Using Artificial Intelligence to create a low cost self-driving car

Objectives/Goals

The purpose of this project is the creation of an autonomous car which should be able to drive automatically without any driver in the urban areas. Road traffic injuries caused an estimated **2.5 million deaths** worldwide in the year 2004 and more than **50 million injured**. A study using British and American crash reports as data, found that 87% of crashes were due solely to driver factors. A self-driving car can be very safe and useful for the entire mankind. In order to realize this, several concurrent software applications process data using Artificial Intelligence to recognize and propose a path which an intelligent car should follow.

Two years ago, Google has managed to create the world's first autonomous car. The **Google autonomous car problem** is caused by using **a very expensive** 3D radar **($ 75,000**), with a very high resolution. The 3D radar is used to recognize the environment and create a high resolution 3D map. **My solution** is a **minimal 3D radar** that would **only cost $4000** and 3 special cameras mounted to recognize from images the marker lines, borders and real time position of the car instead of the 3D radar.

Additionally a driverless car can reduce the distance between the cars, lowering the degree of road loadings, reducing the number of traffic jams, avoiding human errors, and allowing disabled people (even blind people) to drive long distances. A computer as a driver will never make any error. It will be able to calculate very precisely in order to reduce the distance between cars. By using this method, traffic jams will be shorter and parking places will be freer. A lot of fuel used by cars will be dramatically saved.

Methods/Materials

The necessary power is provided by three multi-core laptops that use Artificial Intelligence for the purpose of recognizing traffic signs, traffic lanes, traffic car fingerprints. They process the data from a 3D radar, using GPS coordinates and particle filters, localize the car on Google Maps, the management of a common database with traffic signs, magnetic sensors, acceleration sensors, a distributed software, a supervisory system and the software which drives the stepper motor to turn the steering wheel (acceleration and braking).

Results

Most of the project's components have been done. The software is able to recognize the traffic signs and register them in a common database using Google Maps and GPS. The GPS software component records the signs and direction of travel from that area. Each car participating in the traffic using this software will register the new signs detected and will modify the confidence degree of recognition for other users. Another software component is able to recognize the demarcation lines between lanes. It uses three cameras to calculate exactly or using probabilities the position of the car on the road, where the roadsides are and to propose a new direction even in the absence of traffic signs for the next seconds. Another part of the software uses Artificial Intelligence to detect other car fingerprints from webcam images.

The algorithms were implemented parallel and distributed. I developed a management software system based on semaphores that allows data processing and supervision from 3 different computers with multiple cores.

This project contains also a home-made LIDAR – a 3D radar and a software using OpenGL to create a 3D environment in which the car navigates. By using it, the car will take the decision of avoiding obstacles. The 3D radar helps the entire software system to increase the confidence of decision.

Summary Statement/Conclusions/Discussion

My project tests a new effective and inexpensive way to help the people, even disabled, drive safer and faster. Practically, tens of millions of lives can be saved if people will be using self-driving cars.

Key words:
Parallel calculations, distributed calculations, synchronous, critical sections, Neural Networks, HSV filters, ranger finder (LIDAR), 3D Radar, 3D map, sockets, TCP/IP, Google Maps, Artificial Intelligence

Theoretical considerations

"Keep your eyes open. Closing your eyes means traffic accident."- Jarod Kintz

The percent of information anyone gets from his eyes when driving is 95%

A human driver is able to recognize only traffic signs, traffic lanes and obstacles. The latest statistics concerning vehicle accidents reveal that 87% of traffic crashes are due solely to driver factors. Having this in mind, I am trying to create a completely self-driving car.

Motivation

Tens of millions of people have lost their lives or have become disabled worldwide in the last 10 years as a consequence of traffic accidents. The purpose of this project is to create a safe self-driving car that could help millions of people each year. Almost all of the traffic accidents are caused by human mistakes. *Unfortunately, according to statistics, in the next 10 years, the number of lives lost each year will likely double.*

1. According to the ***World Health Organization***, road traffic injuries caused an estimated **1.26 million deaths** worldwide in the year 2000. (Source: World Health Organization, 2000)

2. Worldwide it was estimated in 2004 that **2.5 million** people were killed in traffic accidents(4.7% of all deaths) (World report on road traffic injury prevention, WHO, 2004)
3. **50 million** were injured in motor vehicle collisions. (World report on road traffic injury prevention, WHO, 2004)
4. $\frac{2.5\ mill\ ion}{365} = 6{,}849\ people\ dies\ per\ day\ in\ car\ accidents.$
5. $\frac{50\ million}{365} = 136{,}986\ people\ injured\ per\ day\ in\ vehicle\ collisions.$
6. **"In 2004 almost 140,000 people were injured each day worldwide in traffic accidents"**

Worldwide statistics

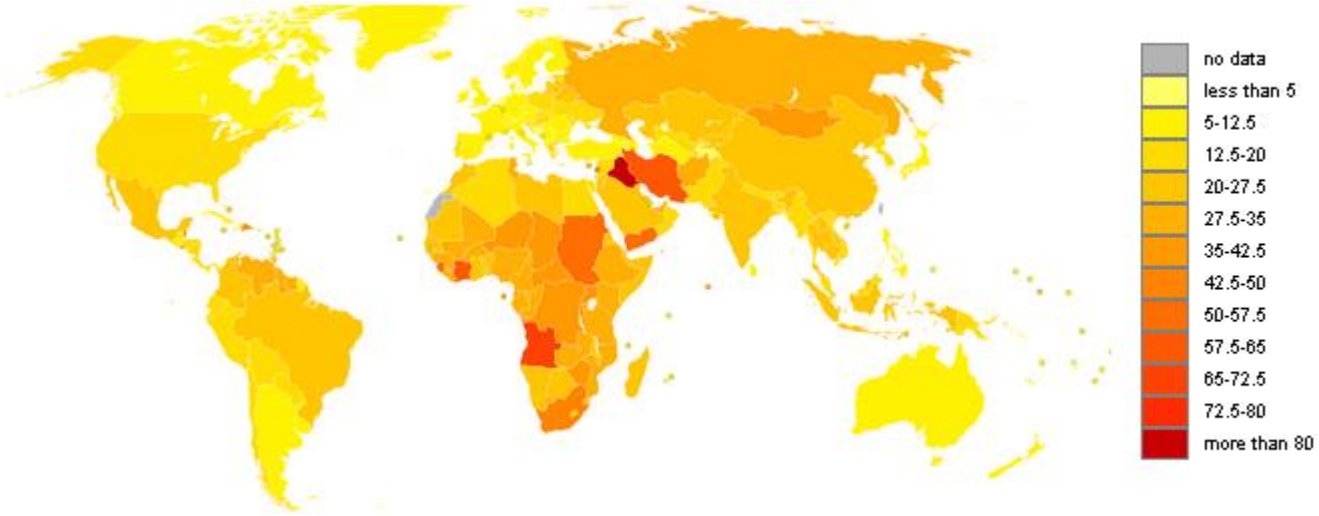
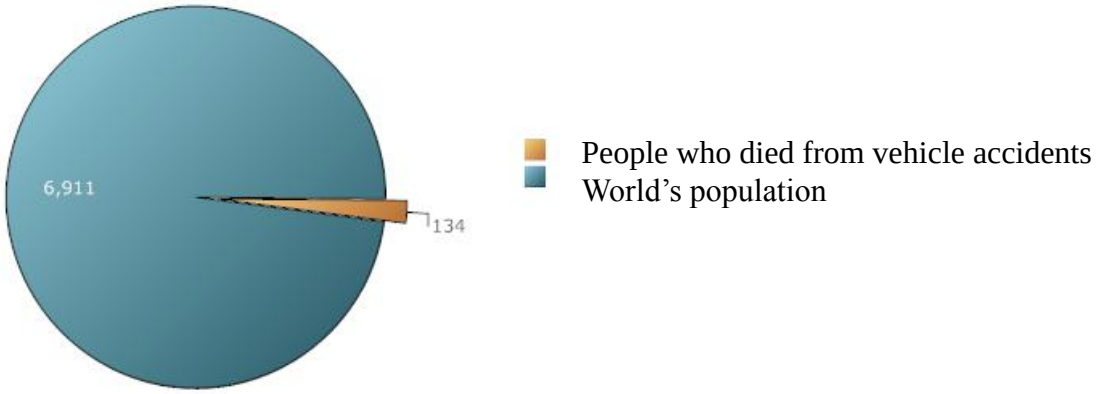


Figure 5. Road fatalities per capita (fatalities per 100,000 inhabitants per year). It was estimated in 2004 that **2.5 million** people were **killed** in traffic accidents/car crashes and **50 million** more injured in motor vehicle collisions worldwide.

People who died from car accidents in the last 15 years.

Using simple math we can estimate the number of lives lost in the last 15 years. If in 2000 the number of people who lost their lives was 1.26 million and in 2004 it was 2.5 million we can estimate the means for the last 15 years as 2.253 (because this number grows each year). So multiplied with 15 years is about ~ 33.8 million people. In the following diagram can be seen a pie chart that represents the raport of the people who died from vehicle accidents to the World's population.



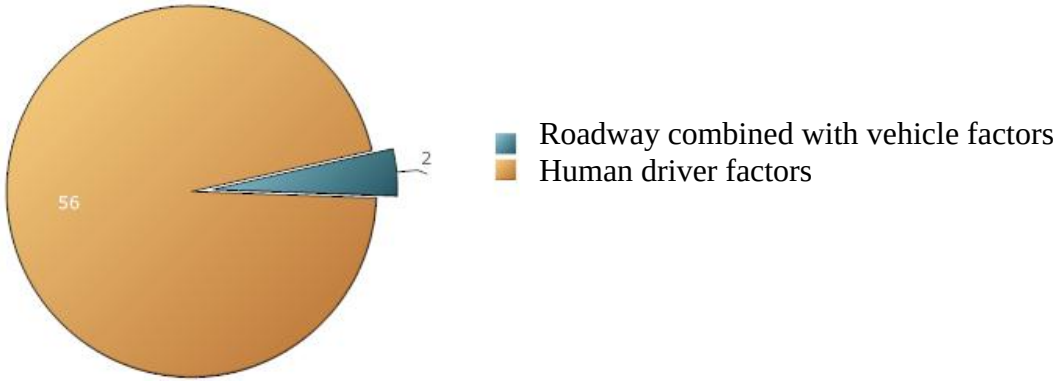
Causes of the traffic accidents

A 2005 study by K. Rumar, using British and American crash reports as data, found that 87% of crashes were due solely to driver factors, 7% to combined roadway and 6% to combined vehicle factors.

Human factors

Human factors in vehicle collisions include all factors related to drivers and other road users that may contribute to a collision. Examples include driver's behavior, visual and auditory acuity, decision-making ability, and reaction speed. According to some statistics 54% of all traffic accidents are caused by alcohol.

In the following pie chart can be seen the percent of human mistakes from all car accidents. I remind you that 87% of crashes were due solely to driver's factors.



The statistics are an estimative based on the data from ***World Health Organization.***

Theoretically a car without a driver in the future should be much safer. Human reaction speed is higher than 200 ms, while the computing power of the newest computers allows traffic calculations even to 10 ms. Furthermore, an autonomous car will be able to calculate the shortest distance and to communicate with other cars using wireless technology, thus avoiding traffic jams or accidents.

Autonomous car's diagram

In the following diagram you can see my autonomous car design. The car contains two cameras designed so as to be able to spot the lanes from left and right. Another camera is placed in the driver's position in order to be able to spot the traffic lanes and traffic signs as if viewed by a normal driver. The 3D radar, called also Rangefinder or LIDAR, is attached to the car to create a 3D model, including a real-time model of the environment. The 3D information, traffic lanes and the traffic signs will be used by the supervisor software in order to calculate the collisions and the car's path.

The Artificial Intelligence calculation is provided by three different laptops. These laptops are synchronized using supervisor software called Master Software. The supervisor software component will quantify all of the processed data by all other components and it will be able to turn the steering wheels in order to maintain the car on the street/ road.

