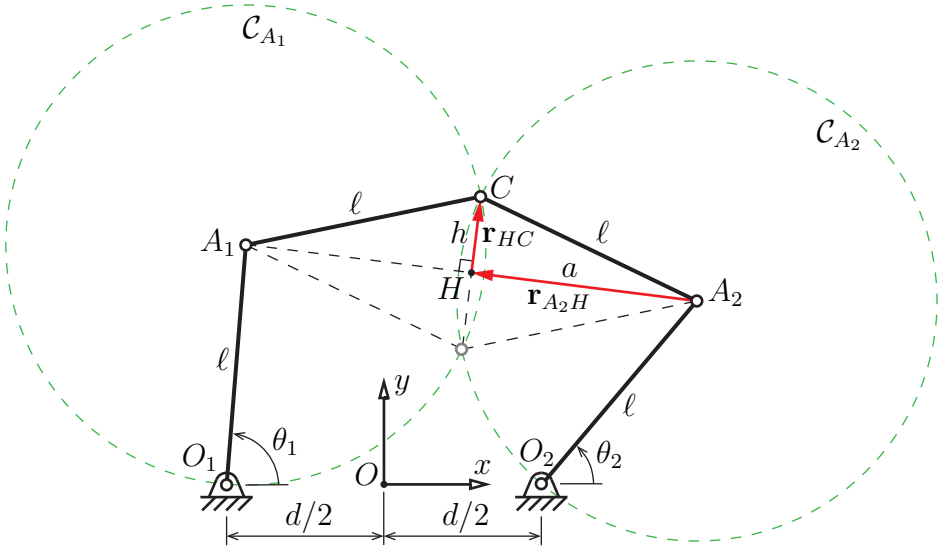


Figure 4.2: Schematic illustrating the solution of the direct kinematics.

4.2 Direct Kinematics

Figure 4.2 shows a schematic of DexTAR that will be used to illustrate the solution of its direct kinematics. In the direct kinematics we know the active-joint angles θ_1 and θ_2 and we look for the position of C, with coordinates x and y with respect to the BCS.

In a five-bar mechanism, the solution of the direct kinematics is very similar to the solution of the inverse kinematics. Essentially, the problem consists in finding the intersection points between circle $\mathcal{C}_{A_1}$, centred at A_1 and of radius ℓ, and circle $\mathcal{C}_{A_2}$, centred at A_2 and of radius ℓ. These intersection points are the possible positions for point C. In general, there are two such intersection points and these can be found as follows.

Let us denote by H the midpoint between A_1 and A_2, and let h be the length of segment HC, and a be the length of segments HA_1 and HA_2. Obviously, the line segment HC is normal to the line segment A_1A_2. Now, let vector $\mathbf{r}_{A_1H}$ represent the vector connecting point A_2 to point H, whose coordinates are expressed in the BCS. Similarly, vector $\mathbf{r}_{HC}$ represents the vector connecting point H to point C, whose coordinates are expressed in the BCS.

Referring to Fig. 4.2, we have

$$\mathbf{r}_{OC} = \mathbf{r}_{OO_2} + \mathbf{r}_{O_2A_2} + \mathbf{r}_{A_2H} + \mathbf{r}_{HC} = \mathbf{r}_{OA_2} + \mathbf{r}_{A_2H} + \mathbf{r}_{HC}, \tag{4.7}$$

where

$$\mathbf{r}_{A_2H} = \frac{1}{2}\left(\mathbf{r}_{OA_1} - \mathbf{r}_{OA_2}\right), \tag{4.8}$$

$$\mathbf{r}_{OA_1} = \begin{bmatrix} -d/2 + \ell\cos\theta_1 \\ \ell\sin\theta_1 \end{bmatrix}, \tag{4.9}$$

$$\mathbf{r}_{OA_2} = \begin{bmatrix} d/2 + \ell\cos\theta_2 \\ \ell\sin\theta_2 \end{bmatrix}, \tag{4.10}$$

