**Self-driving car prototype using Raspberry Pi platform**

Cristian Toma1, Alexandru Florentin Iftimie1

1 The Bucharest University of Economic Studies, Faculty of Cybernetics, Statistics and Economic Informatics, Virgil Madgearu building, Calea Dorobantilor 15-17, 010552, district 1, Bucharest, Romania

[cristian.toma@ie.ase.ro](mailto:cristian.toma@ie.ase.ro), [iftimie.alexandru.florentin@gmail.com](mailto:iftimie.alexandru.florentin@gmail.com)

**Abstract.***This paper presents the attempt to implement and test the capacity of neural networks in reproducing the human behavior for autonomous vehicles by prototyping on a modified RC vehicle. Among the objectives is counted the training of a neural network to estimate the steering angle so that the car is able to keep the lane and to train a neural network for vehicle detection in an image. The information obtained from the last neural network is fed into a Kalman Filter that is used to track and estimate the trajectory of the respective vehicle in order to avoid a possible collision. Because the project is based on a Raspberry Pi platform and it’s processing capacity is very limited, we proposed the implementation of a client-server application. The server would be taking the images from the webcam that is mounted on the car and send them to a more powerful computer in order to process them in real time.*

**Keywords:** IoT, Autonomous Vehicles, Tensorflow, Neural Networks, Computer Vision, Raspberry Pi

**1**

**Introduction**

The ideea of creating autonomous vehicles has been a subject of interest for over a century. In the same time with the advance of hardware and computing power, more complex algorithms have been developed around the area of Computer Vision and Machine Learning.

The first recorded self driving car was in 1989 when a group of researchers from Carnegie Mellon University developed ALVINN which stands for Autonomous Land Vehicle In a Neural Network. The ideea behind was to feed the images with a resolution of 30x32 pixels from a camera facing the road and images from another camera of 8x32 pixels for range finder to a Multilayer Perceptron which was trained to produce a 45 output units vector that represented different steering angles [1]. This type of approach would be later called Behavioural cloning or End-to-End driving.

Recently Nvidia, a graphics cards producer, released a video of their self-driving car that used only a heavy Convolutional Neural Network with over 27 milions of connections without no other robotic techniques, which was fed with images from 3 cameras and succesefully drove on the road without human intervention [2].

The main purpose of this research is to reproduce the above results using IoT embedded platforms such as Raspberry Pi 3 mounted on a Remote Controlled car chassis, that has connected a RGB camera.

The features included in this project are:

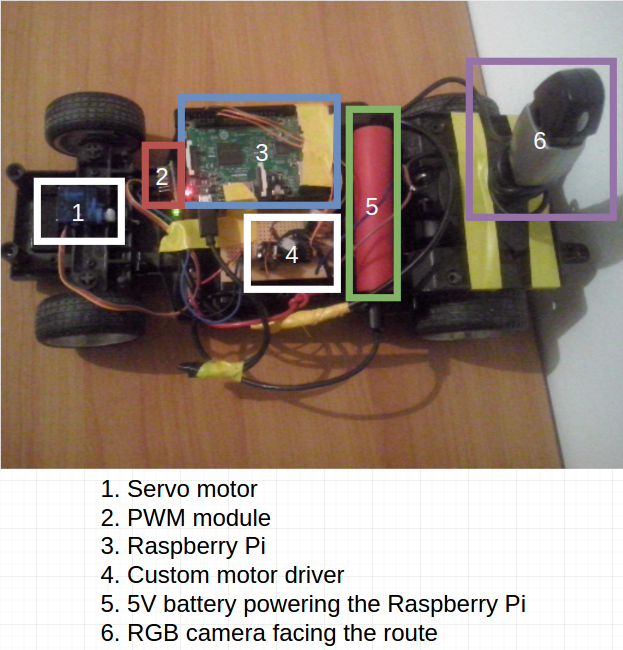
* Training a neural network for predicting steering angles.
* Tracking other vehicles on a sequence of images using a neural network trained for vehicle detection and a Kalman Filter [3] with Kuhn–Munkres algorithm for object tracking, and using that information for predicting whether another car will collide with the driver’s car.
* Creating a client-server application, the server running on Raspberry Pi awaiting for “worker client” and the client being the laptop that processes the received images and responding with commands

In order to achieve these results, traditional Computer Vision techniques had to be studied such as feature extraction, object detection and image recognition which proved to be useful but not very robust for a self-driving car, and also modern Machine Learning approaches were tested such as Neural Networks which proved to be easy to implement and train and much more robust to new scenarios.