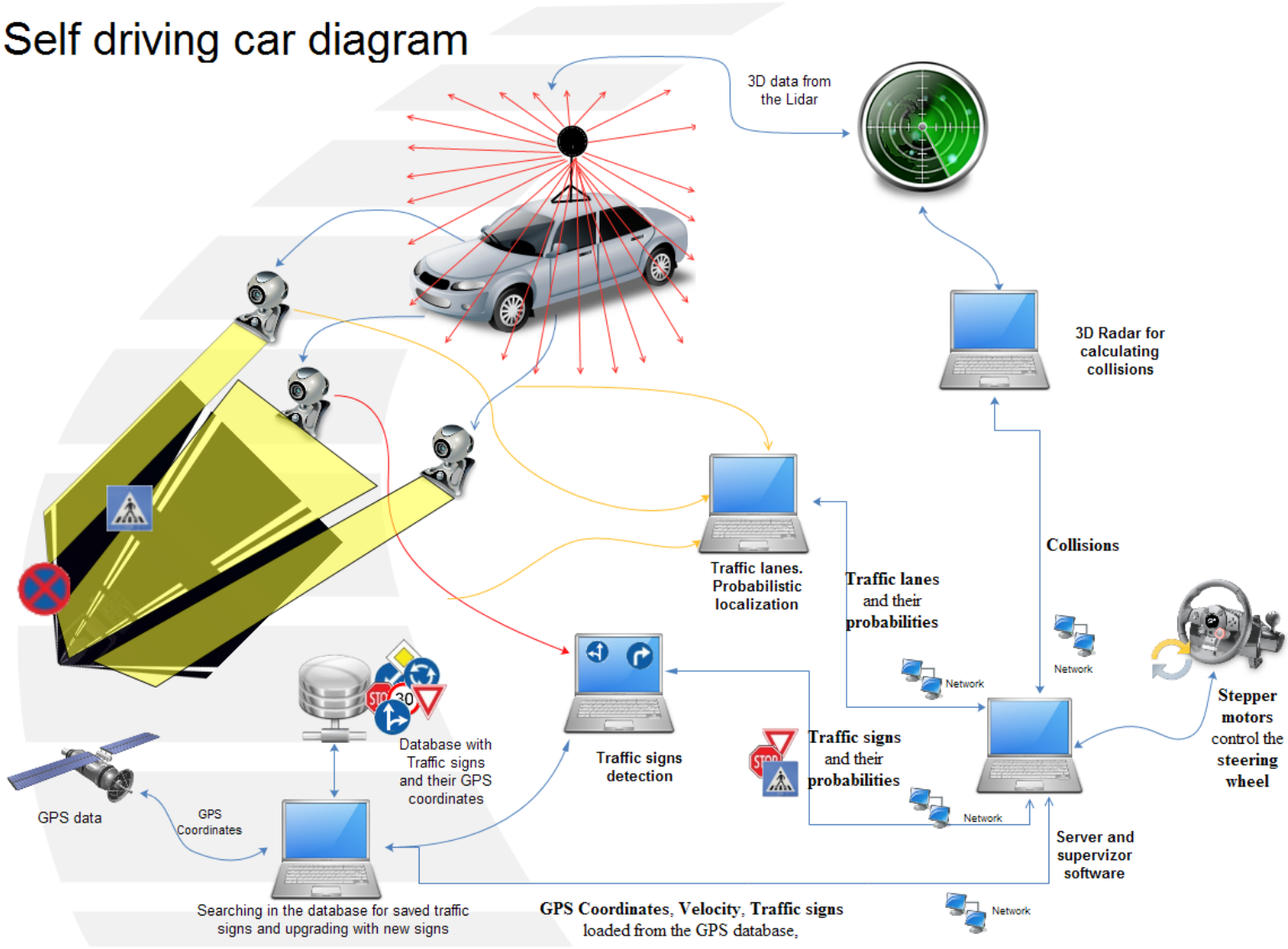


Image no. 1.1. The self-driving car diagram – how the concurrent software parts were built and designed. The supervisor software receives data from all other components via network and controls the car's steering wheel using stepper motors.

Traffic lanes recognition

The software is able to recognize the demarcation traffic lanes on the streets. The algorithm is a unique method that recognizes traffic lanes using three different webcams. Two images were taken from the left and right side of the car. Another image was taken from a webcam placed in the position of the driver to spot the demarcation lanes from the top view. The software is able to compute the distance between the lanes and the probability how close are the car's wheels to traffic lanes.

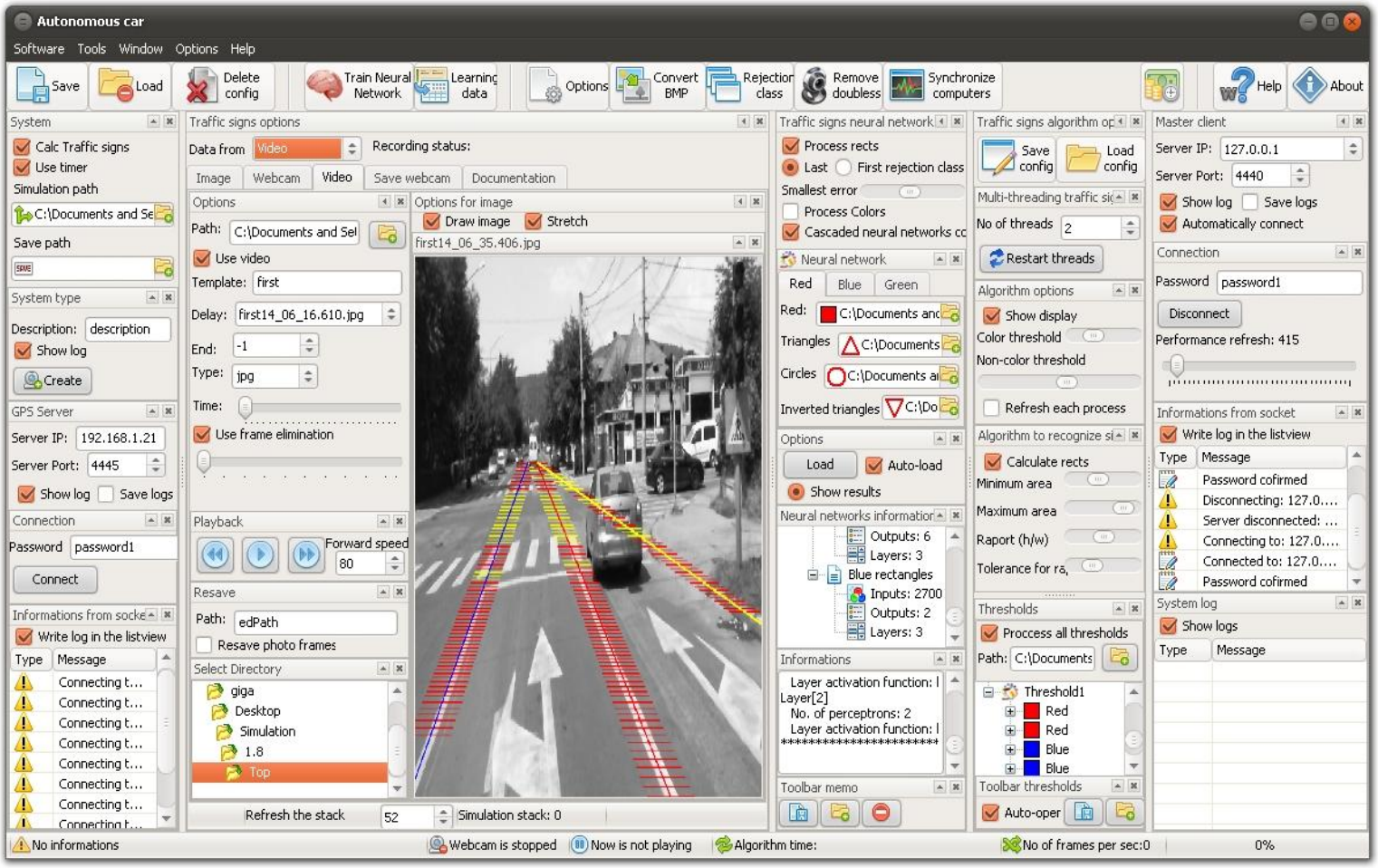
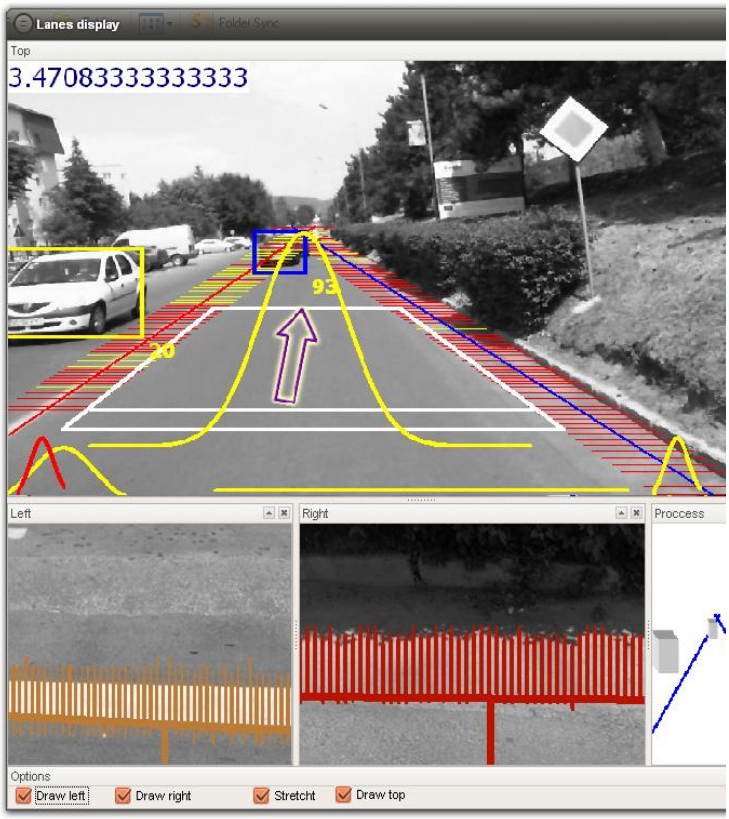
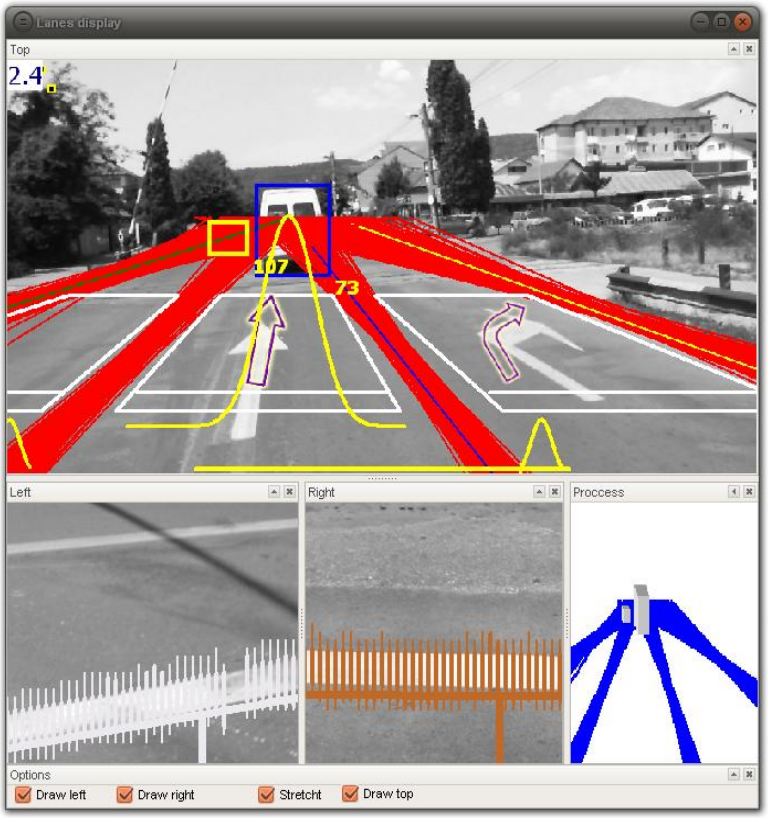


Image no. 1.4. Traffic lanes software screenshot.The software which is able to recognize the traffic lanes from images and correct the direction path of the car in order to keep the car on the street.


