Documentation LEC 2017

-Adaptive Cruise Control-

Team name: RS Lang

Team members:

- Alexandru Rusu
- Alexandru Stahie

1. Short description

In our project, we realized a proof-of-concept car model powered by UDOO Neo board, which is capable of remote control (through Wi-fi), at different constant speeds that can be changed by its user from a specific interface. Also, it can move in all directions, backwards or brake when it receive the specific command, this should be considered more as a simple cruise control mechanism. As it is said in milestone 3, we added a video camera to the system which allows us to see where the car goes from the user interface. This camera, alongside with some image processing algorithms in OpenCV make it possible to detect the other cars in front of us by finding the vehicle number plate and using it for keeping the safe distance from that car, distance that can be set by user to be closer or further. In the same time the application has implemented a lane detection algorithm that is used for keeping the motion lane, this mechanism being very useful for car detection as well as regaining the control of the car after the object avoidance. In other words, we added the adaptive concept for our cruise control mechanism.

In addition, we created another car model, this time powered by a Raspberry Pi 3B board together with an Arduino Uno, with the same specification as the first one and on which runs the same programs, with small changes of course. We used the second car mainly for comparing the architecture behavior of each car, but also for easing our work with car detection (in milestone 3).

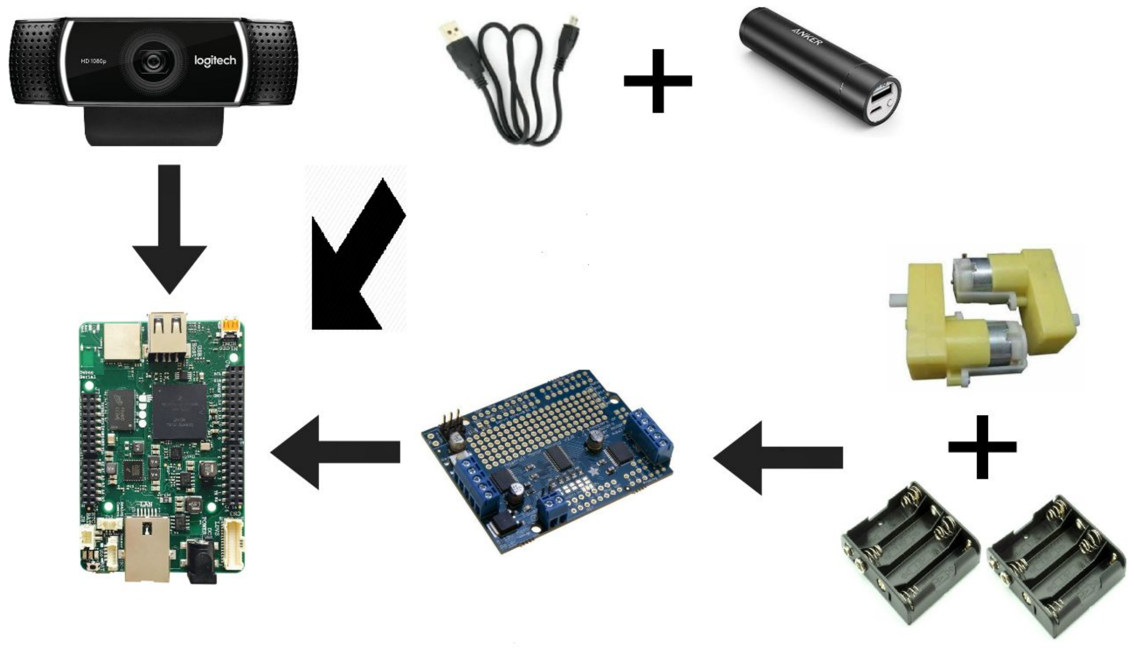
2. Used components and the hardware architecture

For the first car, the one powered by UDOO Neo the used components are:

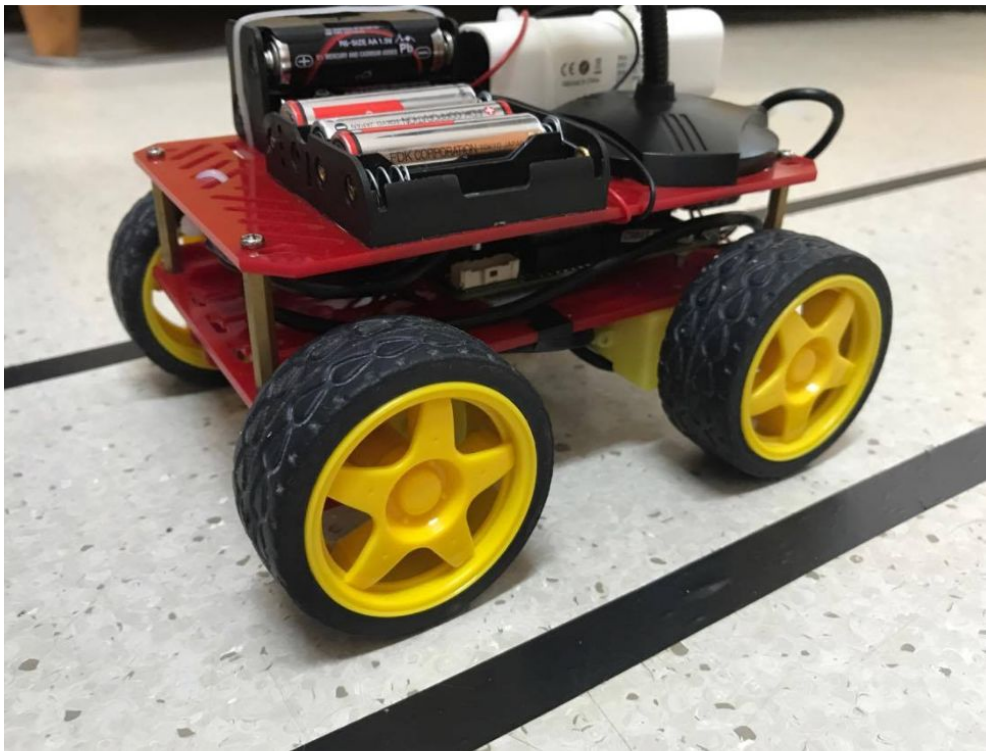
- UDOO Neo board
- Adafruit motor shield V2
- Robot kit with 4 motors (which contains)
    - Two chassis
    - Four wheels
    - Four motors
    - Motor supports;
    - Screws, nuts needed for assembling.
- 2 sockets for 4 AA batteries for motors
- A webcam
- External battery for powering up the UDOO Neo board

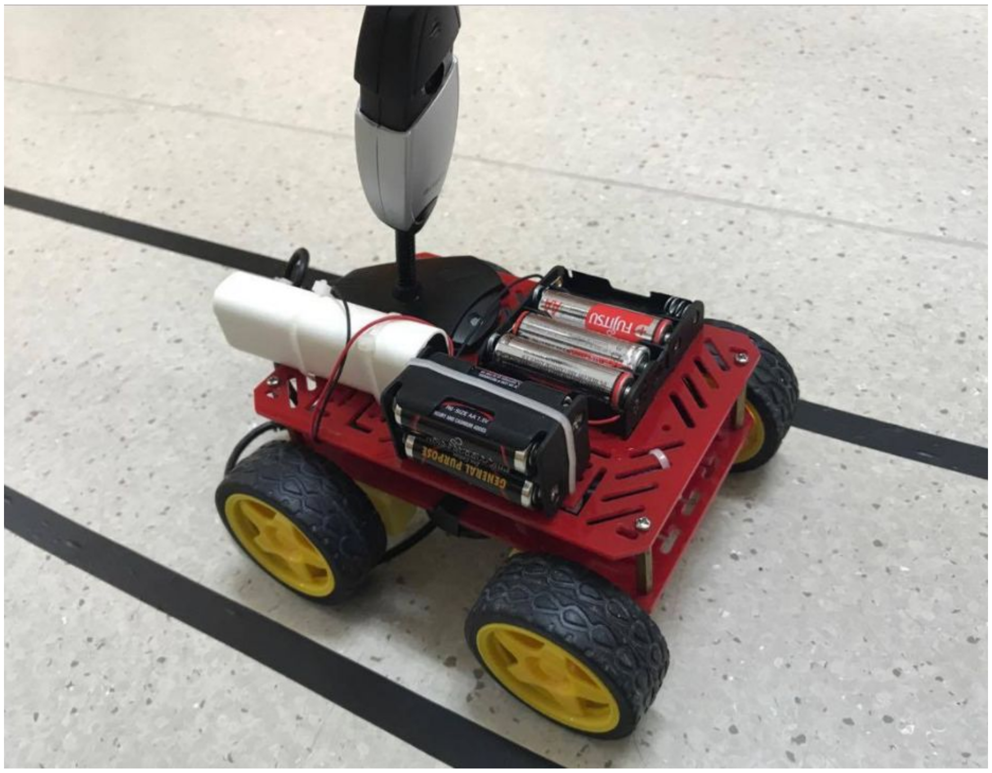
For the second car, the list of used components is mostly the same, with the difference that we used Raspberry Pi 3B board together with an Arduino Uno for powering it up, another difference is that we used Adafruit motor shield V1 for Arduino, instead of V2. Here we also have 1 socket for 4 AA batteries and another 2, each one for 2 AA batteries.

