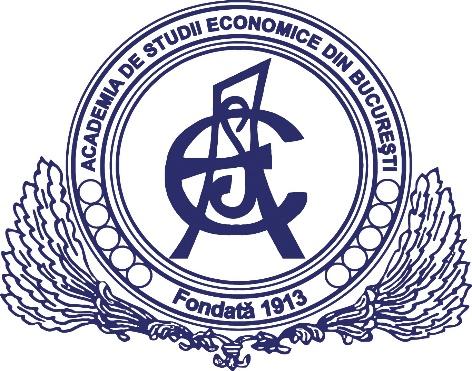
Bucharest University of Economic Studies

Faculty of Cybernetics, Statistics and Economic Informatics



IoT Embedded Solution using Artificial Intelligence and Image Processing for Autonomous Vehicles

Coordinating professor

Cristian TOMA

Graduate

Florentin Alexandru IFTIMIE

**Table of contents**

**1.** [**Introduction**](#_hd7n0l9r3kef) **2**

[**2. Objectives**](#_g9wi681utdkg) **3**

[**3. Infrastructure**](#_i21anw57xwtj) **3**

[**4. Designing the vehicle’s brain**](#_bmlye2nm6pec) **6**

[4.1 Knowledge requirements](#_5jicwv1o9qwk) 6

[4.2 Training a steering angle predictor](#_esr5n08q86k5) 8

[4.3 Collecting data for steering predictor](#_dgbwyyiohgp8) 9

[4.4 Neural network for image segmentation](#_qjf6ij4110z5) 10

[4.5 Collecting data for image segmentation](#_g9widlpa2khr) 12

[4.6 Kalman filter](#_pf9946ie94y1) 14

[**5. Conclusions**](#_e4f25o2yf93m) **15**

[**6. References**](#_1ykt6m6k0tsl) **15**

# 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.

A major breakthrough has been made in 2007 when a group of researchers from Standford University combined robotics, probabilistics and computer vision techniques in order to create Stanley, the self-driving car capable of driving 142 miles through the Mojave desert that also was also awarded the Darpa Grand Challenge 2006 [2].

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 witho no other robotic techniques, which was feeded with images from 3 cameras and succesefully drove on the road without human intervention [3].

The advantages of using autonomous vehicles in society are numerous such as

* reducing the number of car fatalities from 1.3 million each year [5] as the car is capable of predicting the trajectory of other vehicles and making decisions faster than any human
* reducing the fuel consumption using controlled acceleration and braking would make a great difference given the fact that worldwide there is a total of 93,500,000 barrels of petroleum consumed each day [4], thus reducing the pollution
* reducing the traffic jams and wasted time by communicating with other autonomous vehicles through the internet and using complex fleet controlling algorithms.

Some of these functionalities have already been implemented by car manufacturers such as Audi, Mercedes Benz, and Tesla Motors, the last one being the only company that delivered to general public a car with autopilot.

# 2. Objectives

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 [7] with Kuhn–Munkres algorithm for object tracking, and using that information for predicting if another car will collide with the driver’s car.
* Creating a client-server application, the server running on Raspberry Pi awaiting for “worker clinets” and the client running on a laptop and processes the received images and responding with commands

# 3. Infrastructure

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

* Raspberry Pi model 3 B+
* custom built motor driver
* servo motor that replaced the original less capable left-center-right motor
* PWM module
* RGB camera
* one 9.6 NiMh battery and one 5V battery
* laptop with nvidia GPU