Lanes display
Top
2.4
145
99
136
Left
Right
Proccess
Options
Draw left
Draw right
Stretcht
Draw top


Lanes display
Top
3.47083333333333
93
20
Left
Right
Proccess
Options
Draw left
Draw right
Stretcht
Draw top


Lanes display
Top
2.4
107
73
Left
Right
Proccess
Options
Draw left
Draw right
Stretcht
Draw top

Traffic lane tracking

A particle filter, also known as Monte Carlo method is used in statistics as a sophisticated mathematical estimation techniquebased on a long time simulation (called also tracking). The main idea is that the traffic lanes are continuous and they don't disappear in just a few seconds and appear again. "Particle filters are usually used to estimate Bayesian models in which the latent variables are connected in a Markov chain - similar to a hidden Markov model (HMM) where the state space of the latent variables is continuous rather than discrete, and not sufficiently restricted to make exact inference tractable (unlike a linear dynamical system, where the state space of the latent variables is restricted to Gaussian distributions and hence exact inference can be done efficiently using a Kalman filter)." (Doucet A.)

Lanes are detected using a particle filter. A rule-based particle filter system handles the tracking of multiple lanes by deleting invalid lanes and creating new lanes.[30] For each lane, a single lane tracker is used with minor adaptations. The lane is described by some parameters:

$\boldsymbol{\theta}$ the tilt angle between the car and the traffic lane.

$\boldsymbol{\gamma}$the length of the traffic lane

$\boldsymbol{\delta}$the probability of existence for the current traffic lane

The tracking of a single lane is based on the use of a particle filter algorithm. Each particle represents one sample(one possible traffic lane). The evaluation function determines the probability using the measurements determined by this traffic lane sample(particle). Each particle represents a particular parameter set of the lane model(different parameters) described above.

The overall probability (also known as the weight of the particle) is calculated based on the detection using the scan line algorithm for detection.

The transition(State equation)

$$X_1 \sim \mu(x1) \text{and} f(X_k|X_{k-1}\,, (for\ k > 1)$$

Observation equation

$$g(y_k|X_k)$$

Posterior distribution (we want to find the posterior probability)

$$p(X_{1:n}|Y_{1:n})$$

Using Bayes:

$$p(X_{1:n}|Y_{1:n}) = \frac{\boldsymbol{p(X_{1:n}, Y_{1:n})}}{\boldsymbol{p(Y_{1:n})}}$$

The joint probability can be written as:

$$p(X_{1:n}, Y_{1:n}) = p(X_{1:n-1}, Y_{1:n-1})\, f(X_n|X_{n-1})\, g(Y_n|X_n)$$

So

$$p(X_{1:n}|Y_{1:n}) = p(X_{1:n-1}, Y_{1:n-1}) \frac{f(X_n|X_{n-1})\, g(Y_n|X_n)}{p(Y_n|Y_{1:n-1})}$$

Update step

$$\boldsymbol{p(X_n|Y_{1:n}) = \frac{g(Y_n|X_n)\, p(X_n|Y_{1:n-1})}{p(Y_n|Y_{1:n-1})}}$$

Prediction step:

$$\boldsymbol{p}(\boldsymbol{X_n}|Y_{1:n-1}) = \int f(X_n|X_{n-1})p(X_{n-1}|Y_{1:n-1})dx_{n-1}$$

The elementary steps of the Monte Carlo Algorithm(the particle filter)

- Sampling from a set of probability densities $\{\pi_n(X_{1:n})\}$
- Sampling N independent random variables $X_{1:n}^i$ from each probability distribution and estimating the distribution using the next formulas:

$$\varepsilon_n(X_{1:n}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{X_{1:n}^i}(X_{1:n})$$

$\delta_{X_{1:n}^i}(X_{1:n})$is the difference at each sample.

Sometimes these distributions are intractable in closed-form and is specially these case in a non-linear and a non-Gaussian models. The particle filter is based on the calculation of an importance density called weight of the model density. In the following formula I am using $W_n(X_{1:n})$ to describe the weights of the particles and $q_n(1_{1:n})$ to describe the importance density. $Z_n$is a normalization term.

$$\varepsilon_n(X_{1:n}) = \frac{Y_n(X_{1:n})}{Z_n}$$

$$\varepsilon_n(X_{1:n}) = \frac{W_n(X_{1:n})\, q_n(1_{1:n})}{Z_n}$$

$$\varepsilon_n(X_{1:n}) = \sum_{i=1}^{N} W_n^i \delta_{X_{1:n}^i}(X_{1:n}) \text{and} W_n^i = \frac{W_n(X_{1:n}^i)}{\sum_{j=1}^{N} W_n\left(X_{1:n}^j\right)}$$

So the selection importance distribution

$$q_n(X_{1:n}) = q_{n-1}(X_{1:n-1})q_n(X_n|X_{1:n-1})$$

$q_1(x_1)$is the first step - a simple sample from the original/initial distribution
$q_k(X_k|X_{1:k\ 1}^i)$ – for other steps, it picks from the conditional probabilities
The variance of these estimations usually increases with n, so it is necessary a further refinement. In order to do resampling, a method to reduce variance is required- it will be based on sampling again from the newly created approximation distributions. All of the samples are associated with numbers to estimate the distribution.

Summary of the algorithm:

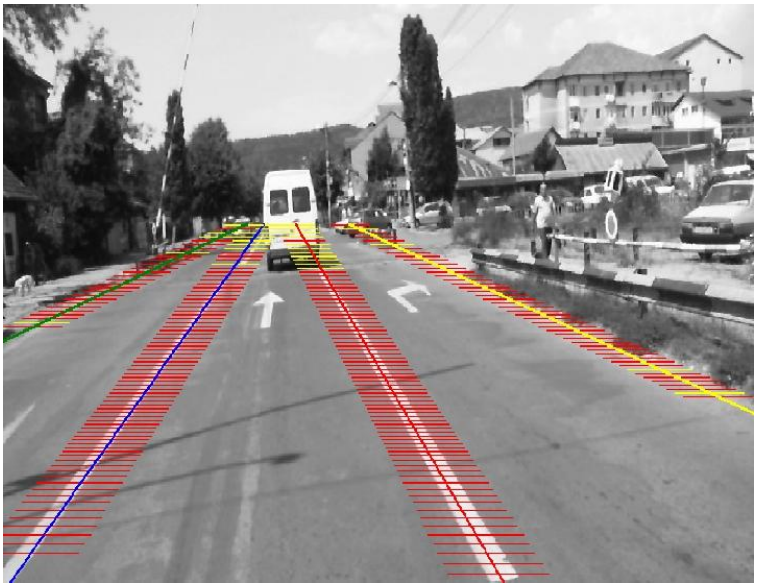
For the first time (n=1)

- Sample $X_1^i \sim q_1(X_1)$
- Compute the probabilities(weights)$W_1(X_1^i)$ and W_1^i
- Resample $\{W_n^i, X_1^i\}$ to obtain N particles $\{\frac{1}{N}, \overline{X}_1^i\}$

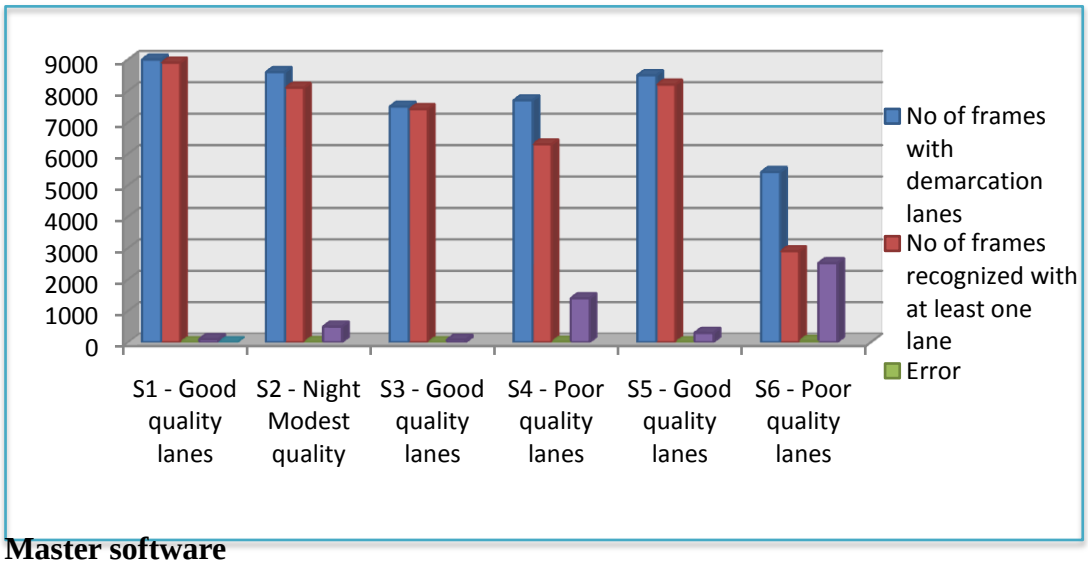
For all other times($n \geq 2$)

- $X_{1:n}^i \sim q_n\left(X_n \middle| \overline{X}_{1:n-1}^i\right)$ and set $X_{1:n}^i \leftarrow (\overline{X}_{1:n-1}^i, X_n^i)$
- Compute weights $\alpha_n(X_{i:n}^i)$ and W_n^i
- If is necessary, resampling the $\{W_n^i, X_{1:n}^i\}$ in order to obtain $\{\frac{1}{N}, \overline{X}_{1:n}^i\}$

Print screens of the traffic lanes detection using particle filters.

Statistics about the traffic lanes detection usingKalman filters with different simulation data.



In order to obtain these statistics I have developed a special statistical tool.

Master software

The master software it is the software which collects the data from all other software components and is able to calculate the decisions of the car path in order to maintain the car on the road. Furthermore this supervisor software is able also to synchronize the software components and is used to start and to control all other software components (Traffic signs, Traffic lanes recognition, GPS Software, 3D Radar software) from distance.

The communication was made using my own Protocol and uses a local wireless connection. In a summary statement this master software is able to collect the entire data from all other software parts and to make the decisions of the car's path.

The master software creates a Model Based Design graph to illustrate the current status of the other software components. Using the drag & drop technique the user is able to control the software components from distance.

