FOREWORD

The practical works are graded based on your involvement, how well you organize your work and report (it should be easy to understand your thinking and easy to find your responses), and how well you follow the submission instructions below. Moreover, by doing only the "basic" realization of the the practical work will lead basically to a "*fair*" mark. You can increase your mark by going further, try to solve some "optional" issues, and even provide some discussions on how to do so (in case you don't have time to handle it).

Honor code

It is our duty to provide a fair evaluation of your performance. You may to discuss the the practice material with the instructor and your classmates. But you are not allowed to share your answers or code with others. Anyone requesting for answers or code, or providing answers or code, or becoming aware of others doing so without reporting to the instructor, is considered in violation of the honor code.
You are allowed to form a small team (up to 3 members), and only one report is requested. But, each member must then join a personal discussions/conclusion about the practical work and their involvements.

The report

A written report should be of a professional/scientific standard – that is, well laid out, and neatly presented. Report writing is an essential skill for students but also and especially for engineers. Engineering reports analyze data, present results and conclusions, and make recommendations in a logical, precise and accessible manner.
In your report, you may (briefly) recall the basic information related to your case study that come from the teaching materials or elsewhere. If you have used other resources (e.g. from the Internet or literature), you **must** reference them in your report.
Do not hesitate to illustrate your report with figures, plots… to validate the different objectives. You can also join some movies (don't forget to reference them and comment them in your report), but take care to the size of the movies!! (prefer a compressed standard format, e.g. `mp4`…)

Instructions for uploading your practical work

Once you have finished your practical work, upload it on Teams, team code: `czd3zo5` (avoid to send it by email).

> **!** Contact me if you are having difficulty joining the 'Advanced Robotic' team or if you are having difficulty accessing the assignment.

1. Upload on time! Late submissions may not be accepted without proper justification;
2. Only upload one `zip` file (or other standard archive format: `rar`, `7z`…);
3. In your `zip` file, you **must** include all source codes and materials required to execute or validate your simulation(s);
4. Always include your report as a **pdf file** (no `docx`, `odt`…);
5. Please name the uploaded file in the following format: `advRob_your-names.zip`[1].
   Please pay attention to the size of the uploaded files.

[1] In case of a team, you may mix the members names to **uniquely** identify you.

1. Simulation with CoppeliaSim

1.1 Introduction

The robot simulator CoppeliaSim, with integrated development environment, is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or BlueZero node, a **remote API client**, or a custom solution. This makes CoppeliaSim very versatile and ideal for multi-robot applications. Controllers can be written in C/C++, Python, Java, Lua, or used with **Matlab®** or **Octave**. CoppeliaSim is used for various applications: fast algorithm development, factory automation simulations, fast prototyping and verification, robotics related education, remote monitoring, safety double-checking, as digital twin, and much more.

1.2 First steps with CoppeliaSim

> **!** If CoppeliaSim cannot been found (e.g. not installed), try to download **CoppeliaSim Edu** (zip package without installer) from: https://www.coppeliarobotics.com/downloads, and unpack it in your local folder (see also Section 3.1).

Once you have launched CoppeliaSim, the user interface will display several elements. Its main elements are shown in Fig. 1.1:

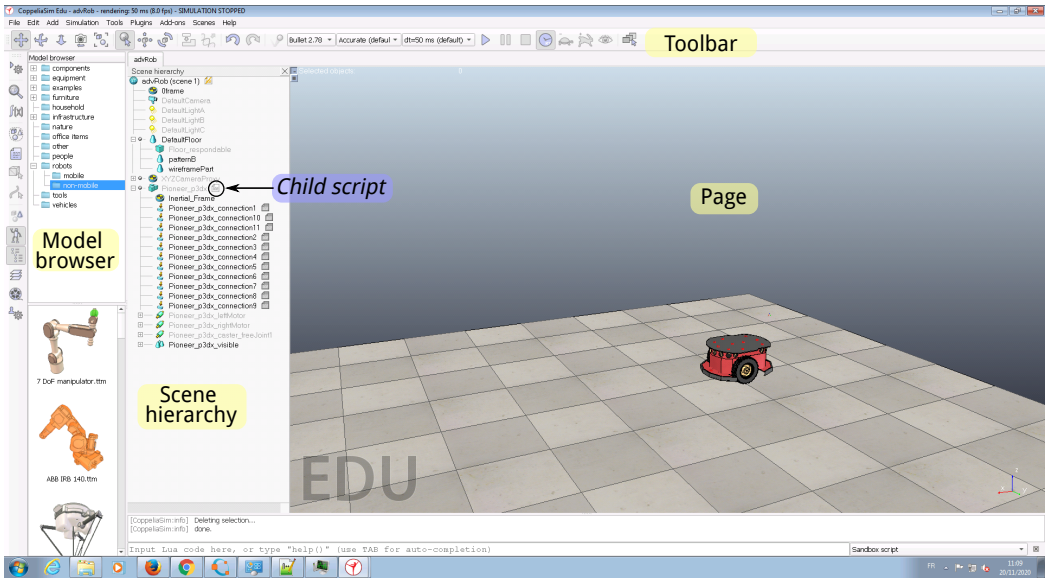


Fig. 1.1: CoppeliaSim user interface main elements.

1. The model browser: the model browser supports drag-and drop of CoppeliaSim models into the scene. When you drop a model into the scene hierarchy, the model appears in a default position (usually at the center of the scene). If you drop a model into the scene view, it will be positioned at the drop location.
2. The scene hierarchy: the scene hierarchy displays all objects in the scene, visualizing their parent-child relationships. Each object also reveals its type via its display icon. The icons can be double-clicked to open their respective object property dialogs. On the right-hand side of some object icons you will find another icon representing a script: those icons can also be double-clicked in order to open the attached child script.

3. The scene view: by default, the current page displays the 3D content of a scene. You can click objects/models to select them, and you can manipulate them via the various toolbar buttons, the menu bar, or various key combinations (e.g. delete, copy/paste, etc.)
4. The toolbar buttons: the toolbar presents functions that are often accessed (e.g. changing the navigation mode, selecting another page, etc.), they allow changing the main settings of a simulation (e.g. the physics engine that will be used), and they allow starting/pausing and stopping a simulation.

Try to *familiarize yourself* with the simulator, for instance by drag-and-dropping a few robot models into the scene, then hitting the start button. Try also to run a few of the demo scenes (`menu bar> File> Open scene...`).

> **!** If CoppeliaSim cannot been launched or you receive a message like:
>
> "Windows cannot access the specified device, path, or file"
>
> Try to download **CoppeliaSim Edu** from: https://www.coppeliarobotics.com/downloads, and unpack it your local folder. If the issue persists, unpack it on a USB key.

1.3 Connection to a client application

CoppeliaSim can be programmed/controlled via many different means. One of them are the child script previously mentioned. For instance, you may edit a robot child script (as illustrated in Fig. 1.1), and have a look to `function sysCall_actuation()` and try to change the programming.

> ⚠ If you load the scene `advRob.ttt`, and want to try to modify the `Pioneer_p3dx` child script, be aware that you must save a backup of the original scene. Otherwise, you risk compromising some functionality of the practical work.

CoppeliaSim offers also several ways to control a simulation from an external application (i.e. a client application). One of them is the remote API. The main features of the remote API are: cross-platform; lightweight (i.e. also embeddable onto a microcontroller); direct support for several languages (C/C++, Java, Python, **Matlab®**, *Octave*...); support for queries (i.e. blocking function calls) or streaming (i.e. non-blocking function calls).
Here, **Matlab®** is used as the client application via the remote API to control the simulation or a *robot*. Therefore, you will have to '*program*' a simulated robot using a Matlab script.

1.4 Running the practical work

In order to test that everything is setup properly, please do the following:

1. Download the practical work material `advRob_TP.zip` from Teams assignment part.
2. Unpack `advRob_TP.zip` to some local folder, and rename it as something like[1]: `advRob_your-name/`.
3. Launch CoppeliaSim and load the scene file `advRob_your-name/scene/advRob.ttt`.
4. Start **Matlab®**, navigate to `advRob_your-name/advRob/` and execute the file `tests.m`.
5. CoppeliaSim should start the simulation and perform various tasks.

If you do not receive any errors and CoppeliaSim performs the simulation it means that your installation is complete and working (there is no need to pay attention to warnings if there are any).