The hardware components assembling is quite easy for each car. Firstly, we stuck the wires to the motors and assembled the car kit. After that, we placed each component on the kit in the best fitting way possible, making sure that in the end we can connect each component and the wires length permits it. As a short description, we connected the motors and all the sockets for AA batteries into the adafruit motor shield, shield that is positioned on the UDOO Neo board. The external battery is used for powering up the board and the webcam is connected to the UDOO by its only USB connector (exemplification in the photo below). This procedure was used for both cars, with the remark that on second car we placed the shield of Arduino Uno board, which is connected to Raspberry, as well as the webcam.

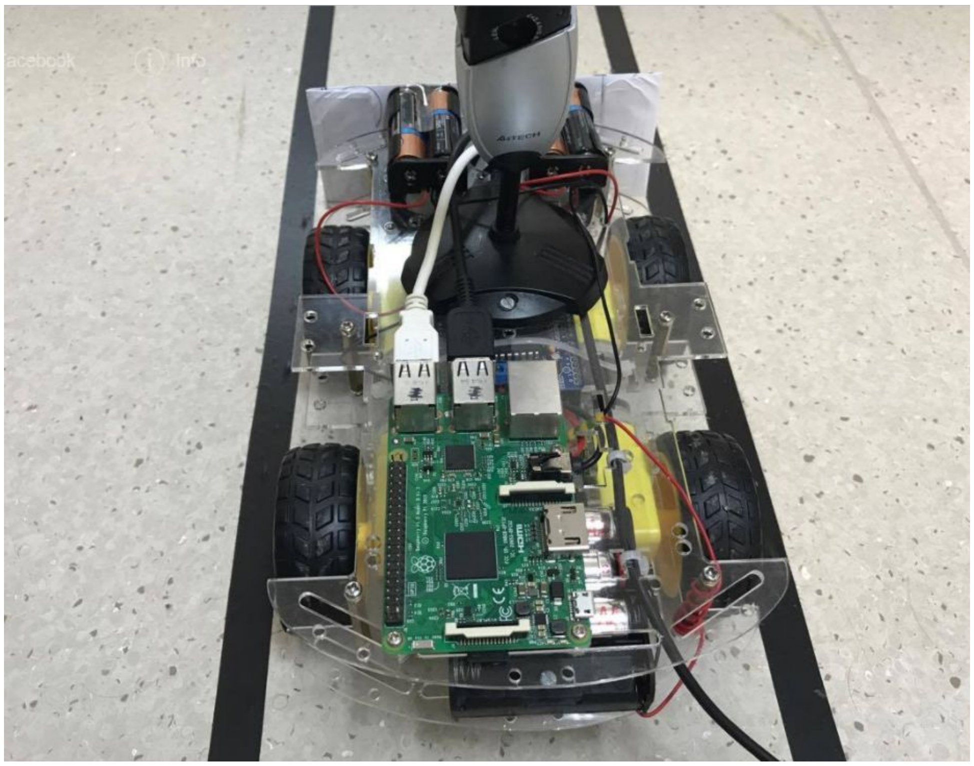


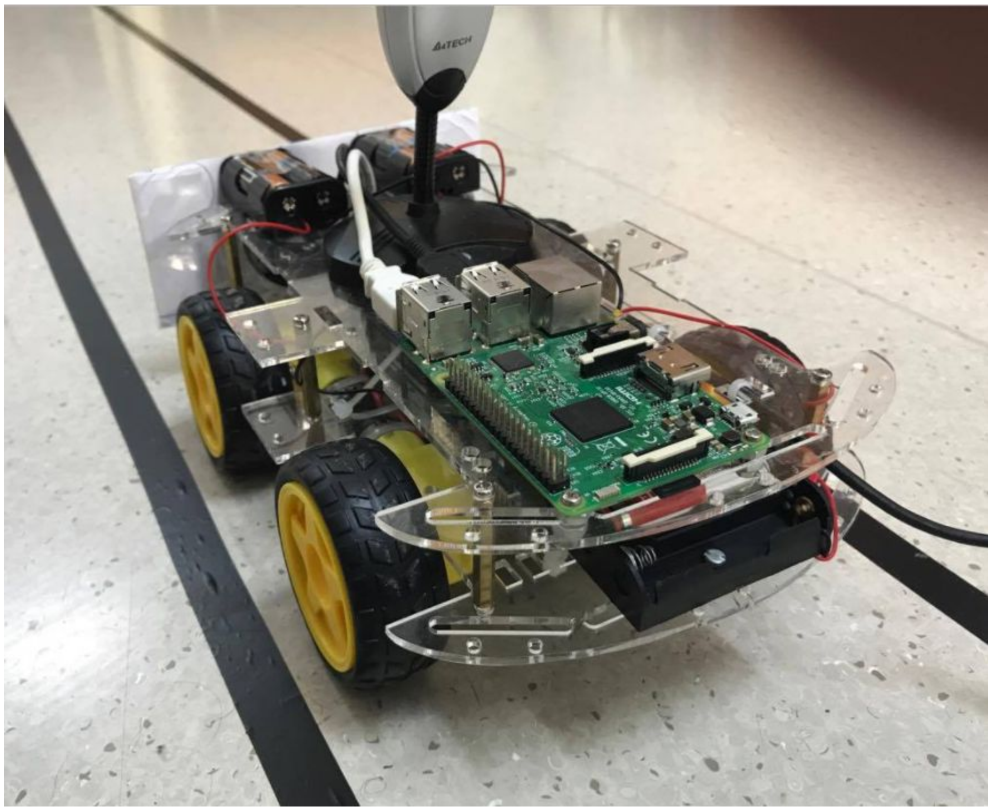
As a result, this is how the car powered by UDOO Neo looks like:



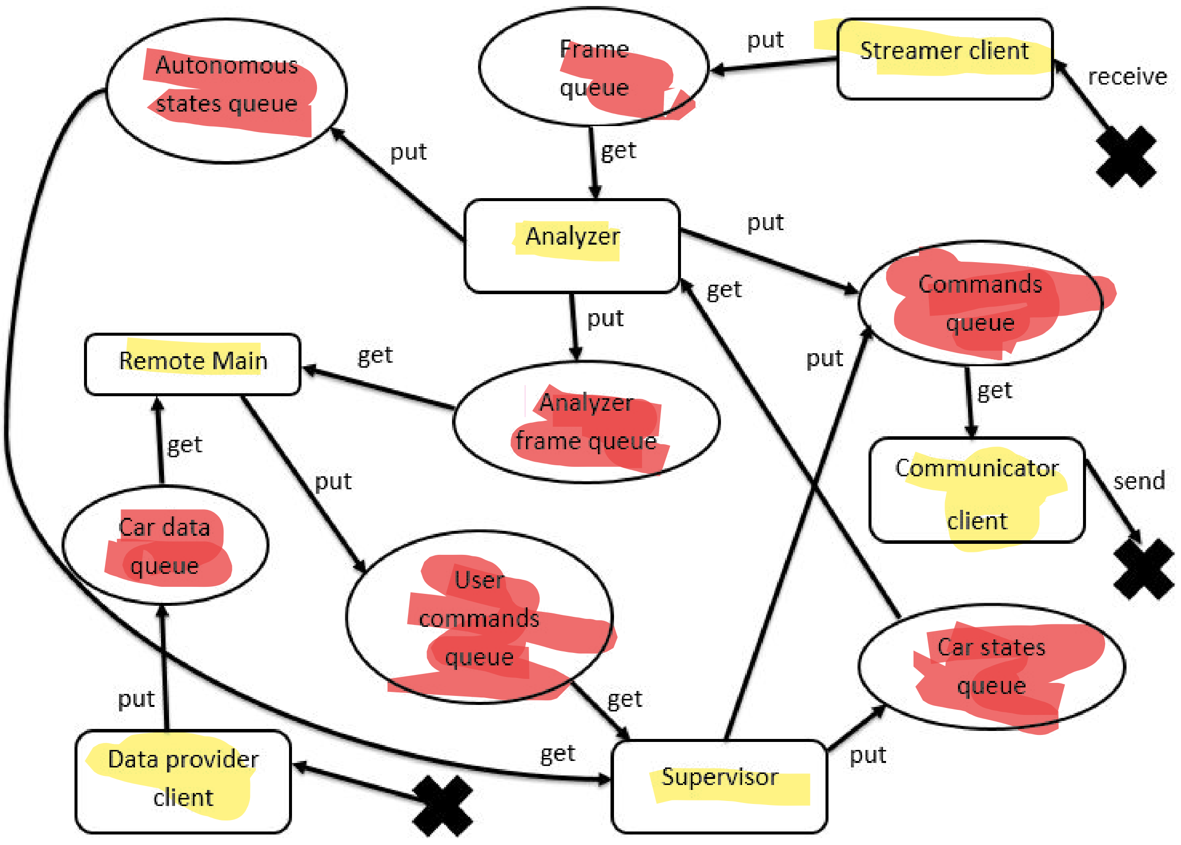


And this is how the second car looks like:





3. The software architecture



Legend:

✖ = Our car

◯ = Memory queue

▢ = Classes

Put and Get command = memory shared between threads. (in this way, the information flow through the application without blocking or slowing the fps rate).

The Remote Main class represents the principal component of the application. There, all the needed threads are launched and synchronised with the help of some memory queues.

Remote Main launch three threads that each represents a socket-based class which will connect to the application that runs on the car. Those client-server connections are used to receive frames from car's camera, send commands to Arduino controller and receive usefull informations about the car(speed, current action, etc).

Every of these clients classes will store or receive the data that is designed to manipulate through memory queues. So, basically, the entire application will share the same memory between all threads that have access to that queues.

Analyser is the heart of the project and also the basic pillar of the application architecture. It has access to the Frame Queue and after it applies our algorithm with OpenCV concepts, it will determine which is the best move that the car has to do in her future step. Therefore, the Analyser will have access to Commands Queue, and to make the user experience even more pleasant, it will populate the Autonomous States Queue that the Remote App will use to show useful and concrete information.

After all the needed commands have been stored in the suitable queue, the Communicator will have the role to send those commands to the Car for being executed. Commands like: go_forward, brake, go_left, etc will be send to the Car application and the car has the the obligation to execute them.

In case of user input in terms of car control, the Remote App will not allow the user to control the car, if the Analyser consider that the user is not allowed.

After every command sent, the Analyser will have to know the current states of the Car, because we can not assume to make approximation in terms of the car current state. That said, the Data Provider will have the role to receive those informations that are sent directly from Arduino Chip and give those states to be accessible for Analyser.

In the end, the Supervisor is the watcher of the entire app and its delegate to update all the needed information to the user. The speed of the car, the preferred speed, the detection warnings and so on.

The application that runs on the car is multithreaded, you can also conclude that from the previous description and it will have the proper servers for every client of the Remote Main. Besides those servers, the Car application has a Serial Manager class which represents the direct communicator with the Arduino Chip and a Recorder class that will get continuously frames from the attached camera. As you can conclude, every class got a thread-based architecture and will communicate with the rest of threads via memory queues.

The memory queues used in that application are the following:

- a Frame Queue which will contain every frame captured by the Recorder
- a Command Queue which will contain every command that the Arduino Chip has to execute
- a User Commands Queue to know when the Car application should stop or launch a specific functionality
- a Car Data Queue which will be populated with every states and speeds that the car has been taken across the running.