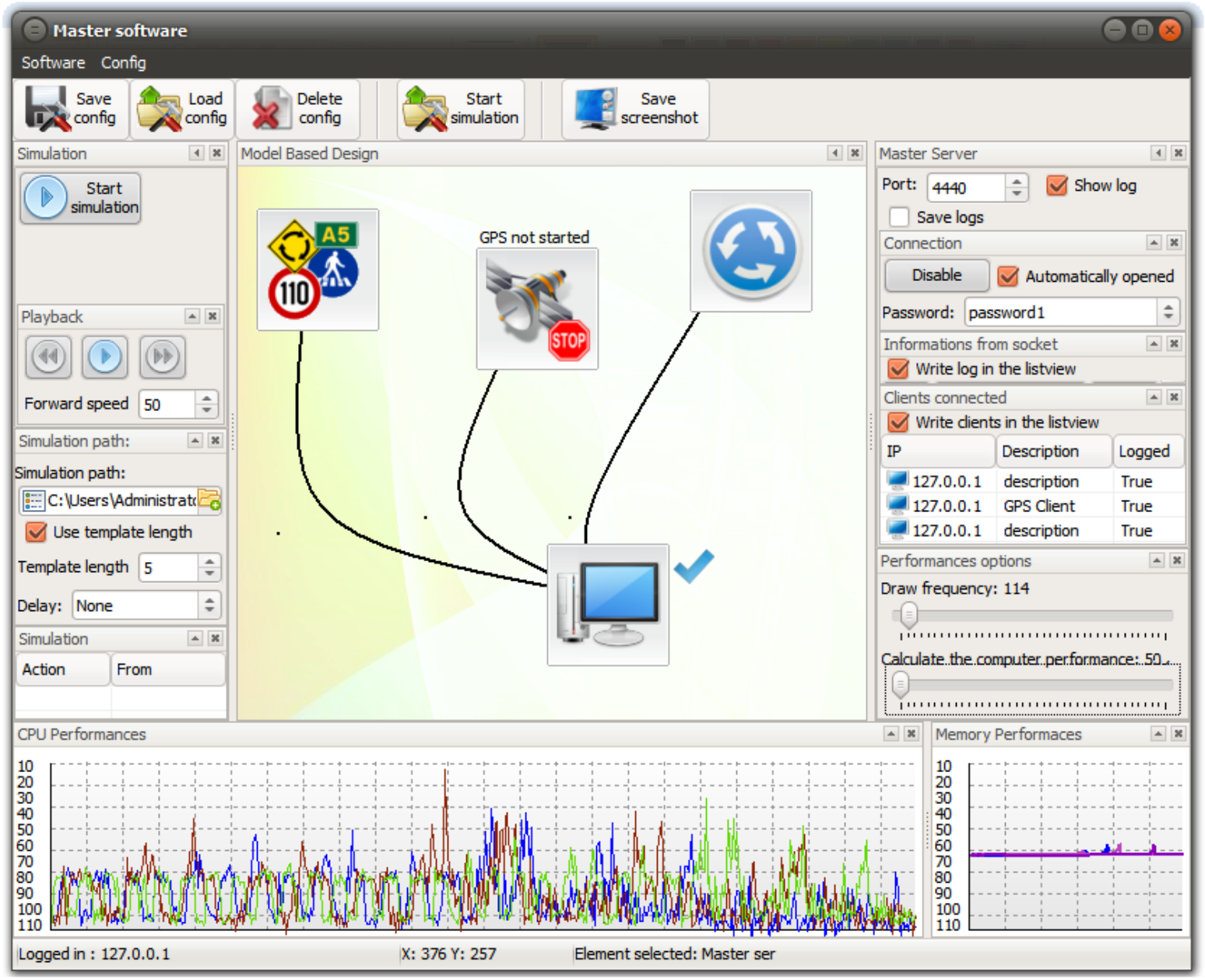


Image no. 1.2. The supervisor software screenshot – In this picture you can see the master software. The master software is able to communicate and control all other software components. In this image you can see the Model Based Design Graph.

The software is able to control all software components, to start the offline simulation and the recording data(webcams images, synchronization time and the GPS coordinates) . The supervisor software can open predefined configuration from XML data such as configuration files, in order to configure itself. The user can control visually the entire car using the Model Based Design graph and it's Popup. In the following diagram anyone can see the MBD graph and the options for controlling the GPS software component. The software also records logs on how the things are going on for an easy debugging.

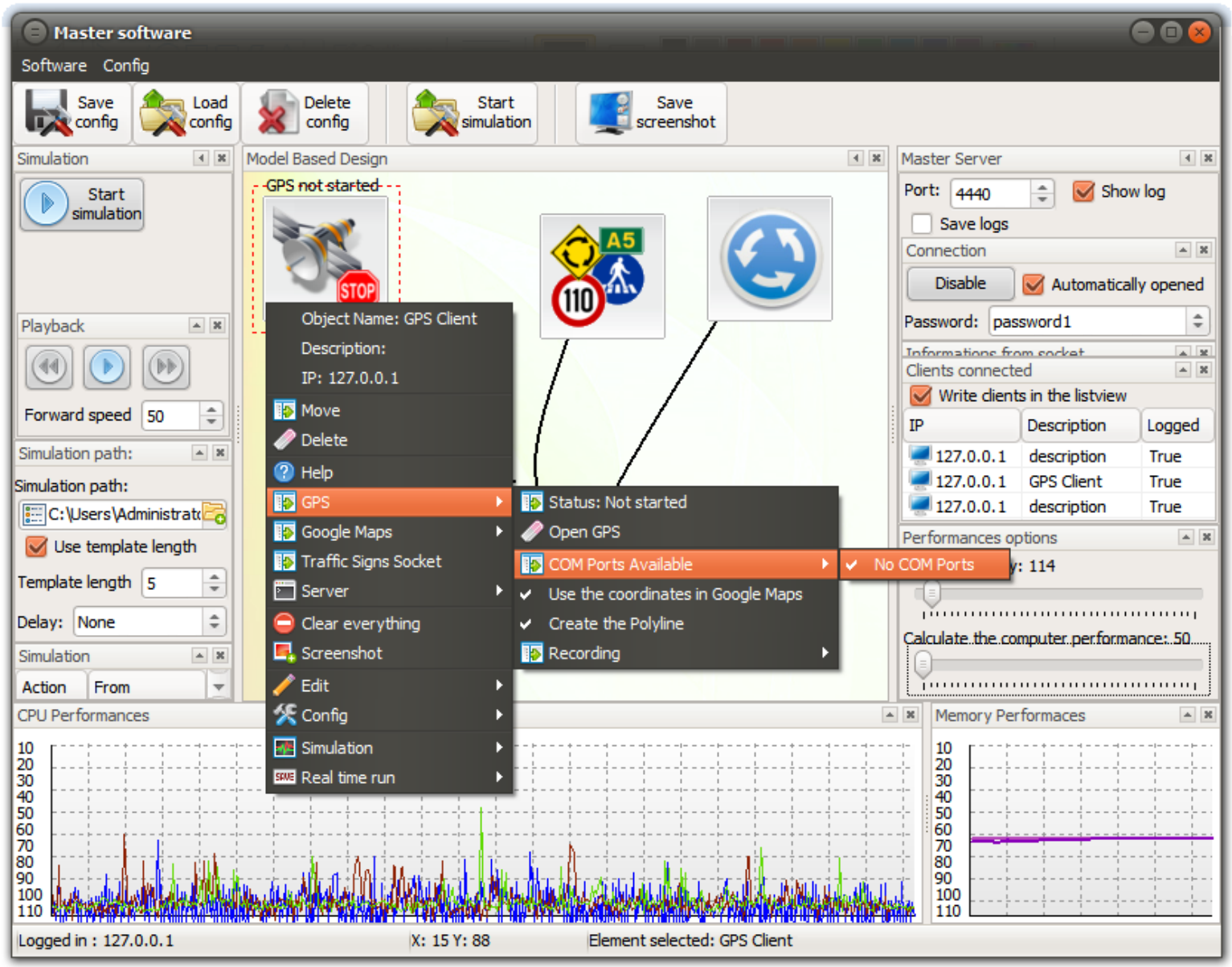


Image no. 1.3. The supervisor software screenshot – In this picture you can see the master software. The master software is able to communicate and control all other software components from the project. In this image you can see a popup menu created in order to configure and control the GPS software component.

3D map software

Another software component was intended to create a 3D map (using the OpenGL technology) then use it to debug and understand the 3D data from the Range Finder (LIDAR). The 3D data is read using the serial protocol RS232 and processed by this software. The software is also able to connect to the supervisor software and to receive further instructions from it. As the 3D radar hasn't been finished yet, in the following image you can see a random 3D map using a 3D random data. In the picture, a dot is a 3D obstacle situated at a distance X. The distance of the obstacle is described by the color of the dot.

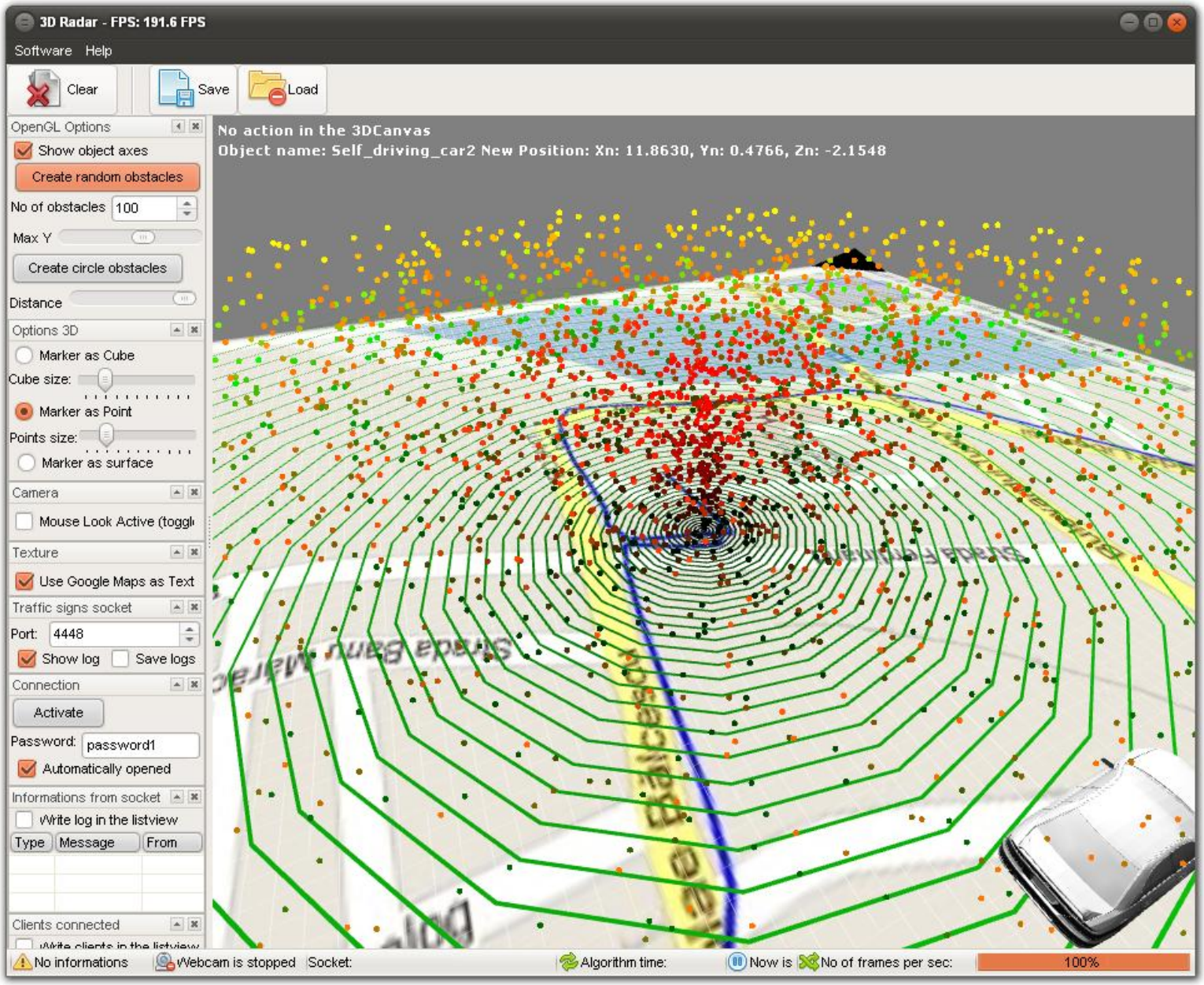


Image 1.4 In the above picture you can see the software interface and the world created by it. The colors of the objects are chosen in order to describe the Pythagorean distance between the car and the object. The data were generated randomly.

The software is able to save the data and load it afterwards for a further offline simulation. The depth of the map is only 80 meters, so the radius, r = 80 m. There 180 points on the circumference, so there 180 samples from a photo detector for each turn. So the distance between two consecutive points is:

$$Distance\ between\ two\ consecutive\ points = \frac{2\pi r}{180} = \frac{160\pi}{180} = 2.79\ meters$$

There is also an option for micro steps(1/10 from a step)

$$Distance\ between\ two\ consecutive\ points\ using\ miro\ steps = \frac{2\pi r}{1800} = \frac{160\pi}{1800} = 0.27\ meters$$

The precision is on 10 bits, so $2^{10} = 1024$
One step is $\frac{80}{1024} = 0.078125\ meters = 7.8125\ cm$
The error between two consecutive steps: $0.078125\ meters = 7.8125\ cm$

These calculations will be for each photodiode.

Since the car does not stand still but moving, I have to calculate the new position each 1/20 seconds. The vehicle's position is calculated with real time data obtained from an accelerometer which is not shown in the electrical scheme. The data must be recalculated though a correction so that the movement of the car should not interfere the objects' distances.