The code that configures a remote API client, required to perform **Matlab®** – CoppeliaSim communication, is located in the `advRob_your-name/advRob/` folder. There are different kinds of functions:

- ☐ `coppelia_XXXX.m`: allowing to communicate with the CoppeliaSim simulator (eg. setup, open connection, start simulation...); These functions **should not be modified!**.

[1] When the practical work is due, you will need to pack (i.e. zip) the `advRob_your-name/`, and upload it on Teams. It will be then convenient for the name you give here to **uniquely** identify yourself or your team.

- `Pioneer_p3dx_XXXX.m`: allowing to communicate with the robot (Pioneer p3dx) in the CoppeliaSim simulator. These functions **should not be modified!**

> ! To use theses function, the scene file `advRob_your-name/scene/advRob.ttt` must first be loaded in CoppeliaSim simulator.

- `tests.m`: shows some example of basic functionalities of the remote client. Useful to check if your installation is working properly.

Your work should be placed in the `advRob_your-name/work/` folder which already contains the following files:

- `advRob_XXXX.m`: to be implemented/adapted/modified for the practical work. Most of these functions are not implemented and therefore will not work until you have implemented them. Others, provide some basic implementation that can be changed to fit your needs.
- `startup.m`: a script that fix the paths to get access to the previous `advRob_your-name/advRob/` files. If in doubt, you can run this script every time you (re)start Matlab.

> i The lines which contain `gen_error('advRob:unimplemented',...)` must be commented or removed once the function are implemented.

Above `advRob_XXXX.m` functions, and several others in the `advRob_your-name/advRob/` folder, can be called irrespective of whether you are running Matlab® (internally they will branch to different code appropriately). If needed, you can add additional functions.

1.5 Troubleshooting

If you encounter any issues either with CoppeliaSim or the remote client (Matlab or Octave), **please try first** to read their respective documentations:

- CoppeliaSim help: http://www.coppeliarobotics.com/helpFiles/
- Matlab help center: https://www.mathworks.com/help/matlab/
- Octave documentation: https://octave.org/doc/latest/

> i You can continue this practice on your own at home. To do this, you can refer to the Chapter 3. In this appendix you will also find further information in case of issues.

2. Autonomous Robot Navigation

2.1 Introduction

The overall goal of this practical work is to develop an **autonomous robot** based on the *Pioneer P3-DX*. The considered environment is an indoor scene as the one illustrated in Fig. 2.1 (cf. scene `advRob.ttt` in CoppeliaSim).