**2 Solution Components**

The main object of the project is the modified RC vehicle. It’s original inner circuits were removed and replaced with custom built electronics. The project used the following parts:

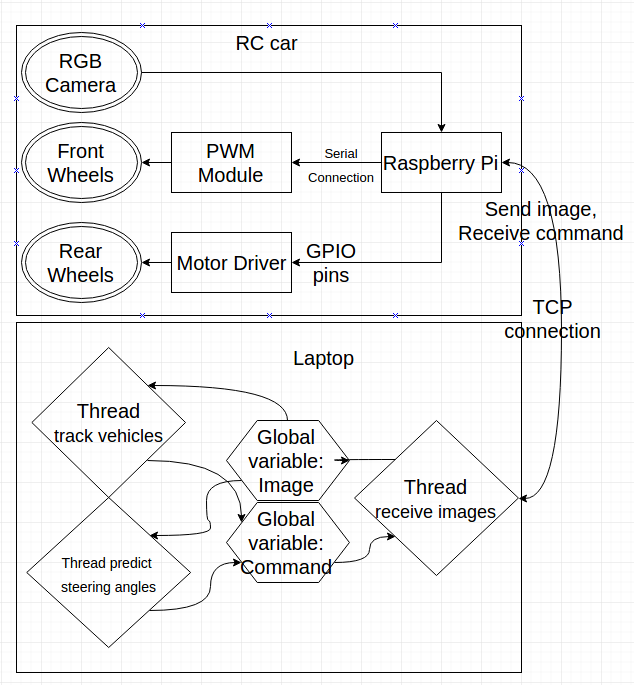
* Raspberry Pi model 3 B+. The only responsability of the Raspberry Pi is to act as an intermediate between the hardware parts and a more powerful computer, as it’s resources are not enough to process all the information coming from the RGB camera. Thereby a server application running on the microprocessor was developed that is capable of listening for “worker client” in order to distribute tasks such as steering angle prediction and vehicle tracking.
* Custom built motor driver was soldered on a prototype board to control the rear wheels. A signal from the GPIO pins of the Raspberry Pi is connected to the LD293D pin to enable the wheel to rotate în both directions
* A servo motor was placed to control the front wheels for a larger set of angles to make the car more realistic as it previously had a simple left-right-center motor
* A PWM microcontroller is used to take the load from the Raspberry Pi of creating software threads that switch the state of GPIO pins very quick to produce a PWM signal
* A RGB camera is the only sensor device of the car, which will prove that it can be enought to give the car some intelligence
* One 9.6 NiMh battery and one 5V battery
* Laptop with Nvidia CUDA enabled GPU for realtime processing of the images



**Fig. 1.** Solution Components

There are two main components to this solution. The first one is represented by the server running on the modified RC Vehicle and the second one is the client running on the laptop.

The control of the wheels is done in two ways. The first one is to use wiringPi library for C programming language which allows for direct control from the GPIO pins. This method is used for the rear wheels. The other method which uses a dedicated hardware specifically a PWM module that receives the code using a serial communication and outputs a PWM signal ranging from 1 ms - 2 ms pulse with a duty cycle of 20 ms which is perfect for controlling a servo motor. This expensive piece of hardware was necesarry beceause the wiringPi library was using threads that can be interrupted by the operating system thus producing a delay of 1 microsecond that can make the the servomotor jitter. This phenomenon was observed when the server application started, and little jitter was present in the servo motor, but as soon as TCP connections were established and images were streaming, the servo motor was shaking uncontrollably. After the upgrade with the PWM module, no jitter was observed since.

****

**Fig. 2.** Solution Architecture

The responability of the client application is to process the images using the trained models to predict the steering angle and to track other vehicles. In order to achieve that, as soon as the client application is lauched the following threads are started:

* One thread which connects to the server and receives the image into a global variable and sends the command from a global variable. The default command is start moving forward at a constant speed with front wheels straight
* One thread that pools the image from the global variable and feeds it into a neural network for angle prediction. The output is sent to the command global variable
* One thread that pools the image from the global variable and tracks the vehicles

**3 Functional Architecture and Data Flow**

As soon as a new TCP connection is established, the server awaits for a command. The protocol of receiving commands and transmission of images is the following:

* The server awaits 4 bytes that represent speed value (1 byte), a code that represents forward or backwards moving direction (1 byte), a value ranging from 4000 to 8000 that represents the steering angle (2 bytes, short int)
* After receiving the command the server processes them and sends the code to the wheels
* First is sent the command to the front wheels through the serial communication
* Second is set the pusle width to the software threads for controlling the speed of the rear wheels
* After processing the commands, the server grabs a color image of size 640x480 from the RGB camera and compresses the image to JPG format, thus reducing the size from 1 Mb to about 60 Kb, allowing to stream multiple images per second as the maximum upload rate of the incorporated network card is about 1.3 Mb, the target of frames per second sent to the client ibeingat least 20 in order to achieve good control.
* The bytes of the compressed image is sent to the worker client and the loop repeats

From the client point of view the application will start the main thread that will open other 3 threads.

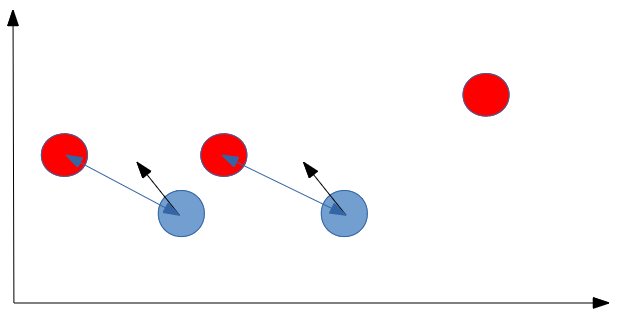
The first thread will declare a global variable for the command and a global variable for the image then it will connect to the Raspberri Pi server and will send in a loop the current command, receive the image size then the entire image that is compressed în JPEG format. It will uncompress the image and then it will place the data in the global variable.

