11 11 TEX LaTEX

DataViewer's User Guide

Dana Vrajitoru, dana.vrajitoru@epfl.ch v0.2, Mon Oct 30 10:55:09 GMT 2000

This is the user documentation for DataViewer. This manual provides a user guide for all the features of DataViewer that are common to all its applications. More information on this software may be found at *http://lcvmwww.epfl.ch/DV*.

Contents

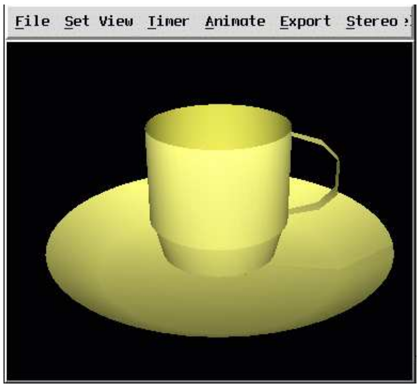

Figure 1:A DataViewer window.

1 Introduction

DataViewer (see 1 (Figure)) is a scientific visualization library created to allow the easy development of efficient visualization programs for those who are not graphics programming experts. It consists of a high-level set of routines for rendering geometric objects and utilizes the OpenGL graphics API for its low level rendering. DataViewer has been used for several different visualization tasks, e.g. most of *VBM*'s visualization modules have been developed using it. The goals that have driven the design of DataViewer are to be efficient, easy to use, portable, closely connectable to numerical computations, and available free of charge to the mathematical and scientific community.

2 Running a DataViewer Application

This section presents the general principles for executing DataViewer, introduces the applications developed in the LCVM2 laboratory and gives a short explanation for writing a new application.

2.1 Running a Demo

To run a DataViewer demo, you must be in the main directory of Dataviewer. For example, if DataViewer is installed under your `/usr/local/` directory, then you should move to

`/usr/local/DataViewer/`

Then run the `runme.sh` script. With most shells, typing the name of the script is enough. Otherwise, you can type

`sh runme.sh`

This should open an empty DataViewer window, similar to 5 (Figure). Call the `File/User` menu, which should activate the dialog shown in 2 (Figure).

Using the "Open Data File ..." button, open the file named

`...DataViewer/applications/Python_demo_complex/data/ala_min.crd`

and then hit the "Draw" button. You should now obtain a result as shown in 3 (Figure).

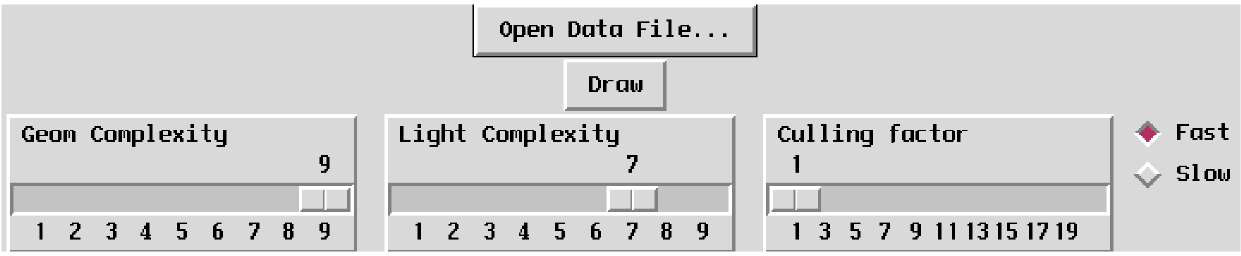

Figure 2: Demo user dialog.

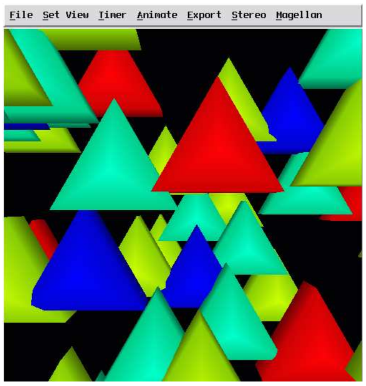

Figure 3:Demo main window.