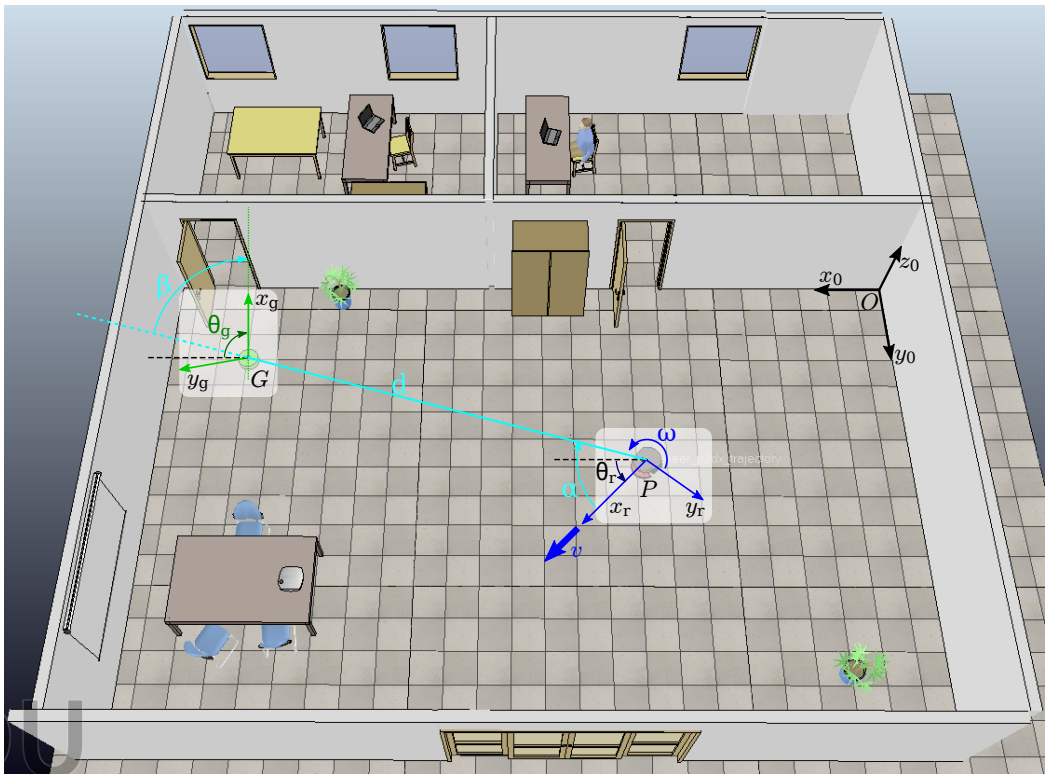


Fig. 2.1: The indoor scene for autonomous robot navigation.

⚠ Be aware that the reference frame $\mathcal{F}_0$ shown in Fig. 2.1 is not necessarily the world frame used by default in CoppeliaSim.

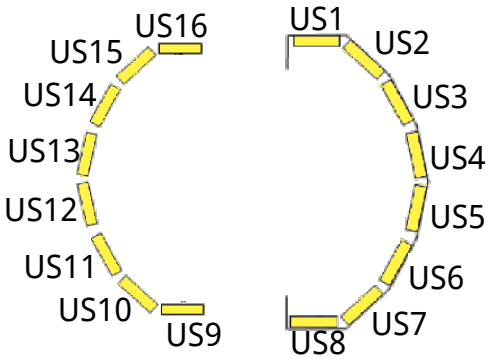
ℹ You may change the `advRob.ttt` scene by moving, removing, adding… some objects. But keep in mind that the environment must reflect an indoor-like one.

2.1.1 Pioneer P3-DX

The robotic platform consists of the Pioneer P3-DX from the company MobileRobots (see Fig. 2.2). It consists of an aluminum body (about 44x38x22 cm) with *two independent driving wheels*. The two DC motors use 38.3:1 gear ratios and contain 500-tick rotary encoders. The mobile base can rotate in place moving both wheels, or it can swing around a stationery wheel in a circle. A non-actuated rear castor wheel is included for balancing the robot. On flat floor, the Pioneer P3-DX can move at speeds of 1.2 m/s. At slower speeds it can carry payloads up to 23 kg. In addition to motor encoders, the P3DX

base includes 16 ultrasonic transducer (range-finding sonar) sensors arranged to provide 180° forward and backward coverage (see Fig. 2.2b).

(a)



(b)

Fig. 2.2: The robotic platform consisting of the Pioneer P3-DX. US1..US16 are the 16 sonars.

> The considered Pioneer P3-DX have a very few sensors. You may add some additional sensors (in CoppeliaSim) to extend the robot capability. You have then to refer to the CoppeliaSim documentation to find out how to do it. In such case, the reason and the choice of the new sensor must be discussed in the report.

> Some of the robot parameters can be accessed after the following instruction:
>
> ```
> robot = Pioneer_p3dx_init(connection);
> ```
>
> then `robot` is a structure with, among other, known parameters.

2.1.2 Instructions

To develop *autonomous robot navigation* different aspects have to be investigated in this work.

1. Given desired linear (v) and angular (ω) velocities of the robot, one should compute the corresponding wheel velocities in order to make the robot drive accordingly (ie. kinematic modeling).
2. Develop a controller that computes velocity input commands (v, ω) to drive the robot to a specified target posture $q_g = (x_g, y_g, \theta_g)$.
3. Detect and avoid encountered obstacles.
4. (optional) Localize where is the robot with respect a specified reference frame $\mathcal{F}_0 : (O, x_0, y_0)$;
5. (optional) Realize the environment mapping.

In order to achieve this practical work, do the following:

- ☐ Unpack `advRob_TP.zip` to some local folder, and rename it as something like: `advRob_your-name/`.
- ☐ Launch CoppeliaSim and load the scene file `advRob_your-name/scene/advRob.ttt`.
- ☐ Start Matlab®, navigate to `advRob_your-name/advRob/` and execute the script `tests.m`, and check that your installation is complete and working.
- ☐ In Matlab®, navigate to `advRob_your-name/work/` and run the script[1] `startup.m` to fix the paths.
- ☐ Edit `work/advRob_simulation.m` and develop the practical/ work…
  Please put your various m-files, scripts and functions that you have implemented and need in the `advRob_your-name/work/` folder.