In the same time another thread specialized into estimating the steering angle. First, it will load the model parameters and then, in an infinite loop it will get a copy from the global variable image, will crop the image from row 240 to row 480 so that only the bottom part will remain that contains the road, thus eliminating the redundant information. The next operation will be to convert it into grayscale which will average the pixels on the 3 channels. After the image has been preprocessed it will be fed into the neural network and the output of this model will be a value ranged from 4000 to 8000. This value will update the global variable command and the other thread will send the command.

The third thread that will be created is specialized into detecting other vehicles in an image. First it will load the model parameters and then, in an infinite loop, it will grab the RGB image from the global variable and pass it with no other preprocessing technique to the neural network. The neural network will output a binary image and with the help of some heuristics the algorithm will find blobs in the binary image and will decide the location of the objects in the image.

After the objects have been detected, the center points will be fed to the Kalman Filter which in turn will have a few operations that will keep track of the points in time. First, a matrix of distances will be computed. On the rows there will be the points from the previous frame and on the columns there will be points from the current frame. After the creation of this matrix it will be padded with columns with zeros if the number of detections is less than the number of tracked states otherwise the algorithm will not work.

The matrix will be then fed to Kuhn-Munkres optimization algorithm that will find which detections belong to which tracked states in order to minimize the total distance between the points.

****

**Fig. 3.** Kuhn-Munkres association stage

In the figure above is an example of how the Kuhn-Munkres algorithm works. With the blue circles are represented the tracked points and their trajectory with a black arrow. The red circles are the current frame detected objects, and with the blue arrows are the associations between the tracked points and the detected points. In this figure in order to minimize the total distance between points, only two of the tracked states will be updated with detections, and the third detected point which is further away will be transformed into a new tracked state.

**4** **Training a steering angle predictor**

No car can be named autonomous if it does not have a system capable of keeping the car on the lane and the best systems have been proven to be the neural networks as these models have outperformed every other traditional computer vision techniques.

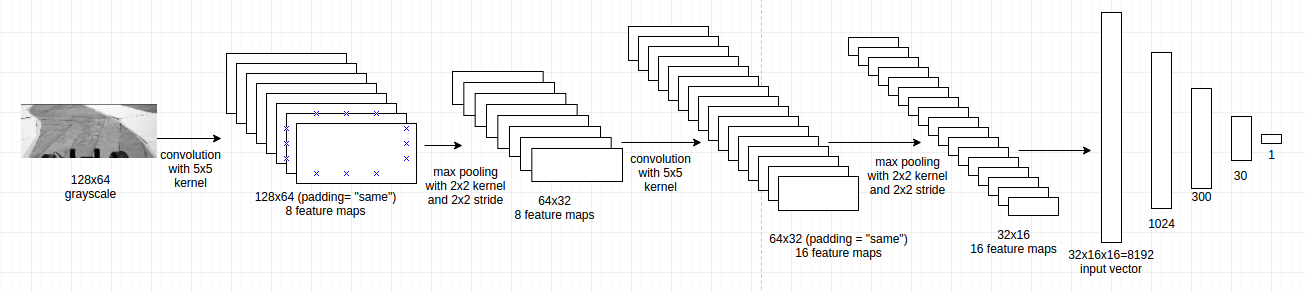
In the search of a model based on neural networks to predict steering angles, multiple experiments have been made. Initially I tried to to train my own implementation in C++ of a neural network which was a simple Multilayer Perceptron with an output of 11 different discrete angles and another convolutional neural network with the same number of outputs. Although this approach achieved high accuracy on the training set and on the test set (about 93%) it was proven to be unsuccessful in a real test as the front wheels were shaking too much and were not even keeping the lane. Summarily, a classification neural network would not fit the needs of this project.

The only neural network that was proven to be successful was the one trained for regression. This means that it has only one output with a continuous value ranging from -10 to 10 (this interval was an arbitrary choice).

The convolutional neural network architecture chosen for predicting the steering angle was chosen as follows:

* convolution layer with a 5x5 kernel and tanh activation function which received 1 input channel of size 128x64 (the image in grayscale) and produced 8 different output channels / feature maps of the same size (this is used for feature extraction)
* max pooling layer with a 2x2 kernel and a stride of 2x2 which received 8 feature maps and produced 8 feature maps (this is chosen for dimensionality reduction from size 128x64 to 64x32)
* convolution layer with a 5x5 kernel and tanh activation function which received 8 input channels of size 64x32 and produced 16 output channels of size 64x32
* max pooling layer with a 2x2 kernel and a stride of 2x2 which received 16 feature maps and produced 16 feature maps (to reduce size from 64x32 to 32x16)
* a layer to merge the previous feature maps into one big vector of size 8192
* fully connected layer with 1024 neurons and tanh activation function
* fully connected layer with 300 neurons and tanh activation function
* fully connected layer with 30 neurons and tanh activation function
* 1 output with linear activation function

The neural network was implemented using Tensorflow library in Python for the simplicity that it offers and for the GPU acceleration support coming together with the graphics card.