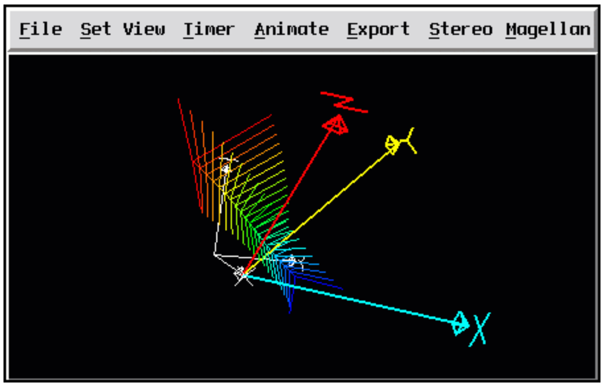

Figure 4:An example of the main window for a DataViewer application.

Play around with the options in the dialog and notice the changes in the image. Remember to press "Draw" after each change. You can also play with the mouse within the main window, and get familiar with the manipulation of the geometrical scene. For more details on this part, see 3 (Section).

2.2 Executing DataViewer

To run a DataViewer application, you need to execute the following command, in which you must replace the quoted parts with something corresponding to your real situation:

"dv_dir/bin/"`DV.py` "dv_dir/python_script/application_script"`.py`

For example, to run the application contained in the demo script, go to the directory `applications/PythonDV_demo_complex/` and run the command

```
bin/DV.py python_script/charmm_crd.py
```

Thus, to run DataViewer you need to execute two python scripts (.py). The first, named DV.py, is the general DataViewer script and usually can be found in the `bin` directory of the application. The second one comes as an argument to the first one, and is different for each application, defining its specific operations (usually linked to the "File/User" menu). It is usually situated in the `python_script` directory of the application.

For example, if you are in the `DataViewer/applications/Python_demo_simple` directory, you can run this application by executing the following command:

```
bin/DV.py python_script/typein.py
```

The command usually opens an empty DataViewer main window (see 5 (Figure)).

To actually draw the scene aimed by the application, you must call the "File/User" menu, that opens an application dialog. In many cases, you have to hit the "Draw" button to visualize a scene similar to 4 (Figure). For more complex applications, other operations may be needed to create the scene. Follow the application's instructions for that part (see 2.3 (Section)).

2.3 DataViewer Applications

Here is a list of the applications that normally come with the DataViewer distribution:

Python_demo_simple

This application, launched by the typein.py python script, allows you to enter the name of the problem you want to treat. The only available problem for this module is the default problem, which does nothing. This demo is an example of how to create your own application.

Python_demo_complex: charmm_crd.py

This python script from the `Python_demo_complex` directory draws a set of spheres that are read from a data file. For example, the `data/ala_min.crd` file contains the simulation of a molecule.

Python_demo_complex: sin_surf.py

This script draws a sinusoidal surface that can be animated.

Python_demo_complex: sphere.py

This script draws 3 spheres.

Python_demo_complex: typein.py

This script allows you to select the problem you want to run and to send the problem an argument string. For example, you can choose the "select_test" problem, the argument string should be an integer number between 1 and 4. This problem will draw 4 spheres with the colors red, green, blue and yellow. The color of the selected sphere (the argument of the problem) will be drawn using a darker shade.

Scene_file: scene_file.py

This python script creates a scene from a scene file. For more information, see the *scene file documentation*.

Scene_file: transfer_file.py

This python script can translate a color map from a povray file to a scene file or the contrary. The povray color map can be initially created with a program like GIMP, and then be transfered to a scene file to be used with the previously described module (scene_file.py).

Other more complex applications of DataViewer are:

VBM

The Visualization of Bifurcation Manifolds is probably the most important application of DataViewer up to now. For more information, see *http://lcvmwww.epfl.ch/VBM*.

Slinky

This application simulates deformations of elastic rods. Complementary to VBM which is specialized on continuation problems, Slinky can solve various initial value problems linked to rods. For more information, see *lcvmsun1.epfl.ch/~jdesiebe/slinky/*.

2.4 Notes for Programmers

The python script of the application must define a class inheriting from the `geom_window.GeomWindow` class, with a `UserWidgets` function inside it. This function must contain the widgets specific to the application. At some point, this class must communicate with a c++ source problem, by calling the `self.geom.set_problem("problem_source_name")` to set the problem name, and `self.geom.file(argument_string)` to transfer the parameters that the c++ function needs.