[1] If you launch Matlab® directly in the `advRob_your-name/work/` directory, the `startup.m` script should be executed automatically.

> The proposed `work/advRob_simulation.m` is a basic program skeleton. You can modify and adapt it to fit your own development of autonomous robot. For instance, you can change the for-loop `MaxStep` value or define a while-loop. On the other hand, the value of parameters with the `% to be defined` comment must be specified and (briefly) discussed in the report.

> The following sections provide some guidelines to address the above **3 mandatory aspects** that should be addressed to get a globally "fair" mark. Trying improving/enhancing some parts will increase the mark, that is to go further from what is presented. Especially, the points 4 and 5 related to the localization and mapping are optional.
> Anyway, keep in mind that the main objective is to develop an *autonomous robot*, with the **Pioneer P3-DX** as robot. You can propose alternative solutions to achieve this objective by specifying the reasons for your choices.

2.2 Motion Control

First, the kinematic models of the Pioneer P3-DX have to be addressed, by expressing the relationship between the input $\mathbf{u} = (\dot{\varphi}_L, \dot{\varphi}_R)^t$ of the left ($\dot{\varphi}_L$) and right ($\dot{\varphi}_R$) wheels and the robot velocities:

$$\mathbf{u} = \mathbf{H}(q)\begin{pmatrix} v \\ \omega \end{pmatrix} \tag{2.1}$$

where $\mathbf{H}$ is a matrix, and $(v, \omega)^t$ is the robot linear and angular velocities (see also Fig. 2.1).

> Give some comments on the *kinematic properties* of the Pioneer P3-DX.

Once the kinematic model is obtained, edit `work/advRob_calculateWheelSpeeds.m` in such a manner that it computes the wheels spinning $(\dot{\varphi}_L, \dot{\varphi}_R)$ based on the given velocities v and ω.

In order to obtain a precise and smooth control of the Pioneer P3-DX onto a specified goal posture $q_g = (x_g, y_g, \theta_g)^t$ (see also Fig. 2.1), a feedback control law have to be implemented. For instance, a basic state feedback control law can be defined as follows:

$$\begin{cases} v &= k_v d \\ \omega &= k_\alpha \alpha + k_\beta \beta \end{cases}, \text{ with } k_v > 0, k_\beta < 0 \text{ and } k_\alpha + \frac{5}{3}k_\beta - \frac{2}{\pi}k_v > 0 \tag{2.2}$$

where k_v, k_β and k_α are constant gains to be defined to ensure good stability of the controller. d is the distance between the robot current posture $q = (x_r, y_r, \theta_r)^t$ to the goal q_g, and the angles α and β are as shown in Fig. 2.1.

> You may use or edit the function `work/advRob_getGoal.m` to define either the goal manually, or to the use "*TargetGhost*" in CoppeliaSim; and then test your control strategy in different situation.

Implement a close-loop position controller by editing `work/advRob_calculateControlOutput.m`. Edit `work/advRob_simulation.m` in such a manner that you can simulate the autonomous navigation of the Pioneer P3-DX.

> You can save and change the script name to reflect different relevant scenarios (eg. `advRob_simulation_case_A.m`, `advRob_simulation_case_B.m`...). For instance, try to investigate the change of the gains k_v, k_β and/or k_α value. Especially, the case when the above constraints Eq. 2.2 are not satisfied.

You may propose some more *advanced motion control scheme*, for instance to take into account the maximum wheel velocity ($|v_{\max}|$), to follow a velocity profile, to improve the quality of the convergence to the goal, and so on.

2.3 Collision Avoidance

Local obstacle avoidance focuses on changing the robot's trajectory as informed by its sensors during robot motion. The resulting robot motion is both a function of the robot's current sensor readings and its relative location to the goal position. In the literature there are several ways to address this problem. Most of these algorithms are based on proximity sensors of the robot, which has a limited vision margin around itself.

