EPITECH
LEARN DIFFERENT*

NEED4STEK

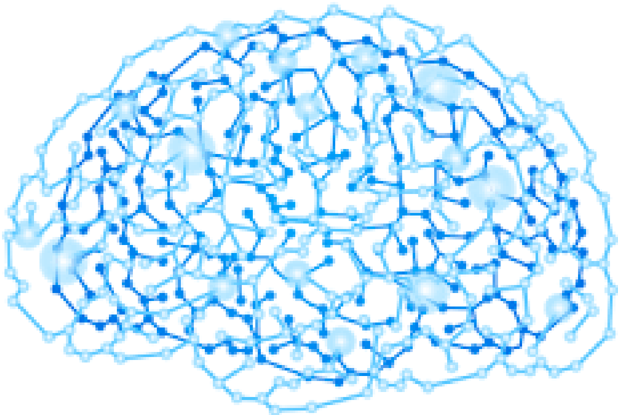
AUTONOMOUS CAR CONCEPT

NEED4STEK

Preliminaries

binary name: ai
language: C
compilation: via Makefile, including re, clean and fclean rules
Authorized functions: all of the libC and libmath functions are authorized.

- ✓ The totality of your source files, except all useless files (binary, temp files, objfiles,...), must be included in your delivery.
- ✓ All the bonus files (including a potential specific Makefile) should be in a directory named bonus.
- ✓ Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).



Project

By now you've understood that it's now your turn to create your own autonomous car. It must be capable of driving on a track without hitting the walls or driving in the wrong direction.

In order to help you with this task, several tools are avaible:

- ✓ CoppeliaSim is a simulator created by Coppelia Robotics. It enables the creation and control of robots. For this project, we have built a virtual track and car that you will use to implement for AI. You will find, further on in this document, how to install/use this software on your machine,
- ✓ an API (binary name: n4s). It is a communication interface that offers an array of actions that you can use (start a simulation, set the car motor's speed, wheel angle,...). You can also use it alone in order to test the communication protocol on the command line. This communication protocol is defined further on,
- ✓ a shell script (binary name: pipes.sh) that enables you to connect your program to the API and launch simulations. It's a binary that you will run in order to test your AI. Its utilization is also presented further on.

CoppeliaSim

Installation

Download CoppeliaSim Edu V4.1.0 (the assets provided for the project function solely with Linux). In order to launch CoppeliaSim, you need to execute the onboard script:

```
Terminal
~/B-AIA-200> ./coppeliaSim.sh
```

Scene

Download and then unzip the latest version of the `B-CPE-201_Need4stek_package.tgz` from the intranet (file provided with the project description).

Miscellaneous ".ttt" files are located in the "scene" file. These are the files that CoppeliaSim uses in order to save the scene simulations.
As far as we're concerned, they contain the track and the car to be driven.
Before beginning your simulations, you must systematically open the `track_1.ttt` scene (or every

other scene that you have made yourself) that is located in the CoppeliaSim (drag 'n drop or via the "File > Open scene..." menu).

Make sure, when starting the CoppeliaSim, that the selected physics engine is the ODE. The car will perform best in this mode. Menu: Simulation -> Using the ODE physics engine.

API

Overral Functionning

The **n4s** binary enables communication with CoppeliaSim (via a socket in C) and controls each of the elements that we have put in our scene.
It reads the commands sent to it on the standard input, carries out the task, and responds by writing on the standard output.
Your binary should, therefore, do the opposite: give off its commands by writing on the standard output and receive the **n4s** responses by reading them on your standard input.

The command in the *pipes.sh* script enables you to correctly link the different binaries. You'll need to execute the *pipes.sh* script in order to launch the simulation and see how it interacts with your AI.

Commands

The communication is done via a text-type protocol:

command	value range	answer type
START_SIMULATION	-	(1)
STOP_SIMULATION	-	(1)
CAR_FORWARD:float	[0;1]	(1)
CAR_BACKWARDS:float	[0;1]	(1)
WHEELS_DIR:float	[-1;1]	(1)
GET_INFO_LIDAR	-	(2)
GET_CURRENT_SPEED	-	(3)
GET_CURRENT_WHEELS	-	(3)
CYCLE_WAIT:int	[0; INT_MAX]	(3)
GET_CAR_SPEED_MAX	-	(3)
GET_CAR_SPEED_MIN	-	(3)

command	value range	answer type
GET_INFO_SIMTIME	-	(4)

All the commands ends with `\n` and inevitably lead to an answer.
If *n4s* doesn't receive a command, it won't send out an answer.
Commands are not case sensitive.