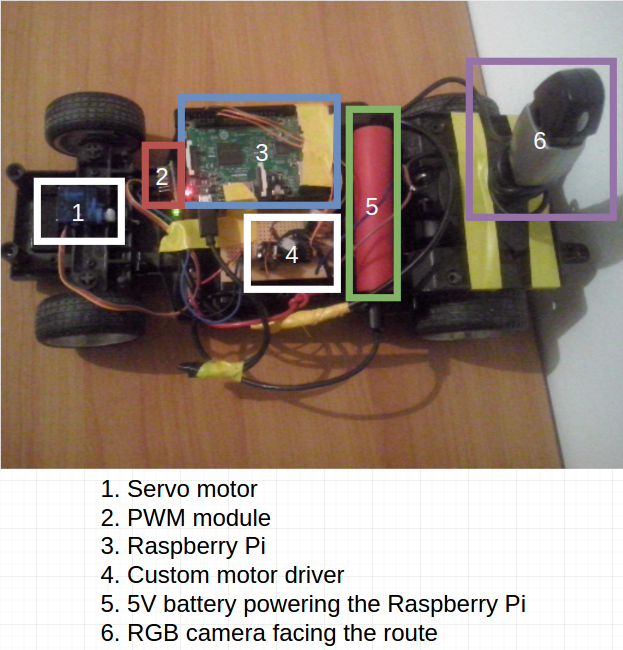


Figure 1. Hardware design

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 clients” in order to distribute tasks such as steering angle prediction and vehicle tracking.

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
* after processing the commands, the server grabs an image 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

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 and even a delay of 1 microsecond 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.

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

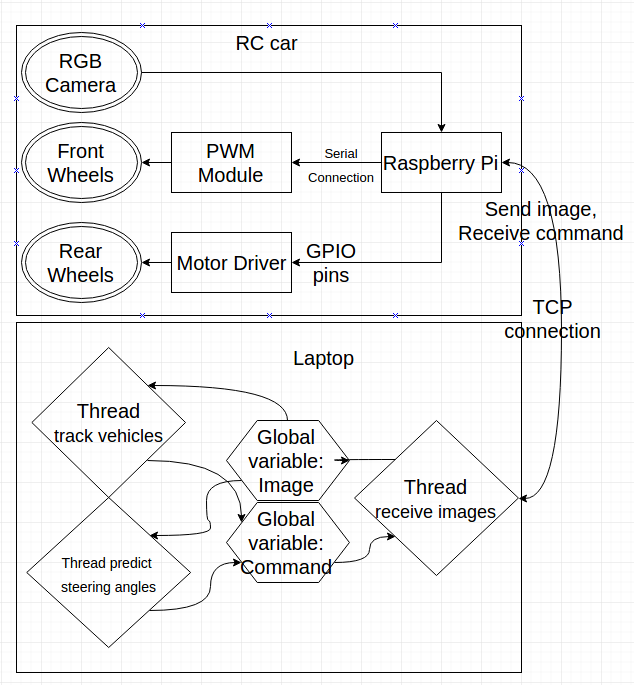


Figure 2. Software architecture

# 4. Designing the vehicle’s brain

## 4.1 Knowledge requirements

In the next sections I will enumerate the operations that will be used to construct the neural network.

An activation function is inspired from biological neural networks which represent the rate of action potential firing in cell given a set of input signals. In computational networks the activation function that I used is the Rectified Linear Unit or ReLU which has the following formula and graph:

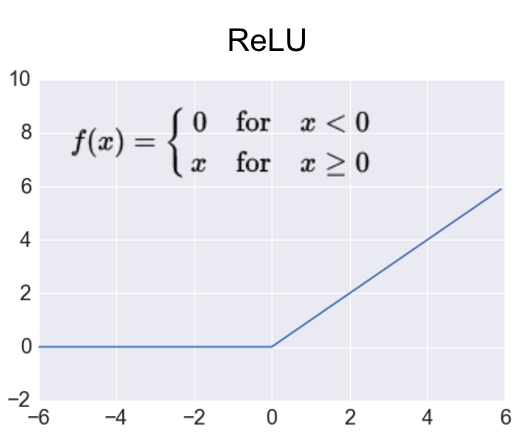


Figure 3. ReLU [source: http://adilmoujahid.com/images/activation.png]

Other activation functions that are mainly used are:

* hyperbolic tangent function or TanH

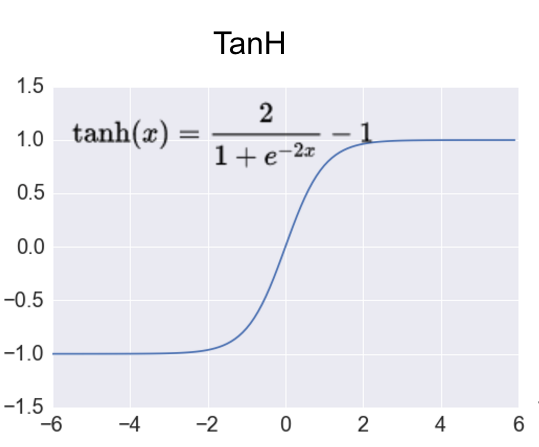


Figure 4. TanH [source: <http://adilmoujahid.com/images/activation.png>]

* sigmoid function

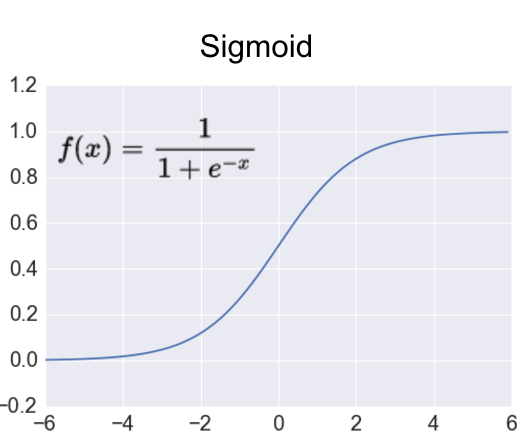


Figure 5. Sigmoid [source: http://adilmoujahid.com/images/activation.png]

In computer vision an image can be viewed as a function I(x,y) where x and y are the coordinates on a 2d space and the output is the pixel value intensity in that point which can be a single value or a tuple if the image has mutiple channels.

A convolution in mathematics is an operation on two functions in order to produce a third one. A convolution of f and g is written as f \* g. Thus, in computer vision, a convolution looks as follows:

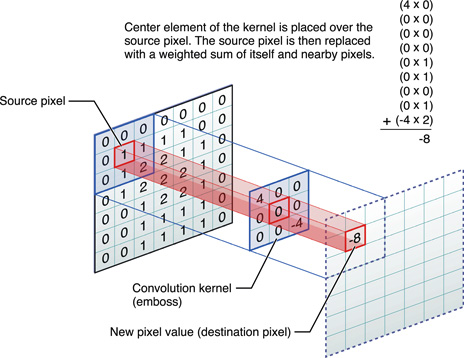


Figure 6. Convolution [source:<https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/Art/kernel_convolution.jpg>]

A max pooling operation is used for reducing the size of the image in order to reduce the number of parameters.

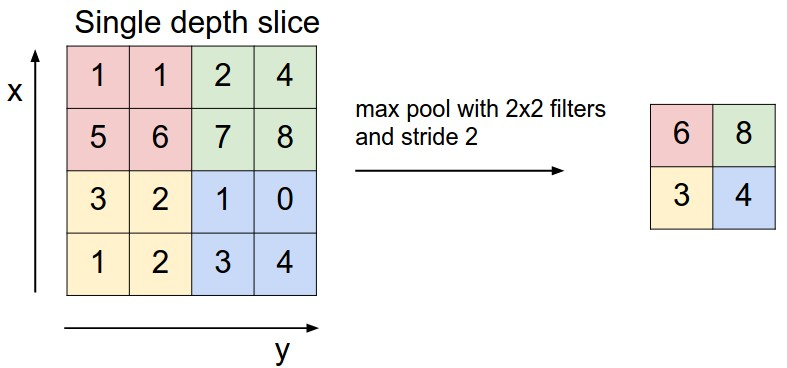


Figure 7. Max Pooling [source: <http://cs231n.github.io/convolutional-networks> ]

The upsamplimg takes a low resolution input and produces a higher resolution output. The ideea is that each pixel from the low resolution input is copied to the higher resolution image weighted by a scalar from the kernel that is also learnable. The upsampling can be considered the inverse of max pooling.

