Robotics

Practical 1: Introduction to CoppeliaSim

Andrew Davison
http://www.doc.ic.ac.uk/~ajd/Robotics/

1 Introduction

In the Robotics course this year, all practical work will be based on the CoppeliaSim robot simulation environment. CoppeliaSim (previously known as V-REP) is a cross-platform 3D robot simulator used across research, education and industry, and is available free for educational purposes. It enables us to realistically construct and test simulated mobile robots in all of the situations we are interested in for this course. We will build and understand algorithms using the powerful built-in programming and analysis tools.

In this first UNASSESSED practical of term, the goal is familiarisation with CoppeliaSim and what it enables.

We remind you that the coursework component of Robotics will be based on the assessment of several practical exercises. This week's work is not part of this assessment, but it is important to start to work with CoppeliaSim so that you are ready for the assessed exercises coming soon.

2 What is CoppeliaSim?

CoppeliaSim is a robotics simulation program which provides a sophisticated graphical user interface to a physically simulated 3D world. Within this simulation, scenes can be quickly assembled by dragging, dropping and adjusting primitive shapes as well as pre-defined objects and scenery to form a scene hierarchy. Pressing the play button runs real-time physics simulation, and the evolution of the scene can be viewed in 3D. Various types of sensors and motor-like joints can be combined with objects to form robot-like agents which can move through and interact with other static parts of the scene.

CoppeliaSim has a fully integrated programming interface (API) which means that a script can be attached to any object to control it within the simulation. Further, almost any aspect of the simulation can be customised and automated using scripts if needed.

These scripts can be written using a built-in editor, are saved as part of a scene file, and most commonly run in a 'callback' fashion, where functions in the scripts are called by the main simulation once per simulation timestep, and can therefore be used to implement robot sensing and control actions. CoppeliaSim's built in programing language is Lua, which may be unfamiliar to many but is easy to learn for anyone familiar with languagues such as Python.

CoppeliaSim has further built-in features which are extremely useful such as easy to use graph plotting of any type of quantity.

Using CoppeliaSim we will follow very much the 'build it up from scratch' and 'learn by doing' teaching philosophy we have followed in previous years of Robotics using hardware robot kits. The 'robots' we build and test will be made up from elemental parts, not unlike Lego, and the algorithms we work on will also be built from first principles.

Each week for the practicals we will be providing starting scene files alongside detailed practical instructions which will be a starting point for your work.

3 Installation

CoppeliaSim Edu (the powerful version which is free for educational purposes) is available for download from `https://www.coppeliarobotics.com/downloads` and should be a straightforward binary install for Windows, MacOS or Ubuntu Linux. Download the latest version (4.2.0 as of January 2022) and install it on your machine.

On MacOS, after downloading you will find the `coppeliaSim.app/` directory in your `Downloads` folder. Copy this whole directory into your `Applications` folder. Then you will be able to launch via the icon CoppeliaSim from your normal MacOs application menus. Note that running the executable may be blocked by MacOS the first time you try. If you open your `Applications` folder and right-click on `coppeliaSim` that will allow you to override the block, and then afterwards it should run from the normal application menu.

If CoppeliaSim doesn't run from your operating system's graphical interface for some reason, you will still be able to launch it from a terminal command line. e.g. on my Mac I can run the executable file at `/Applications/coppeliaSim.app/Contents/MacOS/coppeliaSim`. Other resources that come with the CoppeliaSim install such as built-in scenes and tutorials are also available here in the `/Applications/coppeliaSim.app/Contents/Resources` directory.

4 Getting Help

The Robotics course will have online interactive practical sessions every week, where students will work in groups within a Microsoft Teams session. I will be taking part in all of these sessions together with a large team of teaching assistants and you will be able to ask us questions directly there and interact via screenshare to solve problems.

Outside of the live sessions, your main place to ask for help is the EdStem channel for the Robotics course, where the TAs and I will endeavour to make rapid replies to any questions and to share general useful advice and tips.