In general, there are two possible positions for point C. In other words, vector $\mathbf{r}_{HC}$ can be obtained by rotating vector $\mathbf{r}_{A_2H}$ 90° clockwise or 90° counterclockwise, normalizing it, and then multiplying the resulting vector by the length of segment HC (i.e., h). Thus,

$$\mathbf{r}_{HC} = -\gamma \frac{h}{a} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_{A_2H}, \tag{4.11}$$

where

$$h = \sqrt{\ell^2 - a^2} \tag{4.12}$$

and γ is the *assembly mode index*, with $\gamma = +1$ if C is on the right side of vector $\mathbf{r}_{A_2A_1}$ and $\gamma = -1$ if C is on the left side of vector $\mathbf{r}_{A_2A_1}$. For example, DexTAR is in the positive assembly mode in Fig. 4.2.

Thefore, we have

$$x = \frac{1}{2}(-d + \ell\cos\theta_1 + \ell\cos\theta_2) + \frac{1}{2}\gamma\frac{\sqrt{\ell^2 - a^2}}{a}(\ell\sin\theta_1 - \ell\sin\theta_2), \tag{4.13}$$

$$y = \frac{1}{2}(\ell\sin\theta_1 + \ell\sin\theta_2) + \frac{1}{2}\gamma\frac{\sqrt{\ell^2 - a^2}}{a}(d - \ell\cos\theta_1 + \ell\cos\theta_2). \tag{4.14}$$

To conclude this section, let's see if there are always two solutions. Indeed, what if $a = 0$ or $a \geq \ell$? If $a > \ell$, this means that circles $\mathcal{C}_{A_1}$ and $\mathcal{C}_{A_2}$ do not intersect, because the distance between points A_1 and A_2 is larger than 2ℓ. In other words, the mechanism cannot be assembled for the given active-joint angles. For example, for $\theta_1 = 0$ and $\theta_2 = 180°$, there is obviously no solution to the direct kinematics.

We have $a = 0$ if and only if points A_1 and A_2 coincide, which can never occur in practice in our DexTAR, because of interferences between the proximal links. However, if this would have been possible, then the TCP will freely move along a circle of radius ℓ, even though the motors do not rotate.

Similarly, we have $a = \ell$ if and only if the distance between points A_1 and A_2 is equal to 2ℓ, which can occur in DexTAR when the distal links are collinear. In that case, circles $\mathcal{C}_{A_1}$ and $\mathcal{C}_{A_2}$ are tangent, and we have a single solution for x and y.

These two special configurations ($a = 0$ and $a = \ell$) are called *Type 2 singularities*. As we already saw in Section 1.2, and as we will see in Section 4.3, crossing Type 2 singularities, allows DexTAR to switch assembly mode and thus access a much larger workspace.

Finally, let's just specify for the sake of completeness that at a Type 2 singularity, we can't say that DexTAR is in one assembly mode or in another. It is actually on the boundary between both assembly modes.

4.3 Velocity Kinematics

The term *volesty kinematics* refers to the relationships between the velocities of bodies and joints in a mechanism. More specifically, in this section, we are interested in the relationship between the active-joint angular rates ($\dot{\theta}_1$ and $\dot{\theta}_2$) and the TCP velocity ($\dot{x}$ and $\dot{y}$). This relationship leads to a physical interpretation of the concept of singularities.