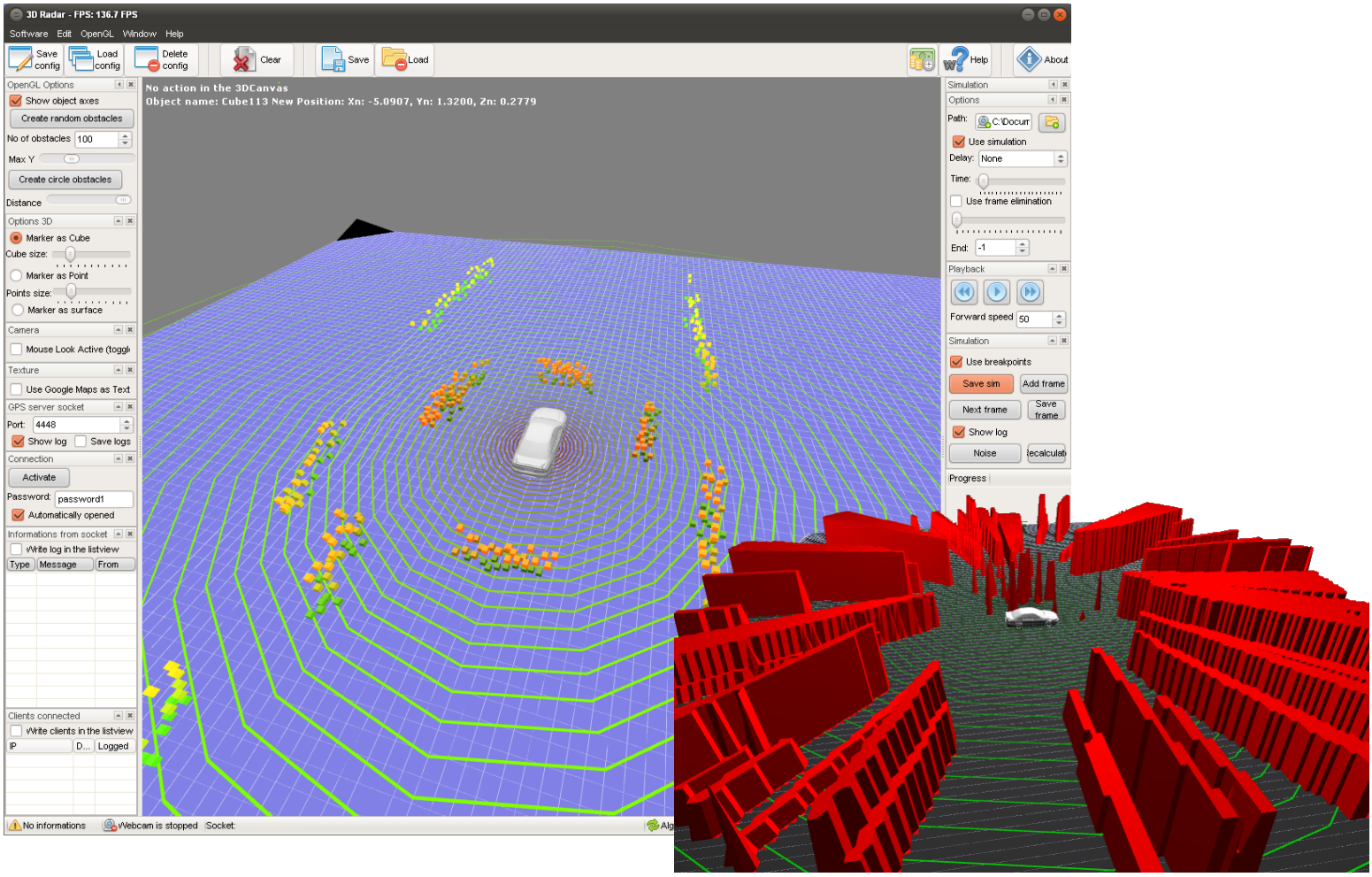


Image 1.4 In the above picture you can see the software interface and the world created by the software using the OpenGL. The colors of the objects are chosen in order to describe the Pythagorean distance between the car and the object. The data were generated randomly.

GPS Software

Sometimes the detection of the traffic signs cannot be done accurately because obstacles such as trees or people may cover the traffic signs. To avoid this and to increase the traffic signs recognition, all autonomous cars collaboratively will try to create a common database with all traffic signs and their GPS position. Using this common database, every time an autonomous car drives, the GPS Software component will update the database with new traffic signs and the supervisor software will receive the storage traffic signs from that GPS position.

The traffic signs from the common database have a very important property namely: **feedback.** The feedback allows me to create a probability of certainty for the traffic sign stored in the common traffic signs database.

The communication between the supervisor software, traffic signs and the GPS software were made using protocols built by myself.

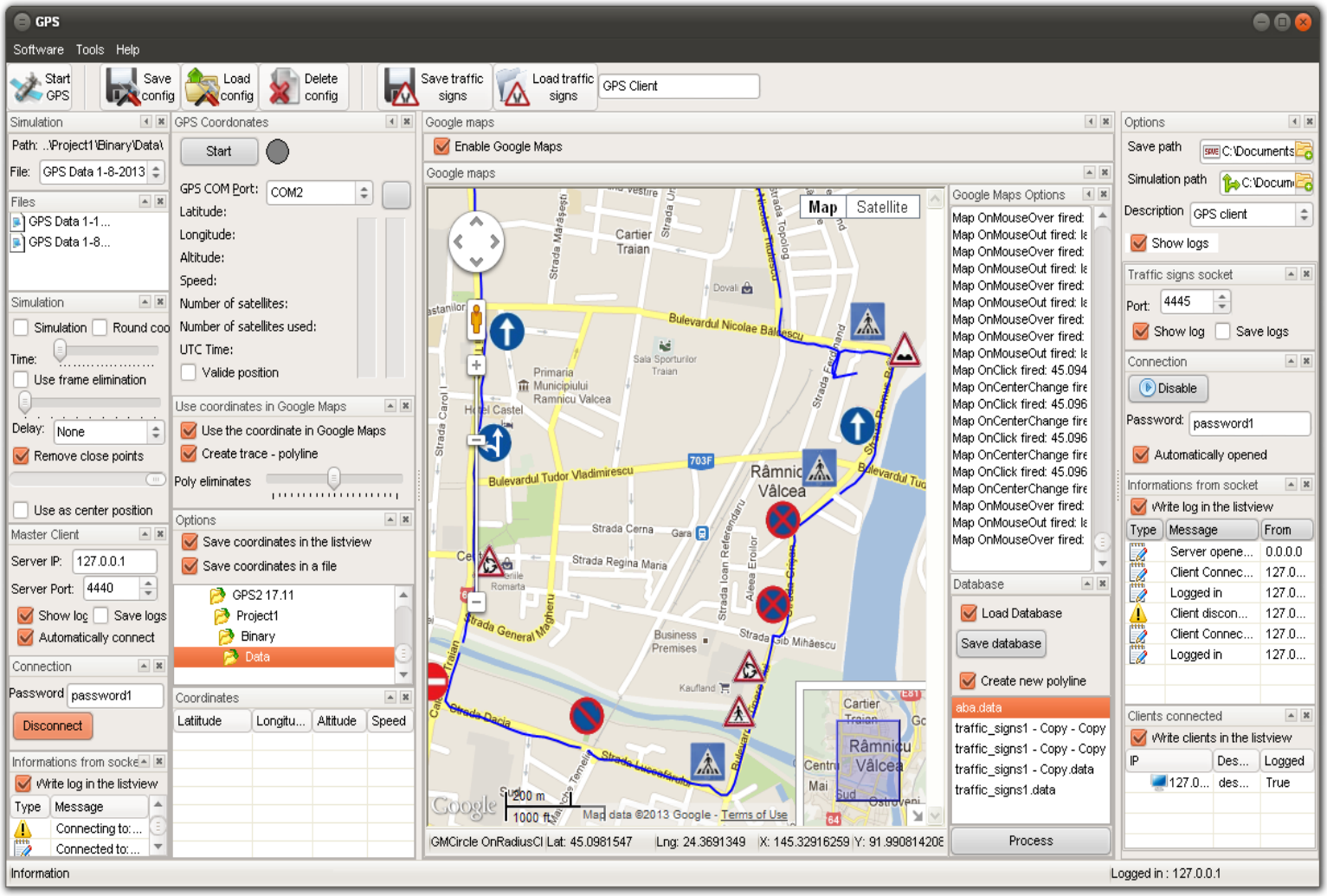
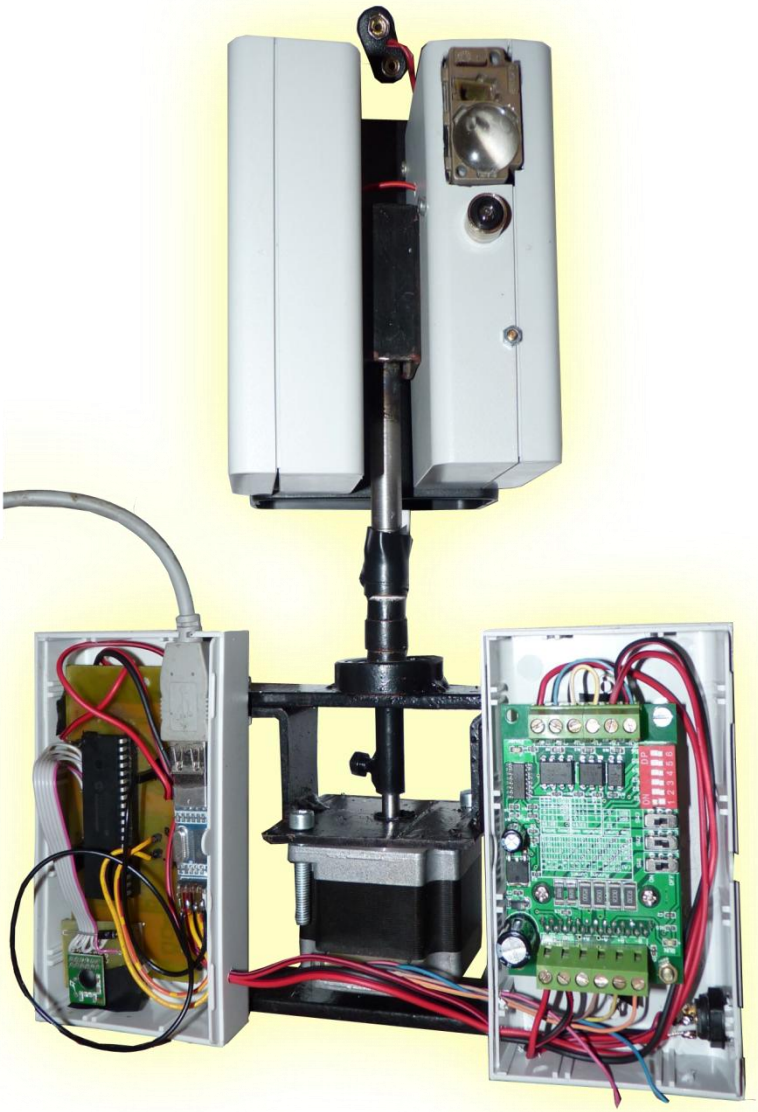
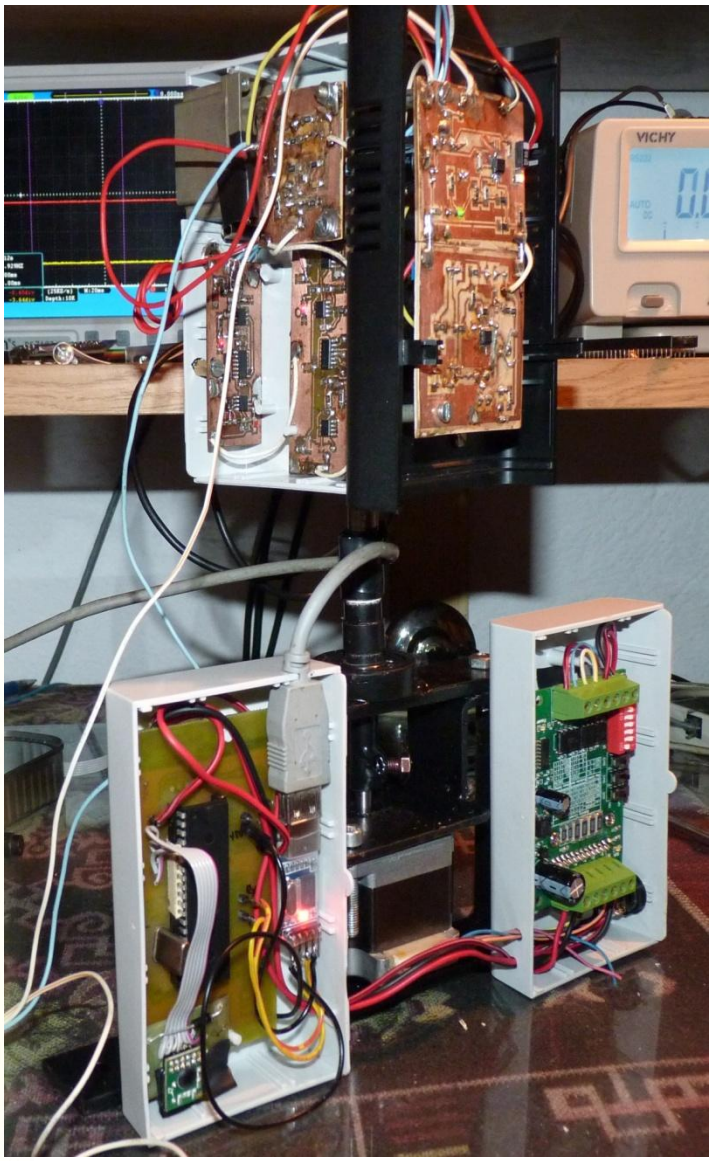


Image no. 1.5. In this screenshot you can see the GPS Software component. In the image, the blue line is the path of the car. The traffic signs were the signs recognized in real time by the software and from now on stored in the common database.