Here, we suggest to use the Braitenberg algorithm. This algorithm is based on the Braitenberg vehicles; where the robot's sensors are tied directly to the motors controls. The Braitenberg algorithm proposes to create a weighted matrix $\mathbf{W}$ that converts the sensor outputs $\mathbf{s} = (s_1, s_2, \dots, s_n)^t$ directly into motors speeds, that is for instance:

$$\begin{pmatrix} \dot{\varphi}_L \\ \dot{\varphi}_R \end{pmatrix} = \mathbf{W} \left(1 - \frac{1}{s_{\max}} \mathbf{s} \right) \tag{2.3}$$

where $s_{\max}$ represents the maximum value of the sensor (eg. $n = 16$ if all Pioneer sonars are considered). If none obstacle is detected ($s = s_{\max}$), the velocities are the same as computed in Section 2.2. As the distance to the obstacle is decreasing, the velocities are slowed down to avoid collision.

Implement a collision avoidance strategies by editing `work/advRob_obstacleAvoidance.m`. Edit `work/advRob_simulation.m` to integrate your obstacle avoidance algorithm.

The above obstacle avoidance based on Braitenberg algorithm is just a suggestion. You can propose an other strategy to avoid collision. In such case your choice must be explained and discussed.

Optional: try to get the robot through an open door, without colliding …

2.4 Global Localization

For a lot of application in robotics, knowledge of the position and orientation of the platform is essential for autonomous navigation. To navigate from one place to another, the vehicle would need to know its position in the indoor scene as well as its heading. On its way, it might come across walls, doorways, and furniture, all of which would be perceived as measurements located along lines by its sensors.

Propose function(s) allowing to extract some lines from the environment. For instance, you can define a function to compute the line regression (ie. line fitting) using a set of points in Cartesian or Polar coordinates.

At least, **two orthogonal lines** have to be extracted from the robot scene. These lines can be used to define a reference frame $\mathcal{F}_0 : (O, x_0, y_0)$, like as shown in Fig. 2.1. Then discuss how the the robot can use these lines to localize itself.

To simplify the global localization process, you can first remove some furniture in the `advRob.ttt` scene. Don't forget to keep a backup of the original file.
To get a set with sufficient input points from a range scan, the robot may perform some specific movements/maneuvers.

3. Appendix: Use CoppeliaSim at home

CoppeliaSim[1], formerly known as V-REP, is a robot simulator used for fast algorithm development, factory automation simulations, fast prototyping and verification, robotics and so on... Educational entities (hobbyists, students, teachers, professors, schools...) can use CoppeliaSim Edu for free. Therefore, you can install and use CoppeliaSim from your home.

3.1 Installation

3.1.1 Pre-requires

As stated, CoppeliaSim can communicate with client application such as **Matlab®** or **Octave**. You should have either one of them installed. However, **Matlab®** require a license to be used. If you can't have access to a **Matlab®** license, then you have to consider the installation of GNU Octave.

> **!** CoppeliaSim can also work with controllers written in C/C++, Python, Java, Lua, etc. You are free to try to use these bindings, but in such cases **do not ask me how to do**!
> I only offer my support for Matlab and Octave clients application.

3.1.2 GNU Octave

GNU Octave is *free software* featuring a high-level scientific programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with *MATLAB®*. Please visit the GNU Octave site and follows the instructions to download[2] and install it. You can have access to the documentation here: https://octave.org/doc/latest/. To get some help about a function you can in the workspace prompt set: `>> help a_function`.

3.1.3 CoppeliaSim Edu

- ☐ Windows: download CoppeliaSim Edu and execute the installer or extract the `zip`-package. You should be able to start CoppeliaSim via the application menu, or by double-clicking on a CoppeliaSim scene.
- ☐ Mac OSX[3] (10.6 - 10.11.6): download CoppeliaSim Edu and unzip it. You should be able to start CoppeliaSim by executing `vrep.app` in that folder. You can also start it via the terminal by typing `./vrep.app/Contents/MacOS/vrep`.
- ☐ Linux: download CoppeliaSim Edu for your architecture (32 or 64 bits), and unpack the archive. You might probably need to run `bash chmod+x coppeliaSim.sh` for appropriate execution permissions. Then you should be able to start CoppeliaSim by typing `./coppeliaSim.sh` in that folder.

[1] Download CoppeliaSim by following this link: https://www.coppeliarobotics.com/downloads
[2] Download GNU Octave by following this link: https://www.gnu.org/software/octave/download.html
[3] As I don't have access to a MacOs, this has not been tested.