Let us start with the equation for the distal link vector i ($i = 1, 2$):

$$\mathbf{r}_{A_iC} = \mathbf{r}_{OC} - \mathbf{r}_{OO_i} - \mathbf{r}_{O_iA_i}. \tag{4.15}$$

Taking the Euclidean norm of both sides of this equation yields

$$\mathbf{r}_{A_iC}^T \mathbf{r}_{A_iC} = (\mathbf{r}_{OC} + \mathbf{r}_{OO_i} - \mathbf{r}_{O_iA_i})^T (\mathbf{r}_{OC} + \mathbf{r}_{OO_i} - \mathbf{r}_{O_iA_i}) = \ell^2. \tag{4.16}$$

Now, we differentiate the above equation with respect to time which leads to

$$2\mathbf{r}_{A_iC}^T (\dot{\mathbf{r}}_{OC} - \dot{\mathbf{r}}_{O_iA_i}) = 0. \tag{4.17}$$

Rearranging this equation and evaluating the time derivative of vector $\mathbf{r}_{O_iA_i}$ gives

$$2\ell \mathbf{n}_i^T \dot{\mathbf{r}}_{OC} = 2\ell \mathbf{n}_i^T \mathbf{r}_{O_iA_i}^{\perp}, \tag{4.18}$$

where $\mathbf{r}_{O_iA_i}^{\perp}$ is the vector obtained by rotating $\mathbf{r}_{O_iA_i}$ 90° counterclockwise,

$$\mathbf{r}_{O_iA_i}^{\perp} = \mathbf{E}\ell \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{r}_{O_iA_i} = \ell \begin{bmatrix} -\sin\theta_i \\ \cos\theta_i \end{bmatrix}, \tag{4.19}$$

and $\mathbf{n}_i$ is the unit vector along $\mathbf{r}_{A_iC}$.

Finally, Eq. (4.18) can be written in matrix form as:

$$\begin{bmatrix} \mathbf{n}_1^T \\ \mathbf{n}_2^T \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \mathbf{n}_1^T \mathbf{r}_{O_1A_1}^{\perp} & 0 \\ 0 & \mathbf{n}_2^T \mathbf{r}_{O_2A_2}^{\perp} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}. \tag{4.20}$$

We will denote by $\mathbf{Z}$ the 2×2 matrix that premultiplies the TCP velocity vector in the above equation, and by $\mathbf{\Lambda}$ the 2×2 diagonal matrix that premultiplies the active-joint rates vector.

Now note that matrix $\mathbf{Z}$ is singular if and only if vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ (i.e., the distal links) are collinear. This is the condition for what we called Type 2 singularity in the previous section. We saw that at a Type 2 singularity, the direct kinematic problem degenerates. In terms of velocities, we can now also state that in a Type 2 singularity, even if all actuators are locked (i.e., if $\dot{\theta}_1 = \dot{\theta}_2 = 0$, there may exist a nonzero TCP velocity vector. We therefore say that in a Type 2 singularity, the TCP gains a degree of freedom. Thus, in practice, due to joint clearances (however small they are in DexTAR), in a Type 2 singularity, the TCP is not fully constrained.

Matrix $\mathbf{\Lambda}$ is singular if and only if $\mathbf{n}_1^T \mathbf{r}_{O_1A_1}^{\perp} = 0$ or $\mathbf{n}_2^T \mathbf{r}_{O_2A_2}^{\perp} = 0$, i.e., if and only if the proximal and distal links in arm 1 or 2 are collinear. This is the condition for what we called Type 1 singularity in the previous section. We saw that at a Type 1 singularity, the inverse kinematic problem for the arm at singularity degenerates. In terms of velocities, we can now also state that in a Type 1 singularity, even if the velocity of the active joint of the arm at singularity is infinitely large, the TCP velocity may be zero. We therefore say that in a Type 1 singularity, the DexTAR's TCP loses one or two degrees of freedom.