**Fig. 4.** Convolutional neural network for steering angle prediction

In order to collect data for training the steering predictor, an application was developed which allowed the user to control the RC vehicle using the mouse position on a label. The format of the data was in two separate files, the first one being the AVI file which contained the sequence of images and the second one was a text file containing the corresponding angle of the wheel whose values ranged from 4000 (meaning maximum right) to 8000 (maximum left).

The images were stored with a resolution size of 640x480x3, thus it was needed the conversion to grayscale, and selecting the Region Of Interest from point (0,230) to point (640,480) resulting in an image of size 640x250. Further more, the images were resized to 128x64 to be a valid input to the network. The input images had pixel value intensities ranging from 0 to 255. Such values are not suitable for a neural network thus, they had to be scaled to -3, 3 (this was an arbitrary choice). The output ranging values is also not suitable thus it had to be scaled from 4000, 8000 to -10, 10.

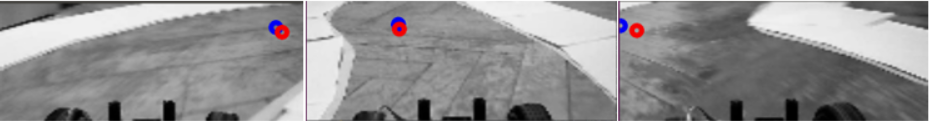
The dataset was split into 2 parts. A training set, a validation set. A validation set is used for stopping the training of the network in order not to overfit the training data. The training of the network will stop when the total cost of validation set is 120% bigger than the cost of the training set. The cost is computed as the mean squared error, where  is the label value and  is the estimated value:

Training the neural network was lasted less than an hour, and it was stopped when the cost of the network was not decreasing anymore. The learning rate was adjusted permanently in order to reduce the training time. Initially the learning rate had a value of 0.05 which is usually considered very high. It allowed the network to learn fast the patterns of the network, but after a few iterations as the cost was bouncing around a local minimum, the learning rate was diminished to 0.005 and the process was repeated until a learning rate so small was selected that the training time would have required many more hours to train, as the changes to the weights were precise but small.

The images used as training data were saved in a video file format accompanied with a text file containing the label value for every image in the video file. As the training data was relatively small, the entire dataset is loaded into memory and it is preprocessed by scaling every pixel value from [0,255] to [-3,3] and also every label value is scaled from [4000,8000] to [-10,10] this preprocessing is recomented in order for the network to become more robust. By working with smaller values, the weights will be updated much easier with higher learning rates. Another preprocessing technique that was applied to the training data was that for every image in the training set a flipped version of the image was created, and for every image in the new dataset, small variations in the brighness of the images were applied in order to simulate the fact that the car could be trained for environments at different time of the day when there is more or less light.

Another problem that migh cause the network to learn improperly is the fact that the training data images are not uniformely distributed. That means that for most of the time the steering angle of the car is kept straight when data is collected, and less images are collected when the car is actually turning. To resolve this problem the steering angles were split into 11 bins and for each bin it was counted how many images were selected for that particular discrete angle. Then the maximum value from the 11 bins was selected and for each other bin, a clone was generated from the images that coresponded to the respective bin until each bin reached the same maximum value. Although this method will greatly increase the training data with redundant images, at least the neural network will not be biased by the greater number of images that have the wheels straight.

The mini-batch size for this session of training was selected arbitrarily as 11. This value also represented the number of discrete angles that was selected for a previous experiment with classification neural network that proved to be inneficient.



**Fig. 5.** Steering angle prediction

The results after training the neural network were very satisfying as the MSE was 0.22 for a training set with 3000 images. To visualize the results the following images have been provided. The blue circle represents the ground truth angle mapped to the width of the image, and the red cicle represents the estimated angle mapped to the witdth of the image.