On the c++ side, you must define a function taking an argument of the type `DVcreate_image_arguments` and returning an integer. The argument send by the python script is a string and can be found in `dv.input_string` . Then any geometrical object created by the application can be added to `dv.top` which is the root of the scene tree and the operation for this action is `dv.top.copy_child(any_object);` . The prototype of the c++ function must be added to `DVpython_problems.h` and the function must also be called from inside the `DVpython_problems` function in the `DVpython_problems.cc` module. Make sure the Makefile of the application has the name of your module in it.

3 Interacting with the Scene

The user can interact with a DataViewer scene using the mouse and the keyboard. The objects displayed in the main window can be rotated, translated and zoomed. Note that

- Once a rotation movement has been imposed on the scene, it keeps gqoing on until it is stopped or another rotation movement is performed.
- Each particular application of DataViewer may define its own scene manipulation operations beside the general ones. Look at the documentation of the application for that.
- The scene can also be controlled by a 6D device called Magellan when available. See the section on the Magellan menu for more details.

Here are the keyboard and mouse bindings for these operations:

3.1 Keyboard bindings

Left and Right keys

Perform rotations about the vertical axis.

Up and Down keys

Perform rotations about the horizontal axis.

Page Up and Page Down keys

Perform rotations about the axis "coming out of the screen".

Control + Left and Right keys

Translate the object horizontally.

Control + Up and Down keys

Translate the object vertically.

Figure 5:DataViewer's main window.

Control + Page Up and Page Down keys

Zoom the object.

3.2 Mouse bindings

Left Mouse Button

This button is usually used by each application for its own purpose.

Hold down the Middle Mouse Button

Performs rotations of the scene based upon the virtual trackball paradigm.

Release the Middle Mouse Button

The scene continues to rotate in the same direction as it was rotating when the button was released.

Click the Middle Mouse Button

The scene stops rotating.

Hold down the Right Mouse Button

Translates the scene based upon the movements of the mouse.

Hold down the Middle Mouse Button with Shift Key held down

As the mouse is moved up the scene is moved closer to the eye-point (zoom in), and as the mouse is moved down the scene is moved further away from the eye-point (zoom out).

4 The Menus

When opening a DataViewer application, you should get the following 5 (Figure).

The next paragraphs describe each of the menus of DataViewer.

Figure 6:The File menu.

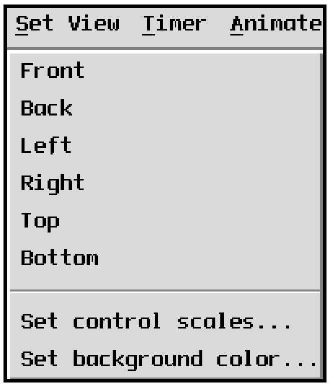

Figure 7:The Set View menu.

4.1 File

The **File** menu has two items: User and Quit.

The **User** item calls the application depend control window. The widgets in this window are defined by each application inside its own python class with help of the UserWidgets() function. See the documentation for each application for more details. Some applications, like *VBM* have disabled this operation and are defining their control windows in a different manner.

The **Quit** item quits DataViewer.

4.2 Set View

The **Set View** menu contains two groups of items, as is it shown in 7 (Figure).

The first group, containing 6 items, allows you to set the view camera to 6 predefined positions. The location and direction of the camera for each of these positions is the following:

The **Set control scales ...** calls the dialog in 8 (Figure).

In this window, the first 3 typeins define the speed of response of the DataViewer scene to the mouse and keyboard actions. Basically, they define the range of change to be done in one key stroke. You may have to change this if you modify the perspective angle.

Axis \ Direction	Positive	Negative
X	Front	Back
Y	Left	Right
Z	Top	Bottom

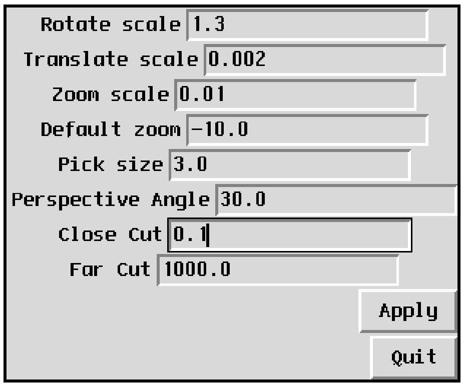

Figure 8:The Set Control Scales window.

The **Default zoom** is the zoom of the scene when the camera position is chosen using the 6 predefined positions (front, back, ...).