The 3D Radar

This project presents a hardware version of a LIDAR – a 3D radar and a software for creating a 3D environment in which the car navigates. By using it, the car will take the decision to avoid obstacles. The 3D radar helps the entire software system to increase the confidence of decision.

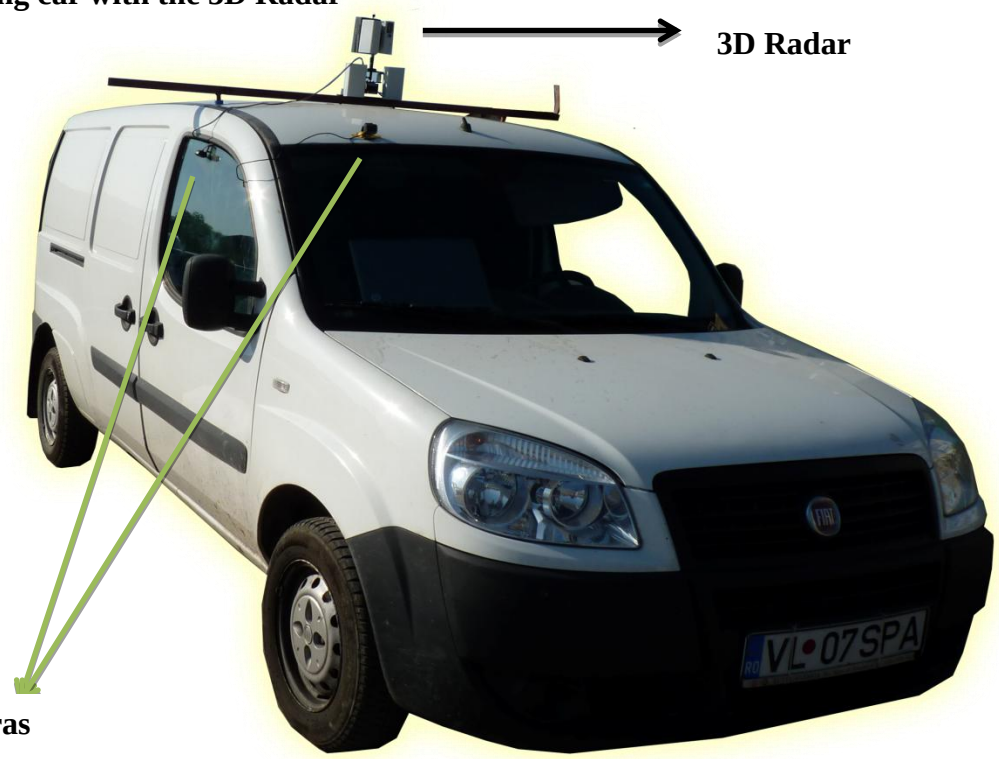




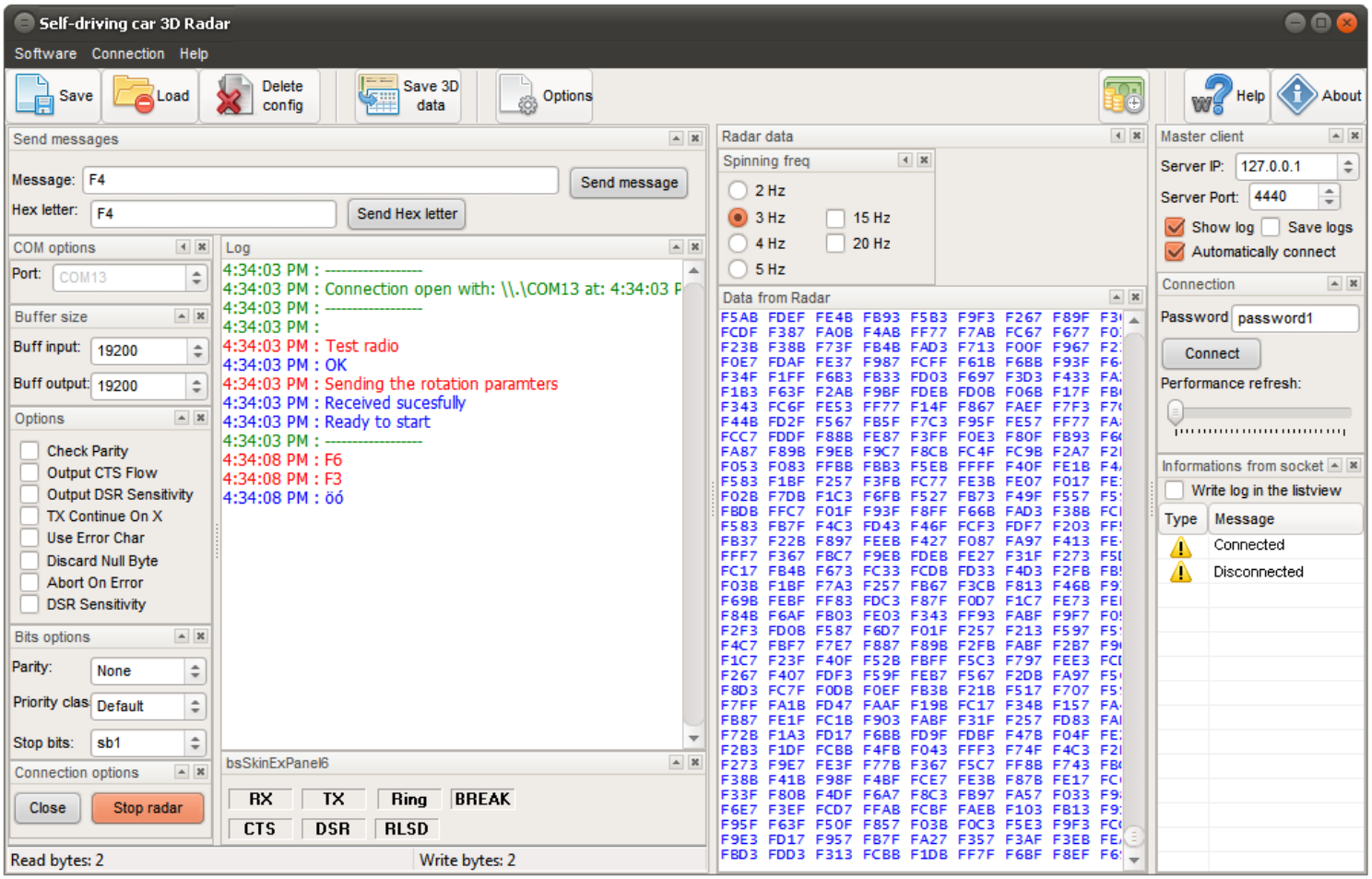
A stepper motor is used to spin the PCB with the photodiodes, alimentation and a PIC16F877. A photodiode is a type of photodetector capable of converting light into either current or voltage. My photodiode is a very special one and is used only in rangefinder; it is an **APD – Avalanche Photodiode Detector –** sourceat**200 V.**

No of data received: No of APDs * No of spins per second * No. of spin grades * No. bits
16 x 10Hz * 180 grades (or 360 or 18000) * 10 bits = 288,000 bits of data/s
=28.800 pixels with the resolution of 10 bits

The self-driving car with the 3D Radar



The data is sent via RS232 serial port. I realized also a software that is able to read the that from COM ports and to send the 3D Data to the supervisor software.



Traffic signs and GPS common database usage

The Traffic Signs Detection software recognizes the traffic signs and sends them to the GPS software component. The GPS Software receives the signs and if they already find in the common database, the software gives positive feedback. If there are other traffic signs saved in the common database but they weren't recognized by the Traffic Signs Detection algorithm, the traffic signs will receive a negative feedback and use these traffic signs in the calculation of the car's path. If this feedback is very low, the signs are automatically eliminated from the database. The communication between these two independent software components was done using a self-madeTCP/IPprotocol.