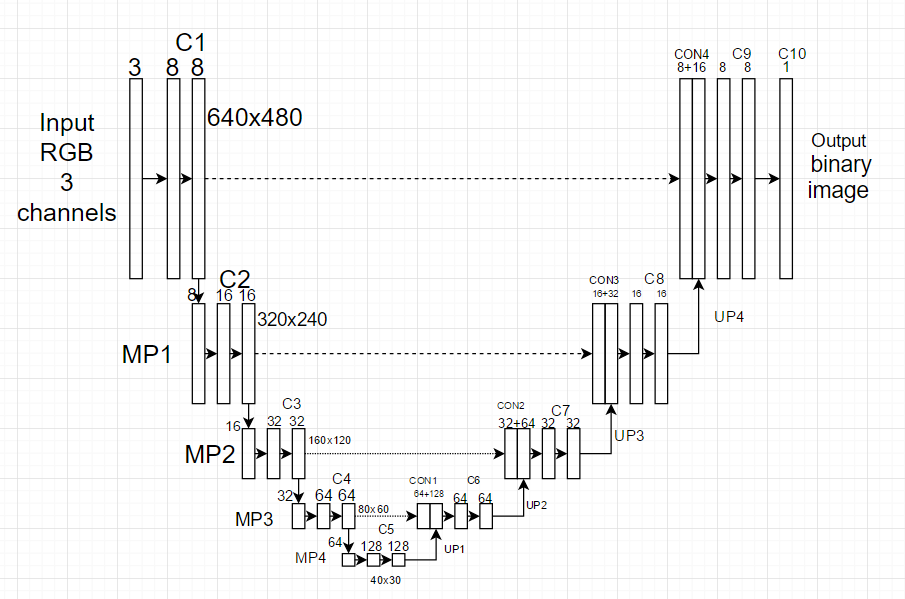
**5** **Training a Neural network for image segmentation**

The motivation for using a segmentation neural network is for vehicle detection followed by tracking using a Kalman Filter. An autonomous vehicle must be capable of being aware of it’s environments and the agents around it to be able to make decisions keepings it’s own integrity and avoiding accidents with other cars.

The U-net [deep learning architecture](https://chatbotslife.com/small-u-net-for-vehicle-detection-9eec216f9fd6#.y5gl6an4e) is one example of such a segmentation model. In segmentation based model, it makes pixelwise prediction to determine if a pixel belongs to an object or not.I t was first proposed and used for Biomedical Image Segmentation by Department of Computer Science from University of Freiburg [4]. The ideea for using this type of network is inspired after a post of an Udacity Self-driving-car nanodegree student [5].

The architecture for the network is an adapted version of the original in order to be able to run on a less performant computer:

* C1: 2 convolution layers with ReLU activation functions taking as input an image with size of 640x480x3 and produces an output of 640x480x8 feature maps
* MP1: 1 max pooling layer with a 2x2 kernel size and a 2x2 stride taking as input 640x480x8 feature maps and produces an output of 320x240x8 feature maps
* C2: 2 convolution layers with ReLU activation function taking as input the output of the max pooling layer (320x240x8 feature maps) and produces an output of 320x240x16 feature maps
* MP2: 1 max pooling layer with a 2x2 kernel size and a 2x2 stride taking as input the output of the last convolution (320x240x16) and produces an output of 160x120x16 feature maps
* C3: 2 convolution layers with ReLU activation function taking as input the output of the last max pooling layer (160x120x16 feature maps) and produces an output of 160x120x32 feature map
* MP3: 1 max pooling layer with a 2x2 kernel size and a 2x2 stride taking as input the output of the last convolution (160x120x32) and produces an output of 80x60x32 feature maps
* C4: 2 convolution layers with ReLU activation function taking as input the output of the last max pooling layer (80x60x32) and produces an output of 80x60x64 feature maps
* MP4: 1 max pooling layer with a 2x2 kernel size and a 2x2 stride taking as input the output of the last convolution (80x60x64) and produces an output of 40x30x64 feature maps
* C5: 2 convolution layers with ReLU activation function taking as input the output of the last max pooling layer (40x30x64) and produces an output of 40x30x128 feature maps
* UP1: 1 upsampling layer with a 2x2 kernel size taking as input the output of C5 (40x30x128) and produces an output of (80x60x128)
* CON1: concatenate the output of UP1 with C4 resulting in 80x60x(128+64) feature maps
* C6: 2 convolution layers with ReLU activation function taking as input the output of CON1 (80x60x192) and produces an output of 80x60x64 feature maps
* UP2: 1 upsampling layer with a 2x2 kernel size taking as input the output of C6 (80x60x64) and produces an output of 160x120x64
* CON2: concatenate the output of UP2 with C3 resulting in 160x120x(64+32) feature maps
* C7: 2 convolution layers with ReLU activation function taking as input the output of CON2 (160x120x96) and produces an output of 160x120x32 feature maps
* UP3: 1 upsampling layer with a 2x2 kernel size taking as input the output of C7 (160x120x32) and produces an output of 320x240x32 feature maps
* CON3: concatenate the output of UP3 with C2 resulting in 320x240x(32+16) feature maps
* C8: 2 convolution layers with ReLU activation function taking as input the output of CON3 (320x240x48) and produces an output of 320x240x16 feature maps
* UP4: 1 upsampling layer with a 2x2 kernel size taking as input the output of C8 (320x240x16) and produces an output of 640x480x16 feature maps
* CON4: concatenate the output of UP4 with C1 resulting in 640x480x(16+8) feature maps
* C9: 2 convolution layers with ReLU activation function taking as input the output of CON4 (640x480x24) and produces an output of 640x480x8 feature maps
* C10: 1 convolution layer with sigmoid activation function taking as input the output of C9 (640x480x8) and produces an output of 640x480x1 binary image