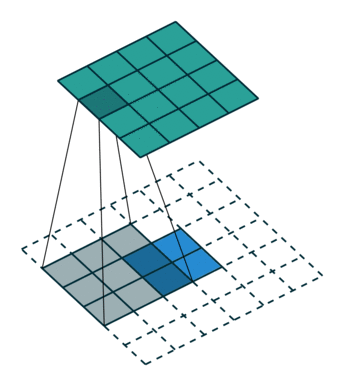


Figure 8. Upsampling[source: <https://i.stack.imgur.com/YyCu2.gif>]

## 4.2 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.

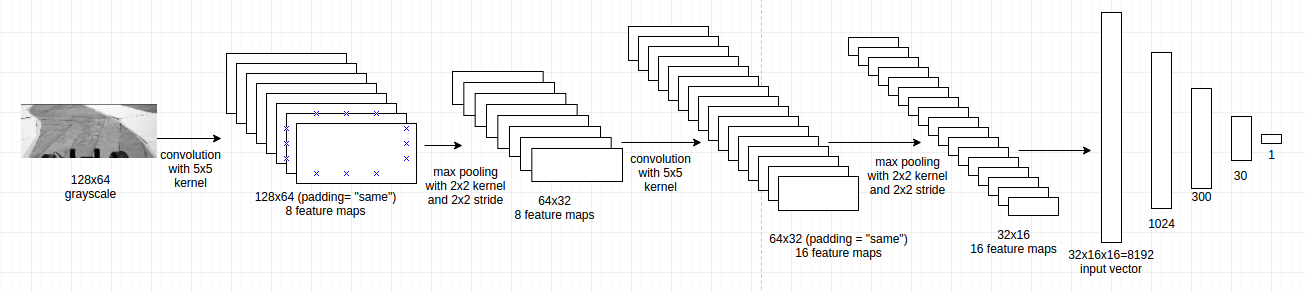


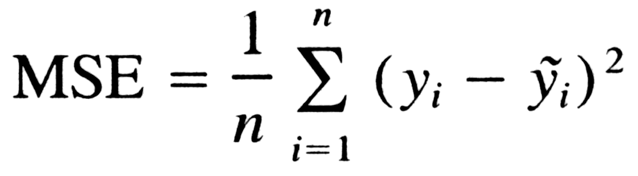
Figure 9. Convolutional neural netowork for steering angle prediction

## 4.3 Collecting data for steering predictor

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 TXT 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:



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.



Figure 10. Steering angle prediction

## 4.4 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 [6]. The ideea for using this type of network is inspired after a post of an Udacity Self-driving-car nanodegree student [8].

The architecture for the network is :