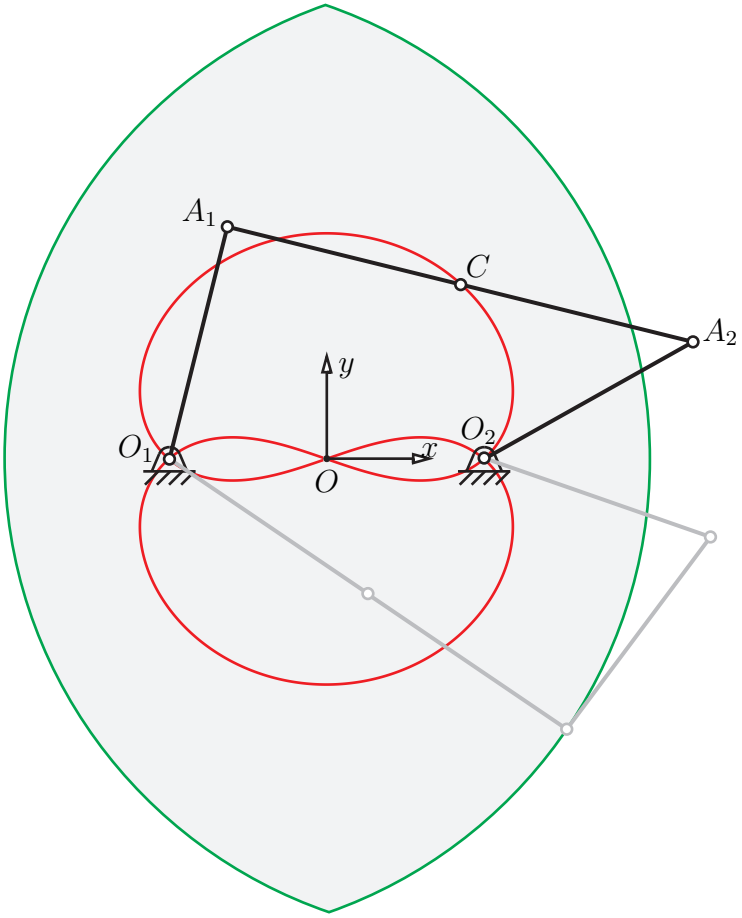


Figure 4.3: Complete workspace (grey area) and Type 2 singularity loci (in red), excluding those corresponding to configurations where the distal links overlap, which are physically impossible).

4.4 Workspace and Singularity Loci

There are several ways of explaining why Type 2 singular configurations are troublesome, but consider this simplest explanation. Imagine your DexTAR has both distal links aligned (i.e., point C lies on A_1A_2, right in the middle of that line segment of length 2ℓ), as shown in Fig. 4.3. Now suppose you want to get away from this configuration by moving both motors in opposite directions. You can't; the robot is jammed.

Therefore, it is of no surprise that there does not exist any other commercial parallel robot that can be in a Type 2 singularity. Indeed, commercial parallel robots are designed in such a way that mechanical interferences and actuator limits prevent the occurrence of such singularities. For example, in the five-bar robot built by Mitsubishi Electric (Fig.1.3(b)), it is impossible to align the distal links.

DexTAR, however, can cross not only Type 1 singularities but Type 2 singularities too. The robot can switch its assembly mode by swinging its distal arms though a Type 2 singularities and using inertia to to the flip. This means that while DexTAR can stop in a Type 2 singular configuration it cannot start from such a configuration.

Figure 4.3 illustrates DexTAR's *complete workspace*, drawn in grey. This area is the intersection of two circles of radius 2ℓ, centred at O_1 and O_2. Its boundaries, drawn in green, correspond to TCP positions where one of both arms are fully stretched, i.e., they correspond to Type 1 singularities. DexTAR is shown in one such Type 1 singular configuration, drawn in grey. Points O_1 and O_2 also correspond to Type 1 singularities. When the TCP is at one of these points, one of the arms is fully folded. In theory, since the distal and proximal arms are of the same length, we have no holes in the complete workspace. In practice, the proximal and distal links are of slightly different lengths (by only a couple of micrometers), so there are tiny holes (of radius a few micrometers) in the complete workspace centred at points O_1 and O_2. Of course, we will ignore these.