The **Pick size** sets the error tolerance for selecting regions in the scene with the mouse. It is only used by those applications of DataViewer that have implemented a selection interaction with the scene.

For example, for *VBM*, the pick size defines the size of the region used for mouse selections. I.e. it is the amount by which you can miss exactly clicking on a vertex and still select it. There is a tradeoff though. If the region is too large, then multiple vertices can be selected with a single click. Which one actually gets used in that case is (mostly) random.

The **Perspective Angle** sets the perspective features of the camera. Thus, a big perspective angle will approach the vanishing point and make the close objects look much bigger than distant objects. A small perspective angle will move away the vanishing point, coming closer to an effect of parallel projection. If the perspective angle is set to a value below 1, then the resulting image can be 2 dimensional at the pixel level.

The **Close and Far Cuts** define the positions of the clipping planes for the drawn area. You may need to decrease the close cut value and increase the far cut value if the scene does not appear entirely on the window.

The **Set background color...** calls the dialog in 9 (Figure).

In this window, you can choose the background color using the 0 to 255 rgb convention. The chosen color will be displayed in the bottom right box above the Cancel button. A translation of the color into 2 hexadecimal units is displayed under the "Selection" label. The OK button sets the background of the main window to the chosen color and closes the dialog. The Cancel button closes the dialog without modifying the background color of the main window. Note that whenever you load another scene in the DataViewer main window, the background color turns to black again.

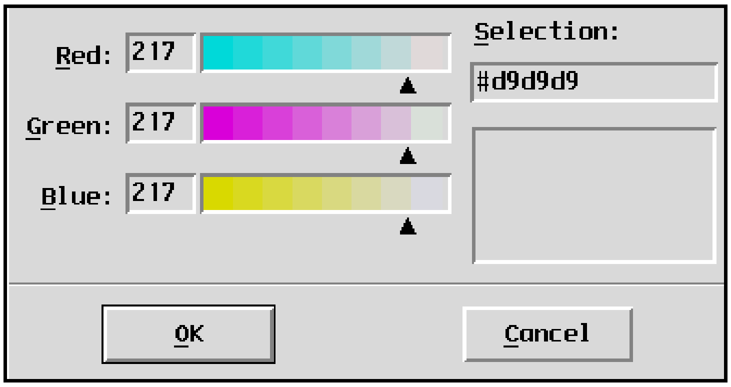

Figure 9:The Set Background Color window.

Figure 10:The Timer menu.

4.3 Timer

The Timer menu, shown in 10 (Figure), sets the Timer feature on and off.

If the Timer is on, then in the window where DataViewer is running from, you should see the number of frames per second that are being drawn. To use it, turn the timer on, then give a rotation motion to the scene (see 3 (Section)). The next time you click to stop the scene's rotation, you will get a message with the number of frames achieved during the whole operation. This feature can be used for benchmarking graphics cards.

4.4 Animate

The Animation menu, shown in 11 (Figure), has a single item.

Compiled animation

The "Animation Window..." operation calls the window in 12 (Figure), which handles the animation defined by the application in c++ through selection containers.

This window allows you to control the animation of the scene, in case the scene constructed by the application is designed with animation features.

For programmers: the animation is done in DataViewer using the classes `DVselection_container`,

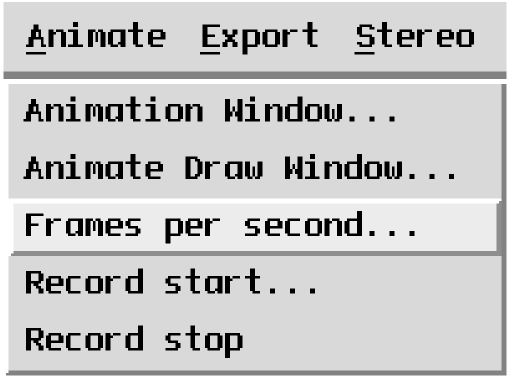

Figure 11:The Animate menu.

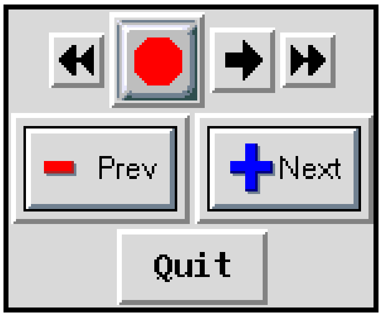

Figure 12:The Animation window.