**Fig. 6.** Convolutional Neural Network for image segmentation

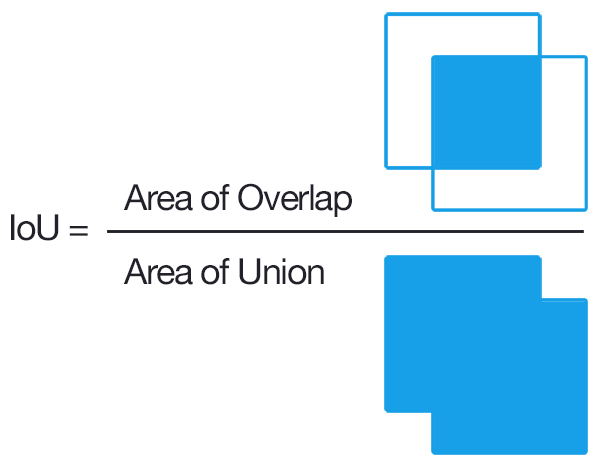
In order to collect data, a small application has been developed that first requested the user to create a video file which contained images of cars, then after the filming was stopped, the application requested the user to annotate images with points, specifically the upper left corner and the lower right corner of the bounding box of the car. The coordinates of the points are saved in a csv file. The user could only annotate one car at a time, and having two cars in an image is not recommended as the neural network could be “confused” by the annotating of one car and not annotating the other.

After the annotation is finished, the data is loaded and the images are labeled with the binary version of them consisting in the rectangles where the points were annotated.

To improve the accuracy of the model the dataset is augmented by scaling, flipping and modifying the brightness of the images.

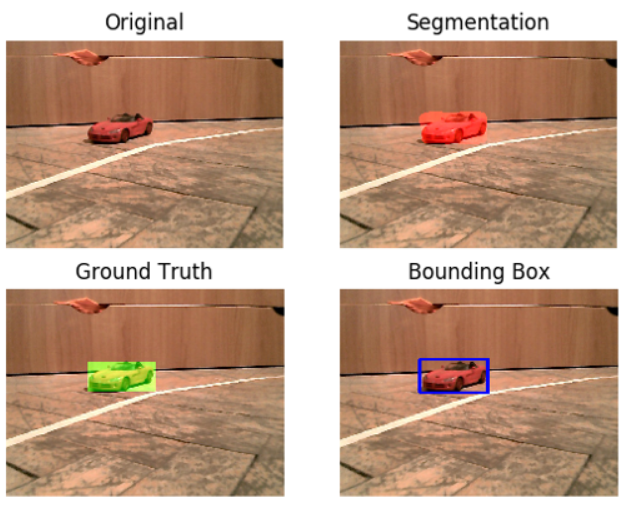
Training the data is done by selecting mini batches of size 6 of augmented images and the training usualy takes about 6-8 hours to produce a good segmentation. The parameters of the neural network occupy 37% of the video RAM (1.48 Gb) of a Nvidia Geforce 960M graphics card. The original network parameters used for biomedical segmentation required a graphics card with at leat 12 Gb of graphics RAM.

To compute the accuracy of a single image the following metric for bounding boxes is used. A higher IoU means a better accuracy. The negative of this value was used as the cost of the neural network.



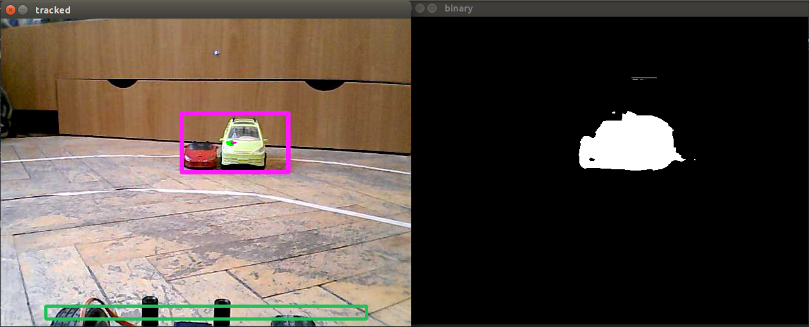
**Fig. 7.** Intersection Over Union [14]

Training the neural network for image segmentation lasted around eight hours. It was a slow process because the network occupies around 600Mb per image when training, because it has to store at every layer the activations. The mini-batch size for this neural network is 7 as the GPU memory is only 4Gb and that is the maximum number of images that the model can train at once. Also for every image in the training set it was augmented by scaling, modifying the brightness, and flipping the image in order to introduce more variation and make the neural network more robust.



**Fig. 8.** Segmentation neural network

The figure above illustrates the results after training the model. The neural network is provided with an image and a label that is simply described by the coordinate of the upper left corner and the lower right corner. After the segmentation is made by the neural network, a few heuristic algorithms were used to extract the objects that had a minimum area, and the those objects were bounded by a rectangle at the margins.