The red curve in Fig. 4.3 corresponds to Type 2 singularity loci, i.e., to TCP positions where DexTAR is at a Type 2 singularity. Only configurations where the distal links are collinear are considered, since configurations where these two links overlap are not feasible in practice. This red curve can be expressed as a polynomial in x and y of order 6, i.e., a *sextic*. It can also be expressed in parametric form as

$$\begin{cases} x &= \rho\cos\vartheta \\ y &= \rho\sin\vartheta \end{cases} \qquad 0 \leq \vartheta \leq 2\pi \tag{4.21}$$

where

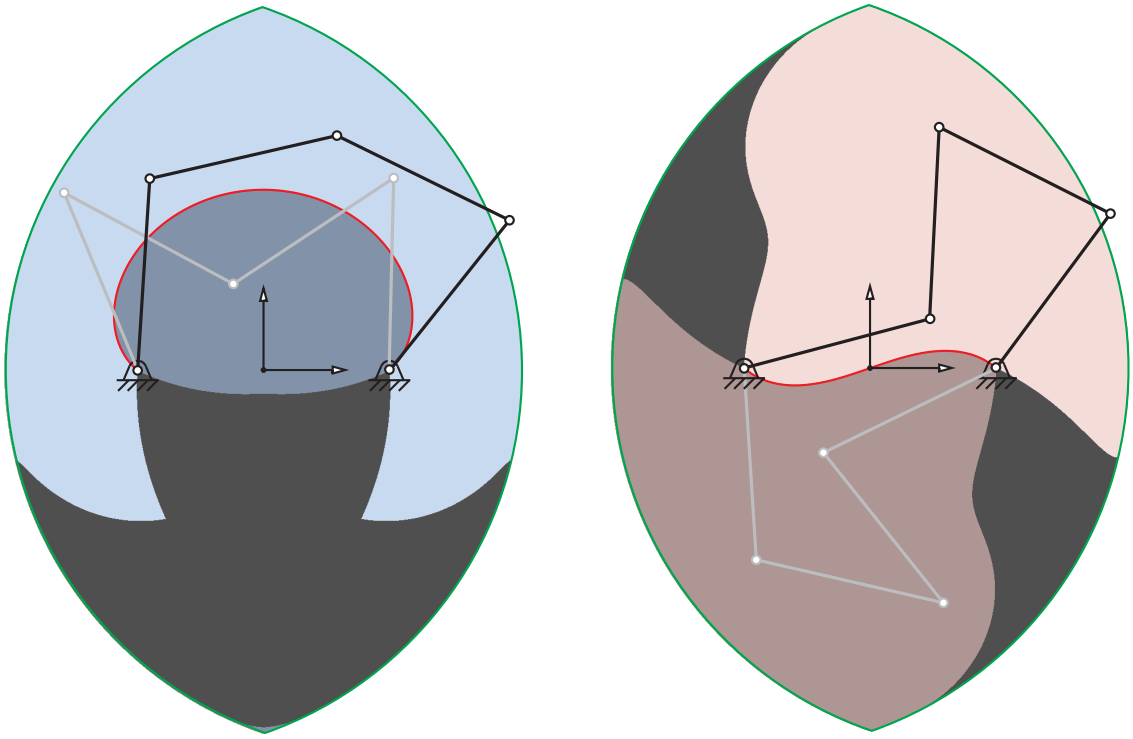
$$\rho = \frac{1}{2}\sqrt{d^2(2\cos^2\vartheta - 1) \pm 2d\sin\vartheta\sqrt{4\ell^2 - d^2\cos^2\vartheta}}. \tag{4.22}$$

The procedure for obtaining the above expressions will not be explained here.

As you have probably noticed, DexTAR's distal links, including the linear actuator mounted on one of them, can never interfere with any other part of the robot. Well, the cable of the electromagnet would get in the way if the distal links were able to spin 360° with respect to one other, but there is something else preventing this from happening. Furthermore, it is important to point out that DexTAR uses slip rings at joints O_1, O_2, A_1 and A_2, and these can spin endlessly. However, DexTAR's proximal links can hit each other. Thus, the condition for a mechanical interference in DexTAR is when the shortest distance between segments O_1A_1 and O_2A_2 is less than or equal to 26 mm, which is the width of the proximal links.