We ask you not to use the general CoppeliaSim forums for questions about our course, and you shouldn't need to because we have a lot of expertise within our TA team.

4.1 General Tips

CoppeliaSim is stand-alone and efficient and should run well on most systems including laptops, though using a mouse will make GUI interaction easier, as will having a relatively large monitor because you will sometimes have many windows open.

When working in groups remotely, of course screensharing your desktop over MS Teams will make it easy to discuss and debug simulations.

CoppeliaSim scene files are cross platform and can easily be shared by any file sharing mechanism. As in any type of programming, make sure to **make backups of any important files**.

Since scene files are binary files, version control is not so easy, though it is certainly possible to cut and paste text in and out of CoppeliaSim scripts. There are also possible ways to get CoppeliaSim to load

scripts from external files, though this would cause other complications. We will share further tips on EdStem as groups find the the best ways to collaborate.

One final tip is to make sure you don't overwrite a scene file by one you may have changed by accident. This can be easy to do for instance if you open several scene files at once and tell CoppeliaSim to save everything as you close it while forgetting you changed something by accident. We have found that good practice when you shut down CoppeliaSim is to go through your open tabs and save and close the scenes separately before closing the application.

5 Useful Documentation and Tutorials

CoppeliaSim has a comprehensive online user manual available at:
`https://www.coppeliarobotics.com/helpFiles/index.html`. We recommend that you first have a go with our simple scene file and the explantion below here, but you can look at the manual any time you want more detail and use it as a reference later.

6 A First Scene File to Try Out

We have created a simple first scene file for you to try CoppaliaSim out. Download:
`https://www.doc.ic.ac.uk/~ajd/Robotics/RoboticsResources/practical1.ttt`
(this file is also available from the schedule part of the course website).

Copy it to a working directory and open it in CoppeliaSim using File→Open Scene from the menus at the top. Please read through the explanation here and play around with things in this scene to learn about CoppeliaSim.

In the main window you will see a 3D scene with various objects. To the left of the main 3D view is the Scene Hierarchy, where you'll see a list of all of the objects in the scene, and the way that these objects have been organised into a tree hierarchy describing relationships between them.

You can click either directly on objects in the 3D view or on the objects in the scene hierarchy to select them and you will see them highlighted in both places.

6.1 Objects and Physics Simulation

At the top of the CoppeliaSim, under the menus, there is a set of important icons which set the main modes of interacting with the scene. The three arrow icons on the left set modes for changing the view of the scene between sideways translation, rotation and in-out zoom-like translation. Try clicking on each of these and then dragging in the main 3D view to change the view.

When a scene is first loaded, the simulation is paused. In one part of the scene you will see a set of cuboid shapes hanging in the air. Look at the top right of the top row of icons and you will see play, pause and stop controls. Press play and you will start the simulation. The objects will fall into a pile under simulated physics. You can temporarily pause the simulation with the pause button, or pressing stop will reset the scene to its initial state.

Next to the play button you will see a box where you can set the value of 'dt'. This is the time-step size for the simulation. You can change it in this box when the simulation is stopped, or during the simulation you can change it using the hare and tortoise buttons to the right. Next to these icons there is button which looks like a clock which toggles the simulation into real-time mode. When this is selected

(the background of the button goes grey) the simulation will run in real-time mode, meaning that it will run no faster than real world time. When it is unselected the simulation can run faster than real-time, as long as the scene model is not too complicated. Most of the time we will keep this in real-time mode, and keep the default value of dt at 50ms.

6.2 Manipulating Objects with the GUI

Press stop to reset the simulation, then try changing the initial state of the cuboid shapes in the air. The 'object/item shift' and 'object/item rotate' buttons in the main top row of icons allow you to do this. If you click on the shift one for instance, you will see a pop-up window which lets you set 'mouse translation' mode where you can directly drag the object in the main 3D view (use the 'along x, y or z' buttons to select which axes directions to alter); or you can use the other modes such as position where you can enter object coordinates as numbers.