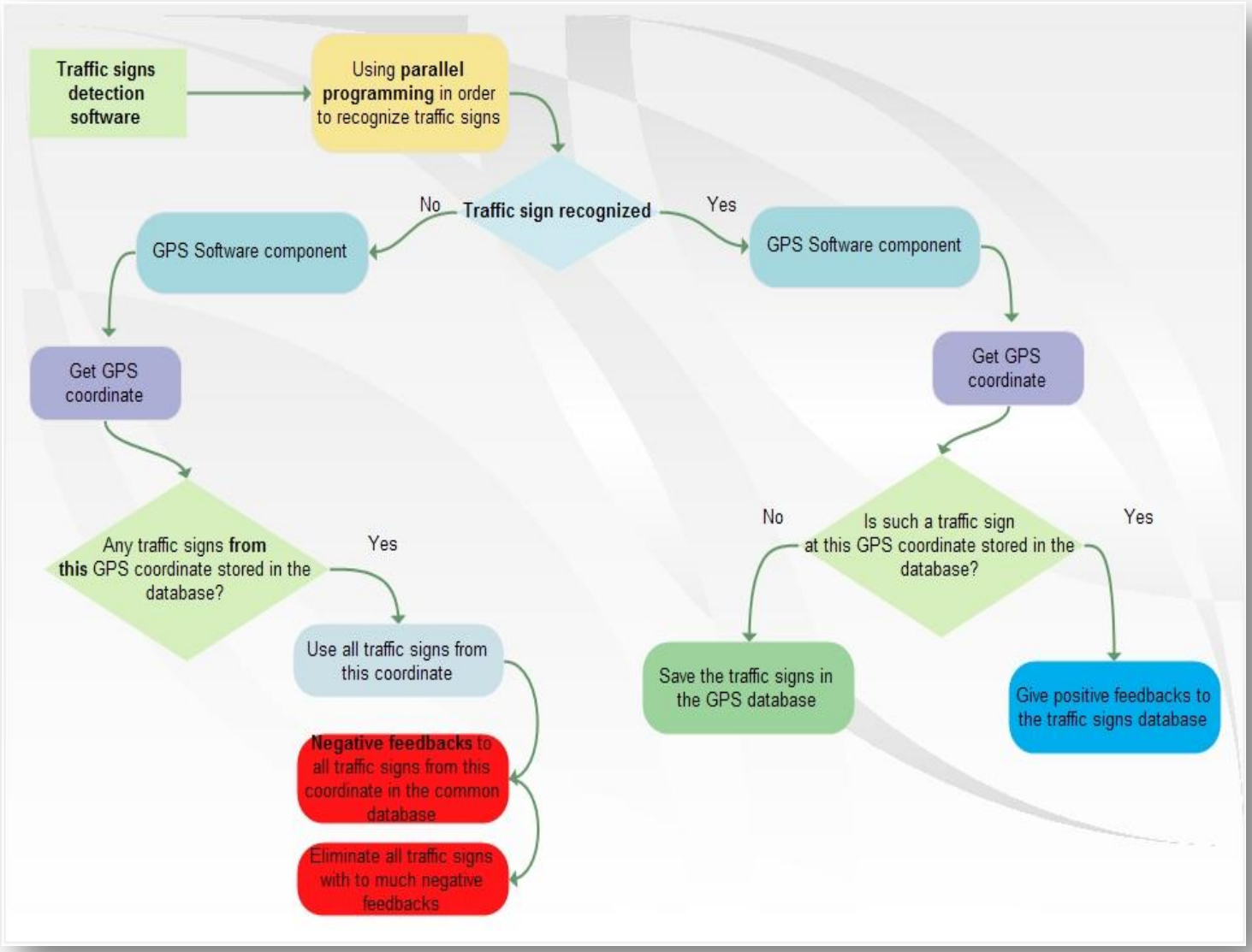
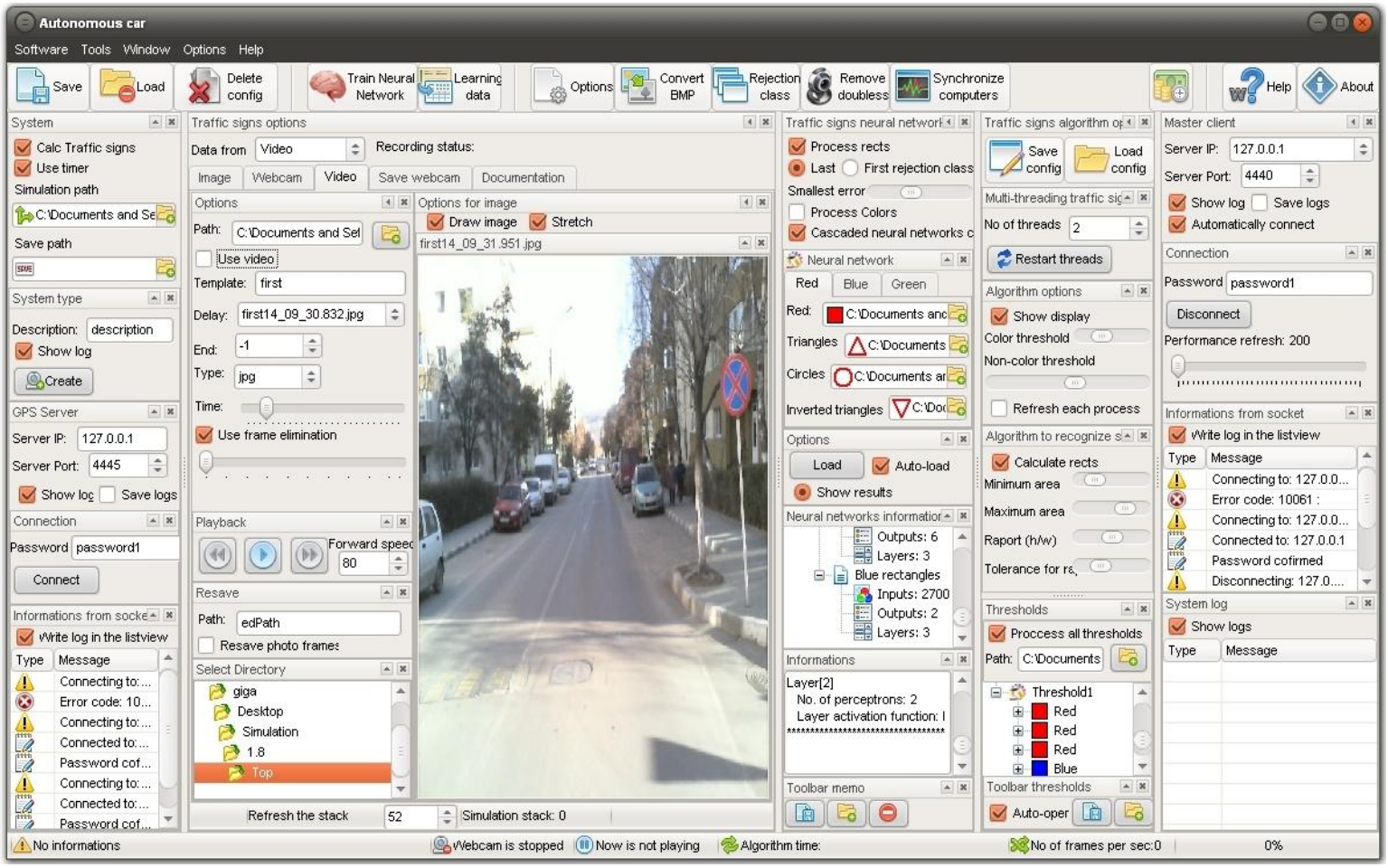
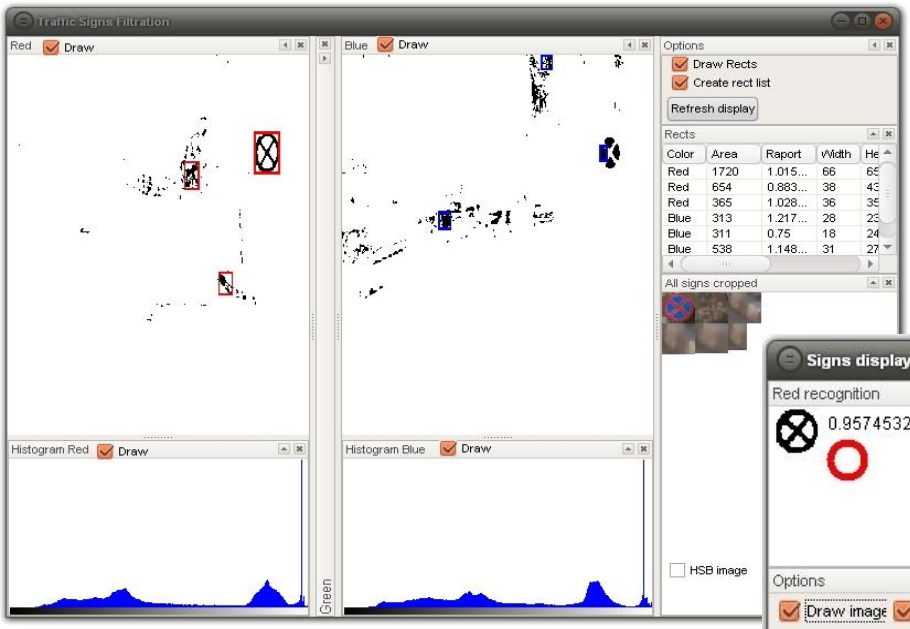


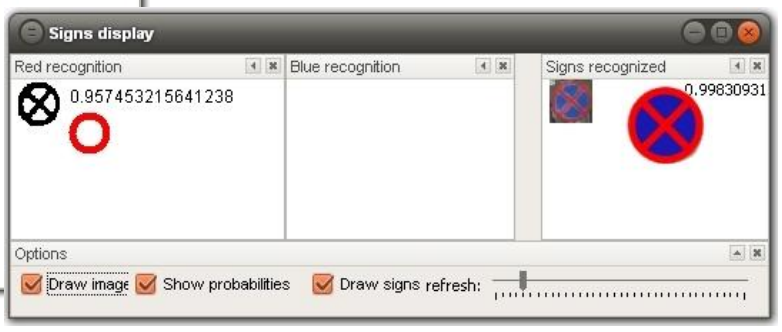
Image no. 1.6. In this diagram anyone can see how the common GPS database works and how the special property called "feedback" is used to calculate the probability of certainty for any traffic sign.

Traffic signs recognition software

As well as a human driver must be familiar with all the traffic signs, my self-driving car needs to be able to recognize **all** of the traffic signs too, and take decisions for all of these traffic signs. The traffic signs detection is based on color filtrations using HSV Spectrum filtration, many cascaded Multi Layer Perceptron Neural Networks, and a common traffic signs database using the GPS coordinates and Google Maps.







Abstractof the traffic signs detection multi-threading code:

The algorithm is **parallel** and it was developed using a multi-threading solution. The software generates a thread for each core. The images are taken from a top-view webcam and are inserted in a synchronous queue. Then, each CPU core, after finishing the last task, verifies whether there is anything in that synchronous queue, and if yes it will take the new task from the queue. The operations with the queue were done using critical sections.

Keywords: **Parallel code, multi-threading, HSV multiple thresholds, 9 cascaded Multi-Layer Perceptron Neural Networks, Synchronize.**

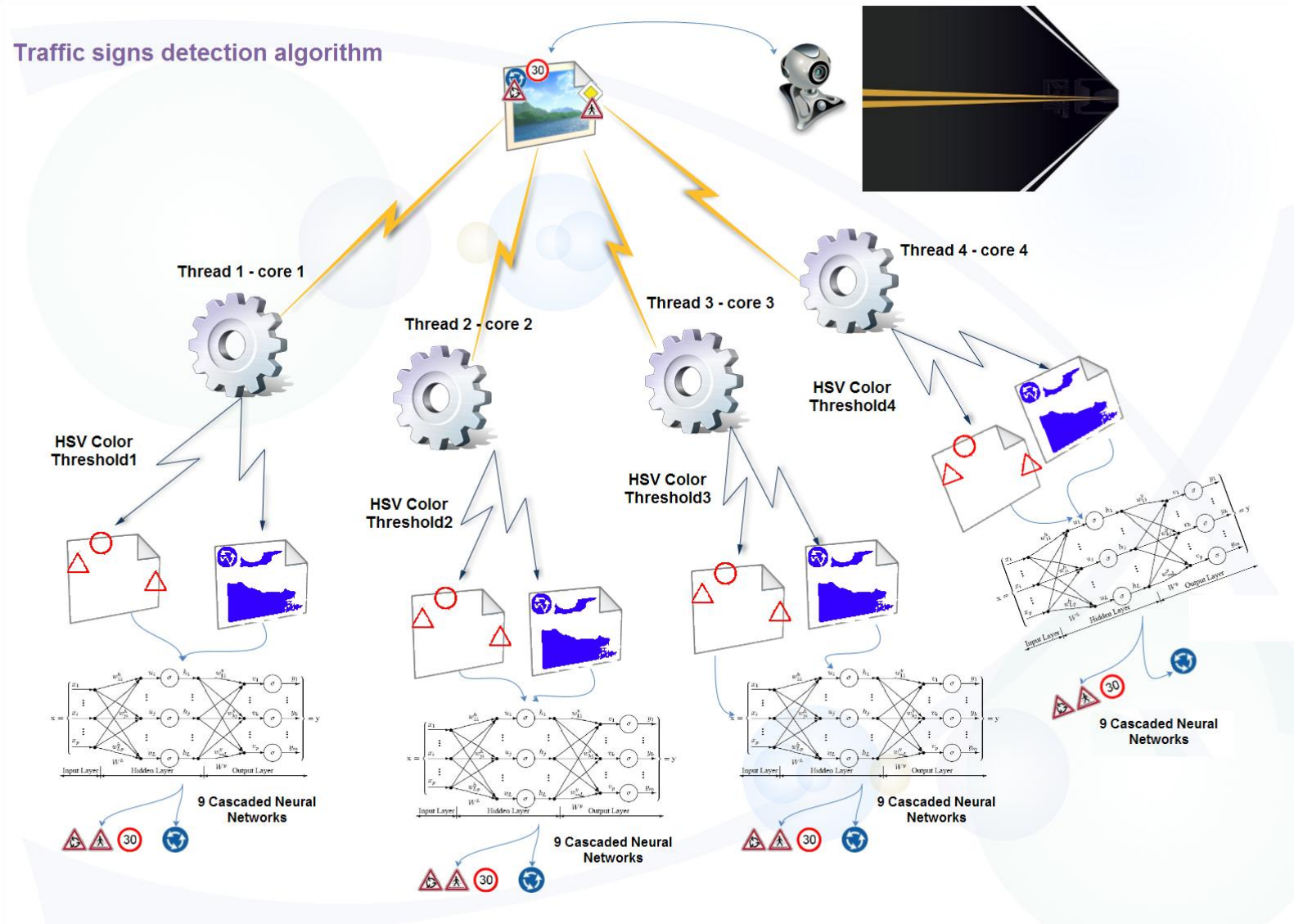


Image no. 1.7. This diagram illustrates how the Traffic signs detection algorithm is able to recognize the traffic signs from simple images. It was created using a multi-threading solution. Each image is processed by each CPU cores using a different HSV threshold.