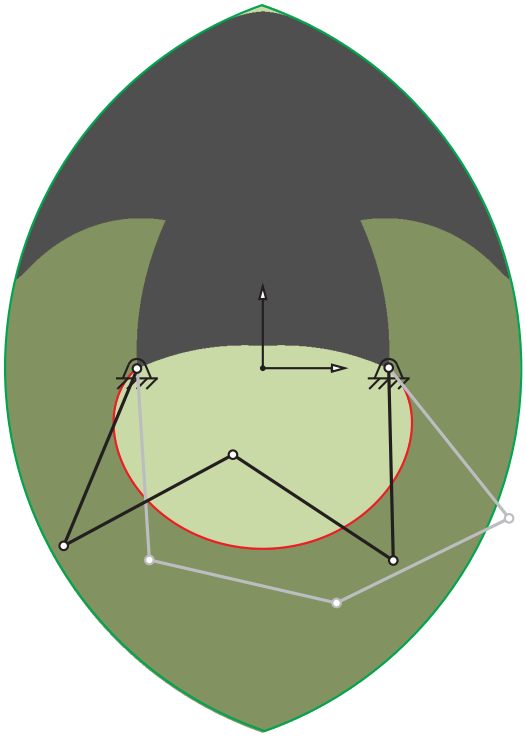
Figure 4.4 shows DexTAR's actual Cartesian workspace for each of the four working modes. In each of the four plots, the area or areas in dark grey correspond to mechanical interferences. As in Fig. 4.3, in each plot, the red curve corresponds to Type 2 singularity loci. This curve separates two coloured areas: the lighter one corresponds to the positive assembly mode, while the darker one corresponds to the negative assembly mode. For more clarity, in each plot, DexTAR is illustrated in black in the positive assembly mode and in light grey in the negative assembly mode. Finally, note that in working modes $-+$ and $+-$, the area of mechanical interferences separates one of the coloured areas into two, one of which is a tiny patch of little practical importance.

Now the most important fact is that if we superpose all four plots and take the union of all accessible areas, the result would be the complete workspace shown in Fig. 4.3. In other words, since DexTAR can switch working modes and assembly modes, the mechanical interferences do not precent it from attaining any point inside the complete lens-shaped area.