* 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

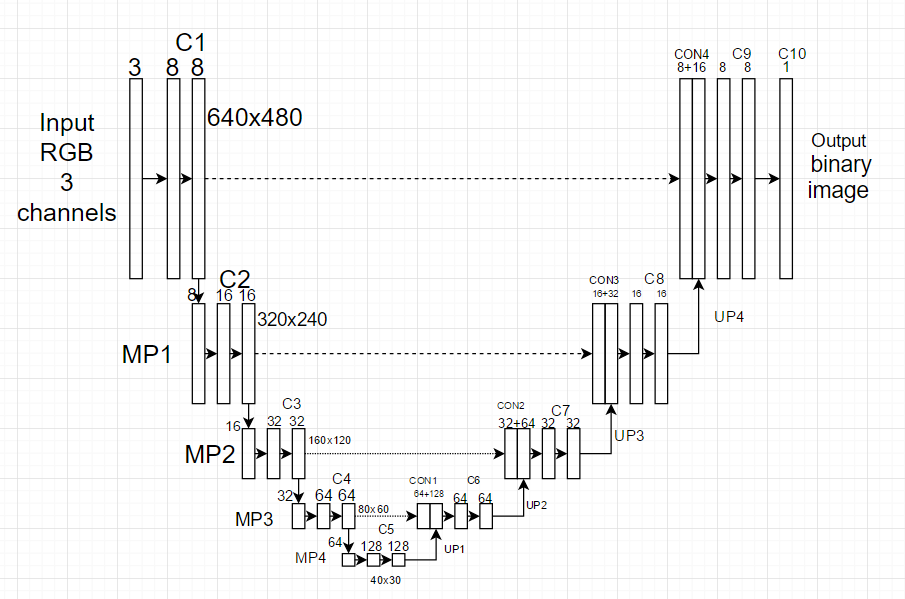


Figure 11. Convolutional neural netowork for image segmentation

## 4.5 Collecting data 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.

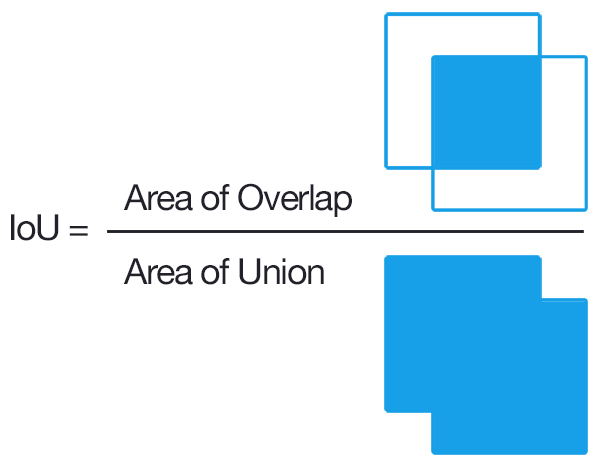


Figure 12. Intersection Over Union

[source: <https://github.com/vxy10/p5_VehicleDetection_Unet>]

The following images are the results after training the model:

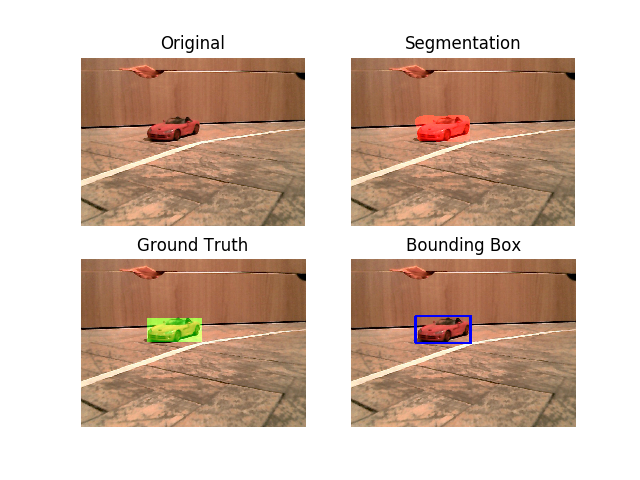


Figure 13. Image segmentation

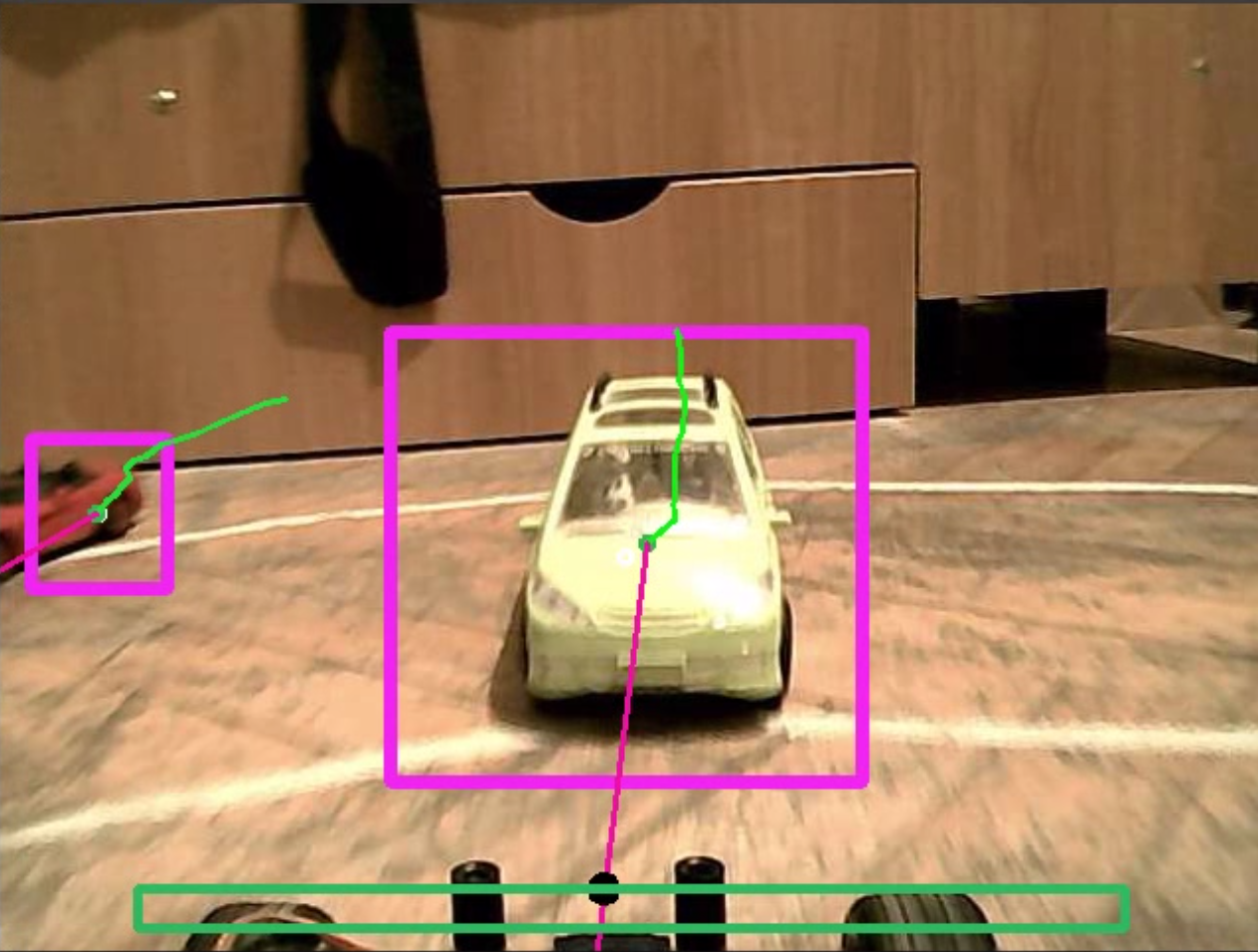
## 4.6 Kalman filter

A kalman filter is a mathematical model that is used to observe the state of a variable in time given a set of measurements that are very noisy, and to predict the next state which is less noisy using hidden velocity vector varible . It has two stages: a prediction step and an update step.

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

The result of this pipeline can be better understood in the following image:



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.

# 5. 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.

Still, for the personal development, this project was a very instructive one with many benefits, and this is the only thing that I trying to achieve.

# 6. 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] Sebastian Thrun: Stanley: The Robot that Won the DARPA Grand Challenge, <http://isl.ecst.csuchico.edu/DOCS/darpa2005/DARPA%202005%20Stanley.pdf> (2006)

[3] 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)

[4] Petroleum consumption: <http://world.bymap.org/OilConsumption.html> (2015,2014)

[5] Annual Global Road Crash Statistics: <http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics>

[6]Olaf Ronneberger, Philipp Fischer, and Thomas Brox: U-Net: Convolutional Networks for Biomedical Image Segmentation, <https://arxiv.org/pdf/1505.04597.pdf> (*18 May 2015)*

[7]Roger R. Labbe Jr: Kalman and Bayesian Filters in Python, <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python> (2015)

[8] [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)