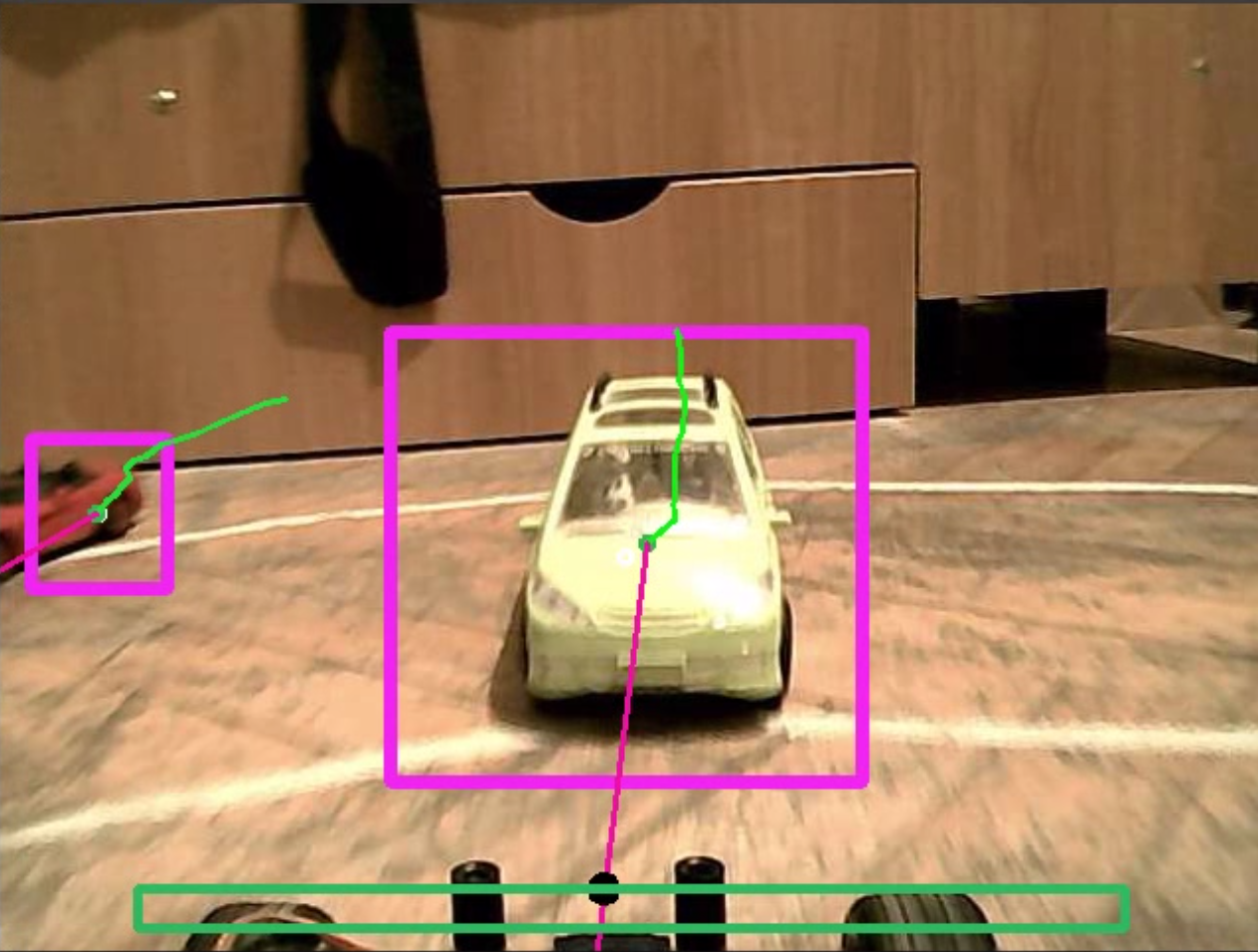
**Fig. 9.** Segmentation neural network disadvantage

The figure above illustrates the disadvantage of using this type of neural network. It can be clearly seen that it is not able to distinguish between very close objects, thus the final operation will result in framing the two objects as a single one. Another more powerful neural network and much faster that will be studied is Faster R-CNN[7] which is capable of recognizing more categories and takes less time to process.

**6** **Implementing the colision detection algorithm**

In this project the Kalman Filter was used to track the detected vehicles on a sequence of images received from an RGB camera. The steps in tracking are the following:

* detect the vehicles in image using the segmentation neural network and store the center points
* assign the detected points to tracked points using the Kuhn–Munkres algorithm
* if there are more detections than tracked points, the detections become tracked states
* if there are less detections than tracked points, the trackings without assignements get a penalty and if the penalty reaches a treshold it is removed from the tracking algorithm
* run the prediction step of the Kalman Filter
* run the update step of the Kalman Filter given the asociated detection
* extract the velocity vector from the kalman filter state variable
* use that velocity vector multiplied by a scalar in combination with the tracked point to see if the trajectory of the vehicle is towards the RC vehicle using line box intersection method used in computer games



**Fig. 10.** Object detection and trajectory estimation

This screenshot is taken at the end of the video. In this video the detected vehicles are stationary, and only the vehicle with the camera is moving. The trace in green represents the points where the car has been detected on the screen. The purple line starting from the center of the bounding box represents the estimated trajectory of the vehicle from the Kalman Filter and the green rectangle from the bottom represents the car body. If the purple line of the tracked vehicle intersects the green rectangle then a black dot is drawn at the intersection and the car stops as a warning has been issued.

