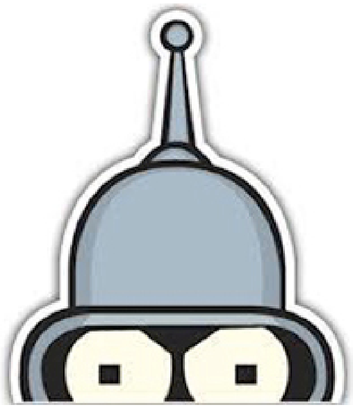

# B2- Elementary Programming in C

B-CPE-156

# Need4Stek

Autonomous Car Concept

v1.1

# Need4Stek

## Autonomous Car Concept

|  |  |
|---|---|
| **binary name**: | ai |
| **repository name**: | CPE_$YEAR_n4s |
| **repository rights**: | ramassage-tek |
| **language**: | C |
| **group size**: | 2-3 |
| **compilation**: | via Makefile, including re, clean and fclean rules |

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- 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).

**Authorized functions**: Every libc and libmath functions.

The Makefile and the generated binary must be at the root of the directory.

> *Google self-driving cars are any in a range of autonomous cars, developed by Google X as part of its project to develop technology for mainly electric cars. The project was formerly led by Sebastian Thrun, former director of the Stanford Artificial Intelligence Laboratory and co-inventor of Google Street View. Thrun's team at Stanford created the robotic vehicle Stanley which won the 2005 DARPA Grand Challenge and its US$2 million prize from the United States Department of Defense. The team developing the system consisted of 15 engineers working for Google, including Chris Urmson, Mike Montemerlo, and Anthony Levandowski who had worked on the DARPA Grand and Urban Challenges.*
>
> *In May 2014, Google announced plans to create a driverless car that had neither a steering wheel nor pedals, and unveiled a fully functioning prototype in December of that year that they planned to test on San Francisco Bay Area roads beginning in 2015. Google plans to make these cars available to the public in 2020.*
>
> Wikipedia (Oct. 2016)

We know Google is very active in the autonomous car market, but it is not the only company to position itself within this market niche; in October 2015, Tesla Motors announced a software update had equipped certain electric car models with an autonomous driving system.

> *Tesla announced today that all of its cars will have a new suite of self-driving hardware as standard, enabling a path to fully autonomous driving. The announcement applies to the Model X, Model S and upcoming Model 3, which should come out in 2018.*
>
> *The AutoPilot feature supports adaptive cruise control and lane keeping assistance, giving the cars nearly self-driving capability. It also enables automatic emergency braking and collision warning.*
>
> www.cnet.com/roadshow/

These two companies have chosen two different methods to create this technological prowess.
Among the car's integrated elements for detecting the road and obstacles (other cars, cyclists, children crossing the street,...), Google decided on LIDAR.
It's a censor that detects the distance between the car and environmental obstacles in order to create a 2D or 3D representation of the real world (which allows a decision to be made: accelerate, turn, brake,...).
Several models exist, with prices ranging from a hundred or so dollars to over $80,000, depending on the desired performance (motor rotation speed and acquired measurements, minimum and maximum distance, precision, ability to create a 2D or 3D representation etc.).

According to the CEO of Tesla Motors, Elon Musk, Tesla thinks that the LIDAR is a tool that is not adapted to autonomous driving, and has opted for the Mobileye technology, which you can check out on the manufacturer's official site.

These two agents are often quoted because they are the current leaders of the market, but motivated programmers could give them competition for next to nothing in costs. Actually, George Hotz (the first hacker to have jailbroken an iPhone) taught Google and Tesla a lesson. At the tender age of 26, Geohot, himself, built a car that was capable of taking the reins on a highway (see here). He might not have implemented automatic warning sign reading or pedestrian detection (in order to avoid accidents), but his proof-of-concept is quite a feat, especially for someone without a huge multinational budget.

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:

- V-REP 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.

# V-REP

## 1- Installation

Download the PRO EDU version of the V-REP 3.2.3 in the version that corresponds to the machine's architecture (the assets provided for the project function solely with Linux and Mac). In order to launch V-REP, you need to execute the onboard script:

```
Terminal
~/B-CPE-156> ./vrep.sh
```

## 2- Scene

Download and then unzip the latest version of the `B-CPE-156_Need4stek_package.tgz` from the intranet (file provided with the project description, can be updated if needed).

Miscellaneous ".ttt" files are located in the "scene" file. These are the files that V-REP 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 V-REP (drag 'n drop or via the "File > Open scene..." menu).

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

# API

## 1- Overall Functioning

The n4s binary enables communication with V-REP (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. Put your ai binary into the same file as the n4s binary that is provided for you. You'll need to execute the pipes.sh script in order to launch the simulation and see how it interacts with your AI.

## 2- 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\|] | (1) |
| GET_CAR_SPEED_MAX | - | (3) |
| GET_CAR_SPEED_MIN | - | (3) |
| 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.

## 3- 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:

| | |
|---|---|
| *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 |
|---|---|
| 0 | Simulation has not been launched |
| 1 | No errors so far |
| 2 | Simulation running |
| 3 | Error. No details can be provided atm |
| 4 | Checkpoint error detected |
| 5 | SimComponent Loading Failure |
| 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 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 |

## 4- Example

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

```
~/B-CPE-156> ./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
```

# A bit more

## 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 Autograder 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.

## Bonuses

Feel free to customize this project and have fun with it! You could for instance:

- create a new scene (that functions with n4s),
- avoid obstacles,
- parallel parking,
- implement your program on a model car,
- ...