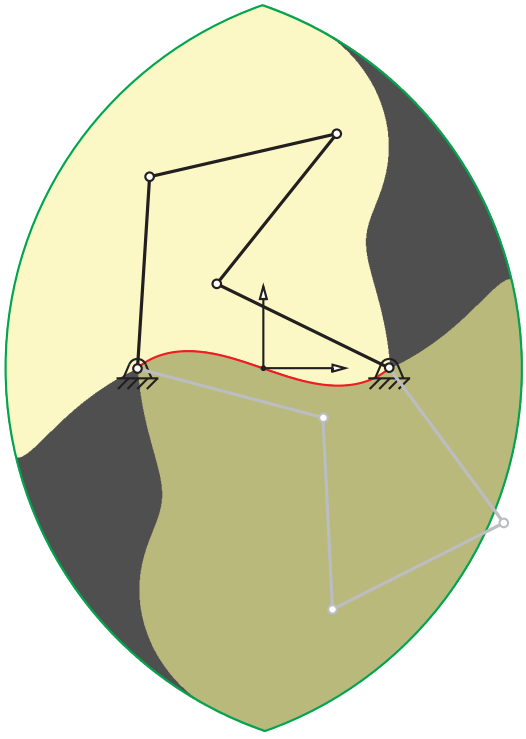


(a) working mode −+

(b) working mode ++

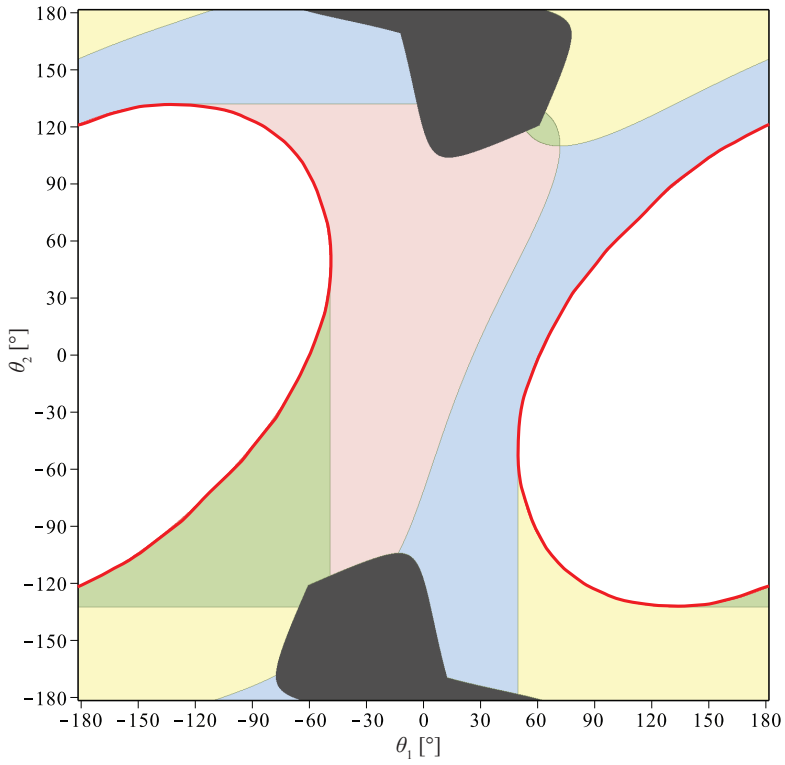


(c) working mode +−

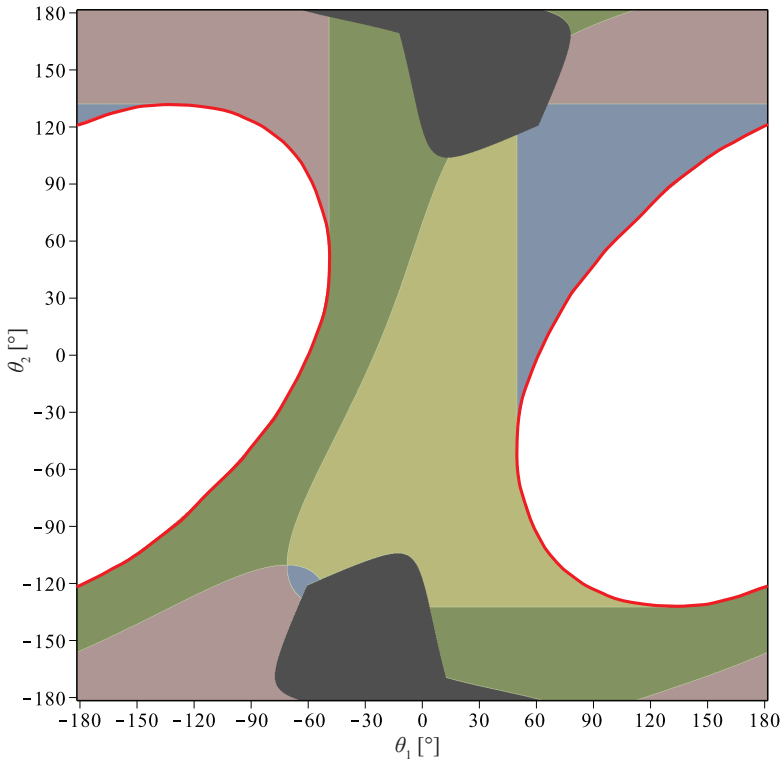


(d) working mode −−

Figure 4.4: Workspace area for each of the four working modes. In each plot, the dark grey area(s) correspond(s) to mechanical interferences, the light coloured area(s) correspond(s) to the positive assembly mode, and the dark coloured one(s) to the negative assembly mode.