**7** **Conclusions**

Designing an autonomous vehicle requires a lot of work. From creating the physical object for testing, designing the software architecture and the intelligence of the vehicle, optimizing the number of images streamed per second, to collecting training data, and waiting for the neural networks to finish training, all these operations are absolutely necessary and many other are needed to create a more advanced autonomous vehicle.

While the above results seem to be successful and the effort to complete this project was rewarded with a great knowledge gain, this is far from being an autonomous vehicle. The neural networks were not trained with enough data to be able to generalize in any condition and the tracking algorithm still needs improvements. A real autonomous vehicle is capable of 3d reconstruction of it’s environment, path planning and recognizing and understanding the significance of every object from it’s surroundings.

The constraints that were met in this project were the fact that a large training trail was not available as all of the training and testing were done in a small student room and the fact that the network quality depends on the quality of training data. The training data provided is small and unreliable, and many of the images are very similar to each other. Another constraint was the fact that the received image from the Raspberry Pi had a low quality because of the limitation of the network car incorporated in the board. Thus, the neural network prediction is not always reliable. Also because of poor local conectivity, the maximum number of frames sent per second was 20, and most of the time it varied between 15 and 18. Because of these small fps, the car reaction time is slowed and in order to drive safe, it was chosen a small constant speed for the rear wheels.

|  |  |  |
| --- | --- | --- |
| Machine | Detection Net | Steering Net |
| Intel i7 6700HQ (3,5 GHz 8 cores), Nvidia 960M (640 CUDA cores) | 0,063 s or 15,63 fps (CPU)  0,061 s or 16,25 fps (GPU) | 0,0062 s or 158,9 fps (CPU)  0,002 s or 491,11 fps (GPU) |
| Intel Pentium 4 (2,16 GHz 4 cores), Intel HD graphics | 1,721 s or 0,58 fps (CPU) | 0,026s or 38,09 fps (CPU) |
| Raspberry Pi ARM (1 GHz 4 cores), integrated GPU | 4,09 s or 0,24 fps (CPU) | 0,095s or 10,46 fps (CPU) |

**Table 1.** Performance benchmarks

The table above presents the performance of the neural networks tested on multiple machines. The best performance was achieved on a gaming laptop, and the Raspberry Pi achieved better than expected given the large number of parameters of the networks.

The inovative side of this project is the limit that we believe we pushed to the maximum. We efficiently used the limited resources of the development platforms to achieve real-time performance. The most demanding operations that had to be fine-tuned were to transfer the image from one device to another, and to make two neural networks run in paralel when both of them are trying to use the entire computing capability of the machine.

**References**

1. Dean A. Pomerleau: ALVINN, an autonomous land vehicle in a neural network, <http://repository.cmu.edu/cgi/viewcontent.cgi?article=2874&context=compsci>, (1989)
2. Mariusz Bojarski: End to End Learning for Self-Driving Cars, <https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf> (25 Apr 2016)
3. Roger R. Labbe Jr: Kalman and Bayesian Filters in Python, <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python> (2015)
4. Olaf Ronneberger, Philipp Fischer, and Thomas Brox: U-Net: Convolutional Networks for Biomedical Image Segmentation, <https://arxiv.org/pdf/1505.04597.pdf> (*2015)*
5. [Vivek Yadav: Towards a real-time vehicle detection: SSD multibox approach,](https://chatbotslife.com/@vivek.yadav) <https://chatbotslife.com/towards-a-real-time-vehicle-detection-ssd-multibox-approach-2519af2751c#.kors0eurl> (2017)
6. Vivek Yadav: Small U-Net for vehicle detection <https://github.com/vxy10/p5_VehicleDetection_Unet> (2017)
7. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (2016)

****Cristian TOMA**** is a member of [Department of Economic Informatics and Cybernetics - D.I.C.E/D.E.I.C](http://dice.ase.ro/) / Computer Science Department, Faculty of Cybernetics, Statistics and Economic Informatics - [C.S.I.E/C.S.E.I](http://csie.ase.ro/) at The Bucharest University of Economic Studies, Romania. He has graduated from the Faculty of Cybernetics, Statistics and Economic Informatics, Economic Informatics specialization, within Academy of Economic Studies Bucharest in 2003. He has graduated from the BRIE master program in 2005, with 24 months scholarship and 7 moths practical stage at University of Bremen, Germany and PhD stage in 2008. He was SA - Solution Architect & SDE - Software Development Engineer consultant at RADCOM company since 2003 to 2014 and now is SW Dev Engineer at Oracle / Sun since 2014.

**Alexandru IFTIMIE** is a student in the final year of Faculty of Cybernetics, Statistics and Economic Informatics at The Bucharest University of Economic Studies, Romania. His domain of interest and research include artificial intelligence, neural networks, computer vision and robotics. His major achievements include first prizes at student scientific competitions, hackathons, and innovative projects competitions.