Note here that in the main 3D view you can always see in the bottom-right corner a mini x,y,z coordinate frame to see which way the world coordinate frame is oriented. Also note that the simulation uses metric units of metres for object positions, and degrees for rotation angles. It is handy to know that the squares in the 'chessboard' floor pattern are 0.5m squares.

Try moving some of the cuboids around, then pressing the play button to drop the objects from the new positions. If you press stop, the objects will return to the locations you moved them to.

Another thing to try is to add some new objects to the scene. Simple shapes can be found via Add→Primitive Shape in the main menu. The new object will appear at the origin of the coordinate frame (in the middle of the floor) by default; you can either set different coordinates when adding it or move it afterwards.

You can use cut, copy and paste with objects as you would in other applications, via the Edit menu or the usual keyboard shortcuts, as well as Undo and Redo.

The other way to add objects is via the 'Model browser', which is the bar in the GUI to the left of the Scene Hierarchy. There are a lot of pre-defined shapes, objects and even complete robots which can easily be added to a scene here by directly dragging them into the scene.

Note that you can close the model browser when you don't need it (deselect it in the Tools menu) to give more screen space to other things.

6.3 Saving and Loading Scene Files

If you have changed the starting positions of objects or added new objects to the scene, the new scene configuration can be saved to a new scene file using the File menu. If you just use 'Save scene', the previous scene file will be overwritten, so choose 'Save scene as' to save the altered scene file under a different filename.

CoppeliaSim can open multiple scenes at once, and each appears in a different tab. You can copy and paste between scenes and this is useful if you want to build and check some new structure in a separate new scene before pasting it into your main one.

6.4 Objects in Hierarchy; Joints and Sensors

In another part of the 3D scene, you will find a kind of tower where a cube is supported by a red rod above a cylindrical base. When the simulation starts, the cube rotates back and forth, but let us first consider the static structure.

If you click the cube or cylinder you will see them light up in the scene hierarchy, and you will see that unlike the falling cuboids which are all at the same level, the cube and cylinder have a tree-like dependency relationship with other objects. In fact there is a five-level hierarchy of parent-child relationships. 'Dummy' is the parent of 'Cylinder', which is parent of 'myMotor', which is parent of 'Cube', which is parent of 'Proximity_sensor'.

The object 'myMotor' is what CoppeliaSim calls a joint. When primitive objects and joints are connected in a hierarchy, they connect together physically. The cylinder and cube are therefore connected by the joint into a single structure.

Joints have a further role however, which is to serve as motors. The joint in our scene is a revolute joint which means that it can apply a rotational torque between the objects it connects. If you press play in the simulation, you will see the cube rotate with respect to the cylinder due to the motion of the joint. We will explain how this motion is controlled in the next section.

The element at the root of the tower hierarchy is a 'Dummy'. This a useful entity in CoppeliaSim which isn't a physical entity in the simulation, but simply a reference point with position and orientation. It often useful to use a dummy as a reference frame for a compound object, but it is also possible to form a compound object without one.

The element at the other end of the hierarchy is a proximity sensor which is attached to the cube. This sensor measures distance along the visualised ray. We have set up a graph to plot the measurement from this sensor as a function of time, and this graph pops up in a new window when the simulation is started. Graphs can be easily configured to report many kinds of things; double click on the graph's icon in the scene hierarchy to change its settings.

Hierarchies can be created in CoppeliaSim by simply dragging objects onto one another within the Scene hierarchy view. The name of any object can be changed by double clicking on it.

7 Programming in CoppeliaSim

The final element of this introduction to CoppeliaSim is a first look how programming works.

Something you will soon learn about CoppeliaSim is that everything that can be done with the GUI and more can also be controlled with code in the form of Lua scripts. It is the interplay between GUI scene interaction and integrated programming which makes CoppeliaSim so powerful.