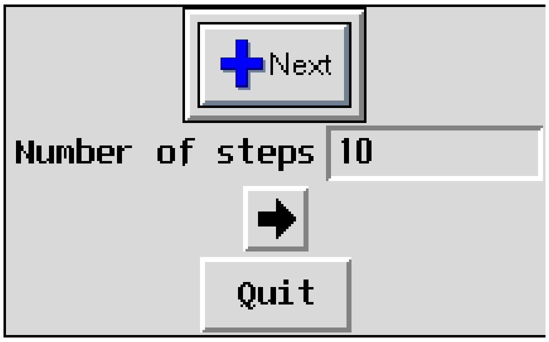

Figure 13:The Animate Draw window.

`DVrotation_interpolation_container` and `DVtranslation_interpolation_container` . The last two are in an experimental phase.

The animation is done by displaying a number of frames extracted or interpolated from the scene tree. The way the display is done is controlled by the buttons on this window.

On the first line, the first button moves the frame pointer to the first frame of the animation. The second button stops the animation that is running. The third button starts a frame by frame animation. The last button moves the animation button to the last animation frame.

The second line lets you move through the animation frames doing one step forward or backward at a time.

The last line contains a button that closes the animation window.

Scripted animation

The "Animate Draw Window..." operation calls the window in 13 (Figure) which handles the animation defined by the application in Python.

This dialog window concerns a special function, named "AnimateDraw" that can be defined by any DataViewer geometrical window (any application). By default, this function is equivalent to the draw function. The application can redefine this function to perform a specific transformation of the scene that can turn into an animation. For example, the bifurcation diagram window in VBM defines the AnimateDraw function as moving the current pointer one step forward in the diagram.

The first button, labeled "Next", makes one call of the AnimateDraw function.

The second line contains a typein specifying a number of steps for running the AnimateDraw function. It is used by the run button described in the next paragraph.

The third line contains a run button that makes a number of calls of the AnimateDraw function equal to the number of steps previously defined.

The last button, labeled "Quit", closes this dialog.

Animation tools

The "Frames per second..." operation calls the window in 14 (Figure). In this dialog, one can specify the speed of the animation for the compiled animation, and possibly soon for the scripted animation too. The default value for this parameter is 30.

The "Record start..." menu item opens a file dialog where you can specify a file name for taking snapshots

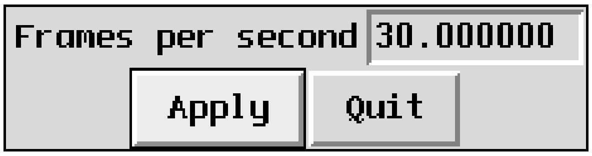

Figure 14:The Frames per second window.

Figure 15:The Export menu.

of the window. When the record is on, at each call of the Draw function, a new snapshot is taken from the window and saved as a ppm bitmap file under a name that is a composition of the specified name and a frame number. This can be used to record the compiled or scripted animation, as well as any move of the scene on the screen, namely rotation, translation or zoom.

The "Record stop" operation cancels the recording going on.

4.5 Export

DataViewer currently provides five exporting formats, available from the Export menu in 15 (Figure).

4.5.1 PPM

The first line in this menu exports the current scene in the ppm bitmap . This operation calls a file dialog where you must specify the name of a ppm file. Then a snapshot of the main DataViewer window will be saved as ppm in the specified file.

Note: as for most snapshot facilities, you must be sure that no other window stands in front of the DataViewer main window at export time, otherwise the resulting image may be disturbed.

4.5.2 POV-Ray

The second option in the Export menu saves the current scene in the required by POV-Ray, the *Persistence of Vision raytracer*. The output file will contain, besides the description of the geometrical objects composing the scene, together with their properties, the position of the camera such that the scene can be rendered in the almost exact position as it was in DataViewer. For a good view of the scene, the file will also contain a number of shadowless spotlights, and only one normal spotlight depending on the view point.

Radius note. When exporting DVlines, DVframed, DVaxes, and DVpoints_1D objects, simple OpenGL lines and points do not have a direct representation as POV-Ray objects, so they will be transformed into cylinders or spheres. In the case, the radius of the exported objects is computed on the basis of the size of the whole scene, so the result can be hazardous. We recommend to draw these objects in DataViewer as cylinders or spheres before exporting to POV-Ray, to have a better control over the radius.