For example, CAR_FORWARD takes a parameter that does indicate the engine power you want to drive with (and not absolute speed):

`CAR_FORWARD:0.5\n` makes the car move forward at half of its maximum speed.

Responses

Here are the different answer formats, corresponding to each type:

- ✓ (1): VALUE_ID:STATUS:CODE_STR:ADDITIONNAL_INFO
- ✓ (2): VALUE_ID:STATUS:CODE_STR[:float]*32:ADDITIONNAL_INFO
- ✓ (3): VALUE_ID:STATUS:CODE_STR:float:ADDITIONNAL_INFO
- ✓ (4): VALUE_ID:STATUS:CODE_STR:[long,long]:ADDITIONNAL_INFO

VALUE_ID indicates the response code. The different values are listed below.
STATUS is whether "OK" or "KO". It indicates if the command execution is a success or a failure.
CODE_STR corresponds to the verbal version of VALUE_ID. The different values are also listed below.
ADDITIONAL_INFO can contain information concerning the last checkpoint passed on the track (followed by the id of this checkpoint and the passage timestamp). The four different types of checkpoints are as follows:

Message	Explanation
First CP Cleared	first checkpoint passed
CP Cleared	passing a checkpoint (other than the first and last on the track)
Lap Cleared	complete lap
Track Cleared	end of the race. You need to stop the car and the simulation

VALUE_ID/CODE_STR correspondance:

VALUE_ID	CODE_STR
0	Simulation has not been launched
1	No errors so far
2	Simulation running
3	Error. No details can be provided atm
5	Checkpoint error detected
6	Network operation failure
7	Server-side Error
8	Client-side Error
9	EOF reached
10	Simulation was correctly ended
11	Empty command
12	Unknown command
13	Wrong arguments provided. Please refer to protocol
14	Too many args provided with the command. Please refer to protocol
15	Pipeline failure
16	Unexpected command argument's value
17	Camera infoget failure
18	Command not found
19	Simulation already up and running
20	CYCLE_DONE
21	Sensor reading failure
22	Scene must contain at least 3 CPs
23	Timer Init Failure
24	Timer get Failure
25	Failed to load Map

Example

Here's an example of the *n4s* binary usage in standalone, in order to test the API:

Terminal

```
~/B-AIA-200> ./n4s
start_simulation
2:OK:Simulation running:No further info
stop_simulation
10:OK:Simulation was correctly ended:No further info
Last state registered: 10:OK:Simulation was correctly ended
```

FAQ

What is the viewing angle?
60 degrees, first-person viewpoint

What is the LIDAR unit of measurement?
Millimeters.

What are the car's dimensions?
The car's dimensions are not exactly known and can vary from one model to another (and very certainly between the virtual model and the real car). Try to have an AI that's intelligent enough to function without this information. There are very simple ways to do it.

How much does the car weigh and what is its maximum speed?
As with the previous question, this is information that you don't necessarily need. What's more, you just need to take measurements when the car starts up in order to deduce its speed.

How many drive-wheels does it have and at what angle is the car able to turn?
Ditto.

Why don't the values returned by the LIDAR exceed 3010?
A real LIDAR uses a limited-reach laser. In order for the simulator to be as close to a real environment as possible, we have curbed the measured distances to this limit. In other words, when an obstacle is "detected" at a distance of 3010, it means that this obstacle is more than 10 feet from the car.

What do the two values in the type 4 response correspond to?
They give, in seconds and nanoseconds, the time that has passed since START_SIMULATION.

What is the evaluation really looking for?
An AI that is able to do laps, rather quickly, on the track (to give you a hint, tell yourself that a lap done in 5 minutes on a small track is much too slow). We're also looking to see that your AI will stop driving and terminate the simulation in a dead-end. In this case, it must stop at least three feet from the wall, without touching it, before terminating the simulation.

v 3.1