Any object in the scene hierarchy can be given a script, which is a program to control it. Scripts can be seen in the scene hierarchy as icons that look like pieces of paper to the right of the name of an object. There is one script at the top next the root of the hierarchy. This is the main simulation script, which is usually just a default way to run all the parts of the simulation in order and should not be changed in normal circumstances.

We have added our own script to the Dummy object which is the root of our tower. This is called a 'child script' which controls one small part of the simulation and we will be using these scripts to control robots. Usually we will attach a script to the root object of the sub-hierarchy which makes up our robot, so we use the Dummy for the tower in this case. (We will use 'non threaded' child scripts, and they can be added to objects via the Add menu.)

Double click on the Dummy object's child script's icon to open it in the built-in editor and see its contents. When a script is edited here the effects of the script will change the next time the simulation is started. When a scene is saved, all associated scripts are saved as part of the scene file.

7.1 Callback Programming using Child Scripts

Unthreaded child scripts work via callbacks, which is a common style of programming especially in real-time and event-driven systems. Any child script in CoppeliaSim has a set of pre-defined functions which will be called by the simulation as it runs. If these functions are left empty the script won't have any effect.

There are only two of these pre-defined functions which are important to us at the moment.
`sysCall_init()` is called for each object at the start of the simulation, and can be used to set things up.
`sysCall_actuation()` is then called once every simulation step, and is where we will put code which implements a robot's actions.

The script we have defined performs a simple control loop which sets the direction of the motor's rotation depending on the value of the proximity sensor. When the distance the sensor measures to the diagonal wall gets too large or too small, we change the value of the variable `myLeftRightFlag` which is then tested to set the rotational velocity of the motor to be either positive or negative.

You should be able to read through the script and understand how it works, and start to get used to the syntax of Lua. The way that comments, function definitions, 'if' statements and so on work in Lua is a little different from Python or other languages but it should be easy to get used to these differences.

You will see that in `sysCall_init()` we set up references to objects via their names in the scene hierarchy so that we can read and set their properties from the script.

We have added some print statements to give an easy insight into what the program is doing, and the output of these appears in the window at the bottom of the GUI when the script runs. These print statements are an easy way to debug your programs. Pause the simulation to have a look at the output in detail. Graphs are another great way to understand how a program is working, and it is possible to make graphs of arbitrary data generate from scripts as we will see in next week's practical.

If you make a change which causes an error, the script will fail. In this case, you will see a little red 'X' appear on the script icon, and red text will appear in the output window to show you where the error is.

8 Exploring Further

You should be in a good position now to be ready for the practicals coming in subsequent weeks.

If you want to explore CoppeliaSim more at this stage, we recommend the user manual at `https://www.coppeliarobotics.com/helpFiles/index.html`. In particular the sections on user interface, writing code in and around CoppeliaSim, and the tutorials are very good.

Full documentation on the programming API is available at:
`https://www.coppeliarobotics.com/helpFiles/en/apiFunctionListCategory.htm` The Lua versions are all on the left. Note that there are various other ways for programs to interact with CoppeliaSim using a range of programming languages (including our own PyRep by Stephen James), but these require installing additional software and may not run on all platforms so we will be supporting only Lua during the course.

There is a lot in the manual and API which may seem daunting at first, especially because almost everything in CoppeliaSim is highly configurable. However, you should quickly get used to it and the best way in will be via the work we set in the practicals.

The tutorials section of the manual is very useful, especially bubbleRob. The scene files for the tutorials are installed as part of the standard setup so you can run them by going to File→Open Scene in

the menus and opening the `Contents/Resources/scenes/tutorials/` directory within your CoppeliaSim install. There are plenty more example scenes to look at in `Contents/Resources/scenes/` with all kinds of interesting pre-built robots, sensors and scenes.

If you are new to programming in Lua, there is a good introduction at: `https://www.lua.org/pil/contents.html`, though again the best way to learn in general will by trying to understand and modify sample programs.