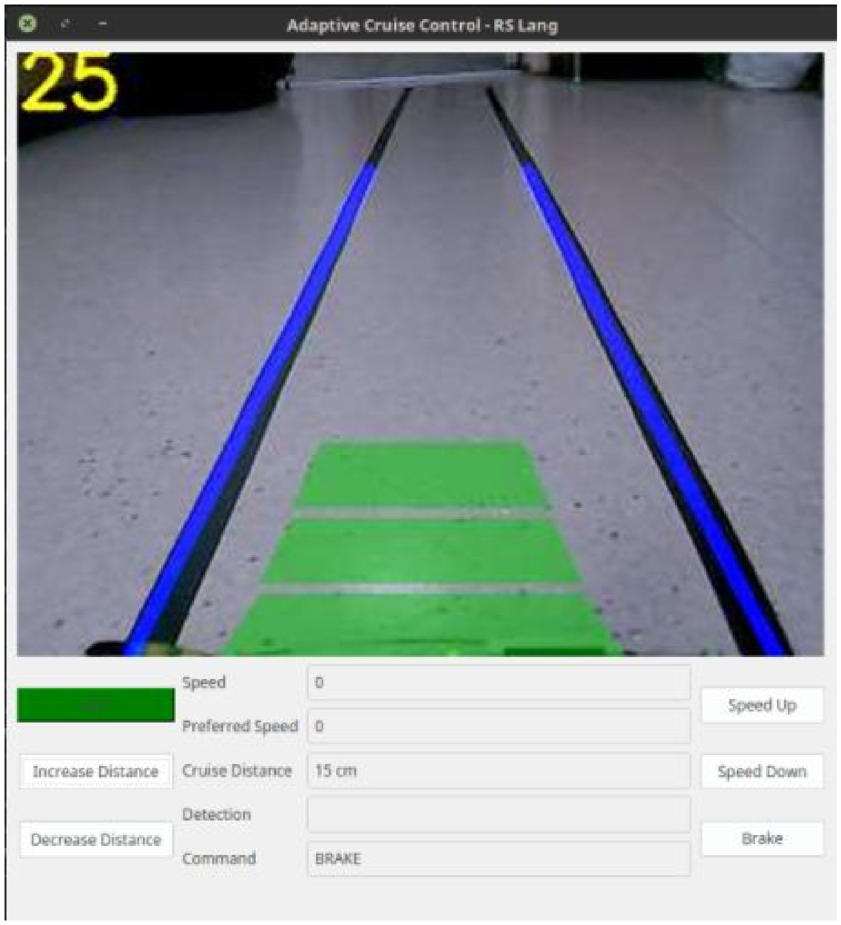
4. User guide

We are supporting that our project is quite easy to be used. That's because of the user interface that mainly control everything. Here we display the video streaming that we receive from the webcam (in the top left corner of the video streaming we show the fps that we have for the moment). We also display some important parameters like:

- Speed (where the user can see the current speed of the car)
- Preferred Speed (here we can set the speed that we want to have)
- Cruise Distance (the distance that we want to keep from the car in front of us)
- Detection (we display information received from adaptive cruise control like "Car ahead" or "Object detected")
- Command (here is the command that the car executes)

In the left side, we have the "ACC" button that activates or deactivates the Adaptive Cruise Control and the Increase/Decrease Distance Buttons, used for modifying the distance from the detected car, it can be set by user to be closer or further.

In the right side, there are the control buttons that help use controlling the car. The car can also be controlled from keyboards "W", "A", "S", "D", "R" (for going back) and "M" for brake.


Adaptive Cruise Control - RS Lang

25

Speed	0	
Preferred Speed	0	
Cruise Distance	15 cm	
Detection		
Command	BRAKE	

5. Steps and problems

Regarding the followed steps for realizing the project, we can say that the requirements were well explained, and the milestones that we had represented a very good guide for us.

About the problems we have had, there were a lot of them starting from small bugs in the program and reaching up to some serious problems that took us some more time to fix. Some of the most serious issues would be:

- The incompatibility of components: for example, the shield we use (adafruit motor shield V2), works at 5V, but in the same time the UDOO Neo board pins work at 3.3V. As a solution, we adapted the shield for 3.3V.
- The light problem: the webcam that we used is not a very good one, but helped us get a good number of fps for live streaming, even so, OpenCV and the algorithms that we used needed some more light, so we adapted to the situation.
- Detecting and measuring the distance from objects that we detect, it is hard to do things that are usually made with sensors, so we used some etalons for image processing programs. We used black lanes for lane detection, vehicle number plate for detecting the car in front of us, and round object which represents the obstacles that must be avoided.