Smoothing note. If the geometry complexity of DVlines, DVframed, and DVcones objects is superior to 8, at each point of the centerline will be placed a sphere with the same radius as the centerline. Thus, the rendered centerline will look smooth, otherwise gaps could appear between each two cylinders or cones. Most of the applications of DataViewer provide a control over the geometrical complexity of the whole scene, which is transmitted to each object unless the object defines its own geometrical complexity. Thus, in most cases, if you turn the geometrical complexity of the scene to 9, the centerline will be exported as cylinder/cones and spheres.

4.5.3 Description

This is a debugging especially meant for programmers of DataViewer. The output file will contain a description of the current scene with all the details of the objects it contains in an easy to read .

4.5.4 Scene file

This option export the current scene to DataViewer's native scene file . The output file can be viewed directly with the Scene file application (see 2.3 (Section)).

4.5.5 Raw data

This option simply exports the current scene as raw data. In the output file, each point is written on a line with simple spaces between its coordinates, and each group of points is separated by an empty line.

4.5.6 Buffer operations

The last two actions under this menu concern an extra geometrical buffer that can be useful for many applications.

Save to buffer

This action saves the current geometrical scene to a buffer that is also drawn by any Draw action. The content of the buffer is thus visible and can be rotated, translated and scaled with together with the current scene, but that cannot be used for interaction. This feature allows the user to compare two scenes produced by different calls of the Draw function inside the same DataViewer window. The contents of the buffer is also exported by any of the exporting methods together with the current scene.

Figure 16:The Stereo menu.

Each time the user saves the current scene to the buffer, it is added to the current content of the buffer. Thus, it is possible to accumulate multiple 3D snapshots of the geometrical scene in the same window. For example, in *VBM* one can simultaneously view the solutions corresponding to multiple points of the bifurcation diagram. In the *Scene File application*, one can merge any number of scene files by opening them and saving them to the buffer, and by exporting the entire geometrical scene as a scene file.

Clear buffer

This action deleted the contents of the extra geometrical buffer, that has been created by the previous action (Save to buffer).

4.6 Stereo

The Stereo menu contains 3 items, as shown in 16 (Figure).

This menu concerns the machines provided with a stereo device. According to the way it is installed, it is very likely that you have to set the screen to a stereo combination before you can turn the device on.

Local note. On lcvmsg2, you have to run the script `SGIStereoOn` before you can turn the stereo on inside DataViewer. It is recommended to turn the stereo screen combination off after quitting DataViewer, with the script `SGIStereoOff`.

The first two items of the menu turn the Stereo device on and off. If you turn the stereo on, then you will be able to see a 3D image with help of the glasses associated with the device.

The **Eye Separation...** item calls the dialog shown in 17 (Figure).

The stereo device is generating and sending each eye a separate image through the special glasses. In this dialog you can choose the distance between these images which is best for your eyes, with help of the Eye Separation typein. A bigger eye separation makes the image appear larger and more distant to you and increases the perspective effect. A too big eye separation can be difficult for the eyes to manage. The default value for this parameter is 0.005.

4.7 Magellan

The Magellan menu, shown in 18 (Figure), can turn on and off the 6D control device called Magellan.

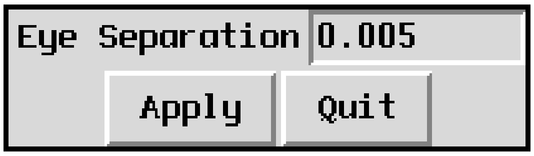

Figure 17:The Eye Separation dialog.

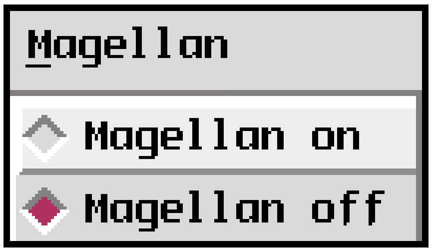

Figure 18:The Magellan menu.

According to its installation, you may have to turn the device on at a hardware level before you use it inside DataViewer. For the lcvmsg2, this operation is done with the `MagellanOn` script which opens a dialog window. By closing it, the Magellan is turned off at a hardware level.