3.2 Troubleshooting

3.2.1 CoppeliaSim

If you are running CoppeliaSim on one of the supported platforms (see further up), then you might still experience problems, mainly related to your graphic card or graphic card driver. Make sure to always use the official drivers, and that they are up-to-date. If the openGl rendering of the scene is slow, faulty, or if the vision sensors do not operate correctly (e.g. the sensor that is used to display the map of the terrain in the floating view of the `advRob.ttt` scene), then try selecting a different item in the openGL settings dialog. Refer to CoppeliaSim help, and read this page: https://www.coppeliarobotics.com/helpFiles/en/settings.htm if needed.

By default, the client will connect to the local port `19997`. On the CoppeliaSim side, a server service is normally automatically started on that port at CoppeliaSim start-up. That behavior and some other settings can be changed in CoppeliaSim's folder, by modifying the file `remoteApiConnections.txt`.

Do not hesitate to browse the CoppeliaSim forum: https://forum.coppeliarobotics.com/, and especially the *'Bug reports'* sections.

3.2.1.1 Troubleshooting on Windows

If CoppeliaSim crash when clicking on the model browser's window:

1. you have administrator grant: enable write access to the whole CoppeliaSim folder;
2. you don't have administrator grant: copy-paste the whole CoppeliaSim folder to a different location that has full write access (e.g. an USB key), and run it from there.

3.2.1.2 Troubleshooting on MacOS

If CoppeliaSim does not launch, you may need to execute the following command in a terminal from within CoppeliaSim's main directory:

```
sudo xattr -r -d com.apple.quarantine *
```

In case, you have experienced periodic crashing (Segmentation fault: 11) on MacOS, try to run CoppeliaSim from a terminal (not clicking CoppeliaSim icon from GUI). This happens on Mac Sierra 10.12.3 (16D32). Symptoms were periodic crashing (every 2 5 min) and nothing appears in the model browser.

3.2.2 CoppeliaSim – Octave connection

In case you run into problems or are having difficulties establishing a connection to CoppeliaSim, remember that the Octave remote API client requires following items in its path, or current folder: `remApiSetup.m`; and the remote API client library for Octave (`remApi.oct`). Make sure to select the correct library, depending on your platform and architecture. Above files can be found in the `advRob/octave/` and `lib/octave/<your-platform>/`, or in CoppeliaSim's `programming/remoteApiBindings/octave/octave` and `programming/remoteApiBindings/octave/lib` folders. To check if these files are your path, and where Octave look for them, try the following in the Octave workspace prompt:

```
>> [connection] = coppelia_setup();
>> which remApiSetup
'remApiSetup' is a function from the file <some_where_on_your_computer>/
advRob_your-name/advRob/octave/remApiSetup.m
>> which remApi.oct
'remApi.oct' is the file <some_where_on_your_computer>/advRob_your-name/advRob/
../lib/octave/linux/remApi.oct
>>
```

Refer to CoppeliaSim help, and read this page:
https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm if needed.

If you get an error message similar to:

```
error: coppelia_setup: library open failed: <some_where_on_your_computer>/
advRob_your-name/advRob/../lib/octave/win/remApi.oct
```

Then, go to directory `advRob_your-name/advRob/lib/octave/src/`, and run the script `buildWin`, `buildLin` or `buildMac` according to your platform, and then copy the `remApi.oct` to `advRob_your-name/advRob/l win or mac}/`. If it fails to build the `remApi.oct` tell me on Teams (but don't care about the warnings).

3.2.3 CoppeliaSim – Matlab connection

In case you run into problems or are having difficulties establishing a connection to CoppeliaSim, remember that the Matlab remote API client requires following items in its path, or current MATLAB folder: `remApi.m`; `remoteApiProto.m`; and the remote API client library for Matlab® (eg. `remoteApi.dll`, `remoteApi.so` or `remoteApi.dylib`). Make sure to select the correct library, depending on your platform and architecture. Above files can be found in the `advRob/octave/` and `lib/matlab/<your-platform>/`, or in CoppeliaSim's `programming/remoteApiBindings/matlab/` and `programming/remoteApiBindings/lib` folders. The same test as above with Octave with the use of the `which` function can be used in the Matlab workspace.

Refer to CoppeliaSim help, and read this page:
https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm if needed.