(a) assembly mode +



(b) assembly mode −

Figure 4.5: Active-joint space for each of the two assembly modes.

We will conclude this section with a plot of DexTAR's *active-joint space*. For each pair $\{\theta_1, \theta_2\}$, there generally exist two possible TCP positions, corresponding to the two assembly modes. Therefore, we need to plot one active-joint space for each assembly mode.

Figure 4.5 shows DexTAR's active-joint space for each of the two assembly modes for $\theta_1 \in [-180°,\ 180°]$ and $\theta_2 \in [-180°,\ 180°]$. Because of the nature of angles, the best mapping for the active-joint space would be a torus, but this is obviously difficult to represent on paper. We have therefore settled for a Cartesian plot. However, keep in mind that each plot is a piece of a mosaic (e.g., 361° is the same as 1°).

Note that we use the same colour coding as in Fig. 4.4. Namely, the red curves (that look like parts of ellipses but are not) correspond to Type 2 singularities. Crossing them, basically brings you from one of the subfigures to the other. In each subfigure, the coloured areas correspond to a particular working mode, as illustrated in Fig. 4.4. The green curves (and lines) correspond to Type 1 singularity loci. The dark gray areas correspond to mechanical interferences. Finally, the white areas correspond to configurations where DexTAR cannot be assembled (the distance between points A_1 and A_2 is greater than 2ℓ.

To gain a better understanding of DexTAR's workspace, jog the robot in DexTAR Sim and observe the TCP position and active-joint angles in the Cartesian workspace and active-joint space mappings. Note that you can jog the robot directly by sliding the current position/configuration in the workspace map.

To conclude this section, let's just recall that for a desired TCP position, you can have up to four possible configurations (some of the theoretically possible ones might not be feasible in practice due to mechanical interferences). Each of these configurations will correspond to a different working mode, and some might even correspond to a different assembly mode. Choosing the right configuration for the desired TCP position, in order to minimize the time for the displacement is an intricate research topic that was studied in [9], but is still open. Would you be interested in investigating this problem?

References

[1] Pollard Jr., W.L.G., *Spray Painting Machine*, US Patent 2,213,108, filed October 29, 1934, issued August 27, 1940.

[2] Makino, H., Kato, A., and Yamazaki, Y., "Research and commercialization of SCARA robot," *International Journal of Automation Technology*, vol. 1, no. 1, pp. 61–62, 2007.

[3] *World Robotics – Industrial Robots*, IFR Statistical Department, 2014.

[4] Makino, H., *Assembly Robot*, US Patent 4,341,502, filed March 24, 1980, issued July 27, 1982.

[5] Fyler, D.C., *Control Arm Assembly*, US Patent 4,712,971, filed February 13, 1985, issued December 15, 1987.

[6] Tavkhelidze, D.S., and Davitashvili, N.S., "Kinematic analysis of five-link spherical mechanisms", *Mechanism and Machine Theory*, vol. 9, no. 2, pp. 181–190, 1974.

[7] Asada, H., and Youcef-Toumi, K., "Analysis and design of a direct-drive arm with a five-bar-link parallel drive mechanism", *Journal of Dynamic Systems, Measurement, and Control*, vol. 106, no. 3, pp. 225–230, 1984.

[8] Campos, L., Bourbonnais, F., Bonev, I.A., and Bigras, P., "Development of a five-bar parallel robot with large workspace," *ASME 2010 International Design Engineering Technical Conferences*, Montreal, QC, Canada, August 15–18, 2010.

[9] Bourbonnais, F., Bigras, P., and Bonev, I.A., "Minimum-time trajectory planning and control of a reconfigurable pick-and-place parallel robot," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 20, pp. 740–749, 2014.