Cascaded Neural Networks in order to recognize traffic signs

The traffic signs detection is based on Multi-Layer Perceptron Neural Networks. The neural networks were trained using real images of traffic signs. The neural networks were trained using Back-Propagation with Momentum algorithm.

There are two neural networks for traffic signs forms(blue forms and red forms).For each traffic sign form there is a neural network associated with. The networks were learned in this software.

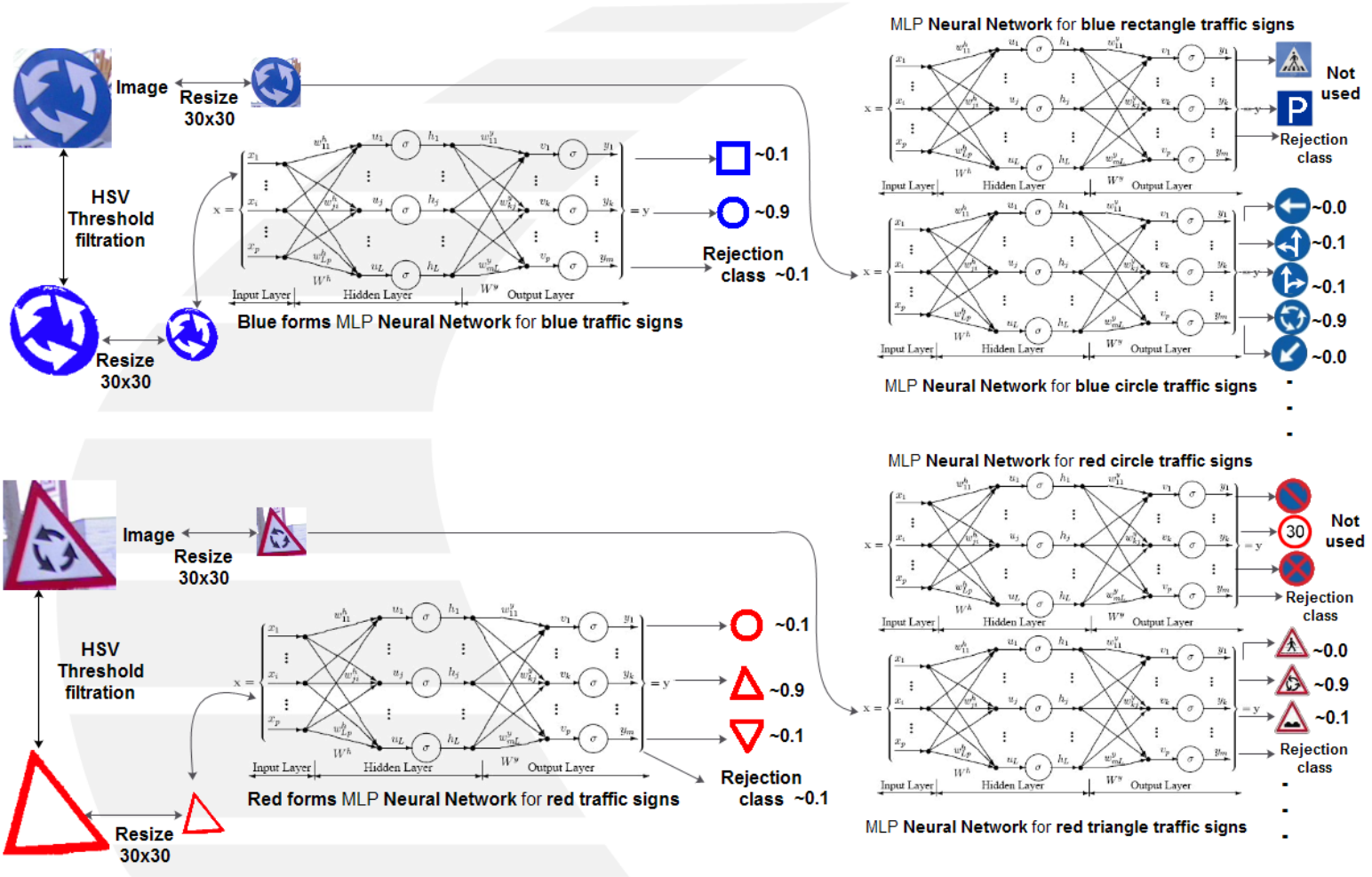


Image no. 1.8. This diagram illustrates how the neural networks are used to recognize traffic signs neural network. This neural method uses 9 cascaded Multi-Layer Perceptron Neural Networks. For training the neural networks I used Back-Propagation with momentum with more than 350 images as traffic signs. Currently the algorithm is able to recognize about 37 different traffic signs.

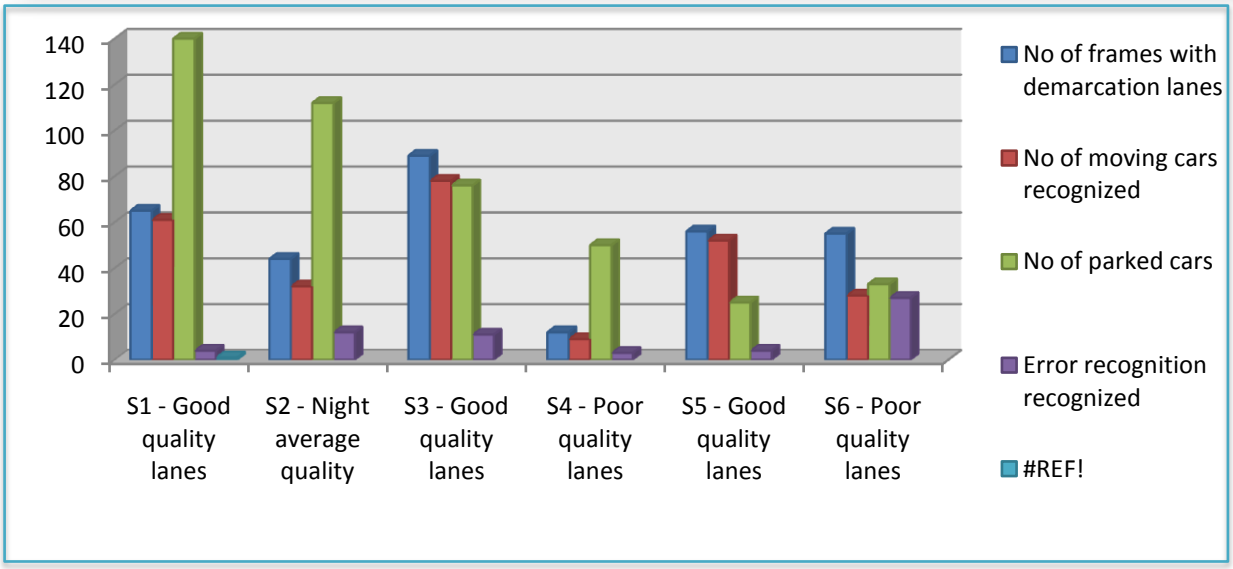
Sign color	Sign graphical form	No. of different traffic signs can be recognized
Red	**Circle**	**15**
Red	**Triangle**	**13**
Red	**Inverted**	**1**
Blue	**Circle**	**23**
Blue	**Rectangle**	**20**

Table no. 1.1. This table shows how many different traffic signs can be recognized by the cascaded Neural Networks of the software.

Real-time multiple vehicle detection and tracking using particle filters

For my projects I've developed a real-time vision system which is able to analyze color videos taken from a driver looking HD web camera in a car driving on a street. The system uses a edges(first derivative of the image), combination of color, and motion information to recognize and track the vehicles from the highway. The algorithm is based on particle filters.

The principle of the algorithm is based on a histogram. In the image processing theory, the histogram of an image refers to a histogram of the pixel intensity values (tones). The image histogram is a graph showing all of the pixels of an image at each different intensity value found in that image. For instance, in a 8-bit grayscale image there are only 256 different possible intensities, so the histogram will graphically display 256 numbers showing the distribution of pixels (intensity of the pixels) amongst those grayscale values. Histograms can be also calculated for color images - individual histograms of red, green and blue channels or 3-D histograms can be produced, with the three axes representing the red, blue and green channels and the brightness at each point representing the pixel count. Following the calculation of the histogram the software processes and uses statistics in order to identify the particles and to calculate the weight of each particle.



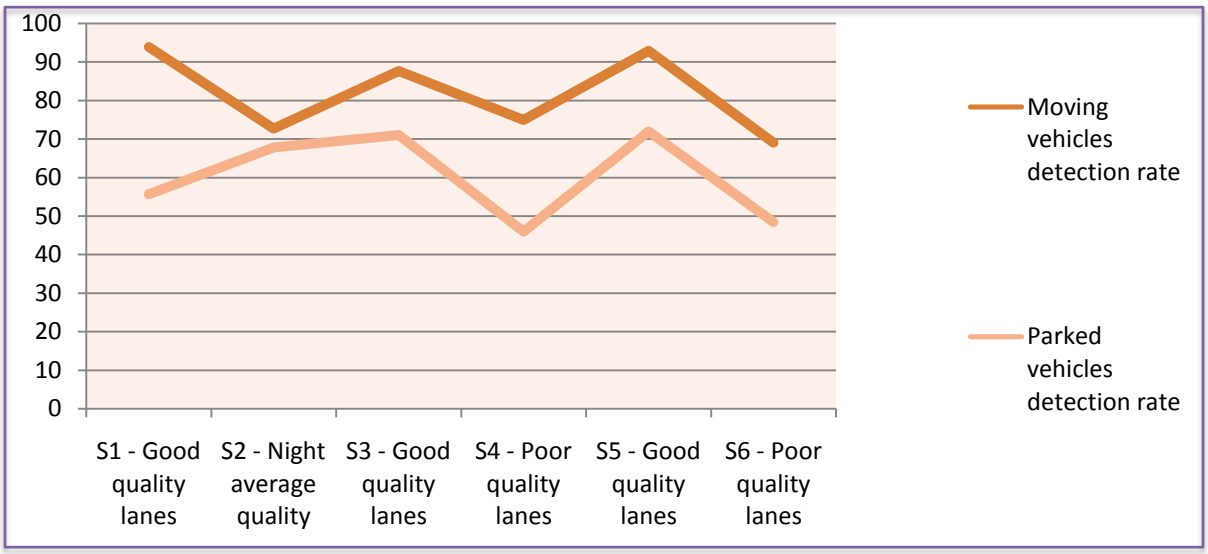
The chart describes the efficiency of the detecting cars using particle filters and neural networks.

The following chart shows the efficacy of the vehicles detection based on histogram tracking using Kalman filters(particle filters).

There are two types of cars:

- Moving cars
- Parked cars

The software was written in order to recognize more the moving cars, in order to avoid any kind of collision with them.

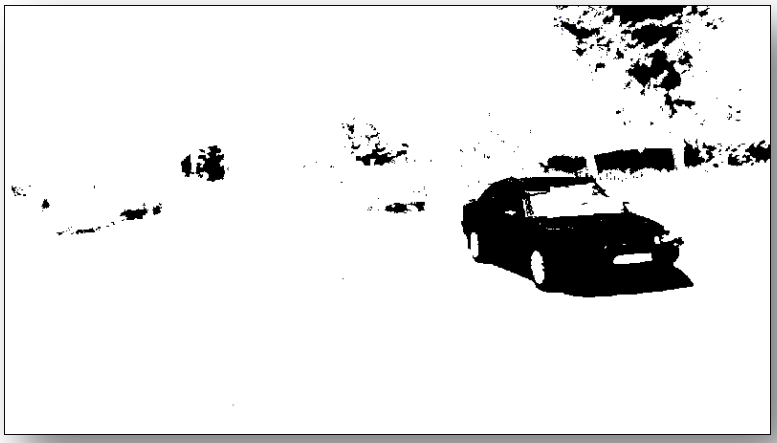


In the following picture anyone can see an image taken from the car's camera. All 5 cars

from this picture are painted entirely with the same color.

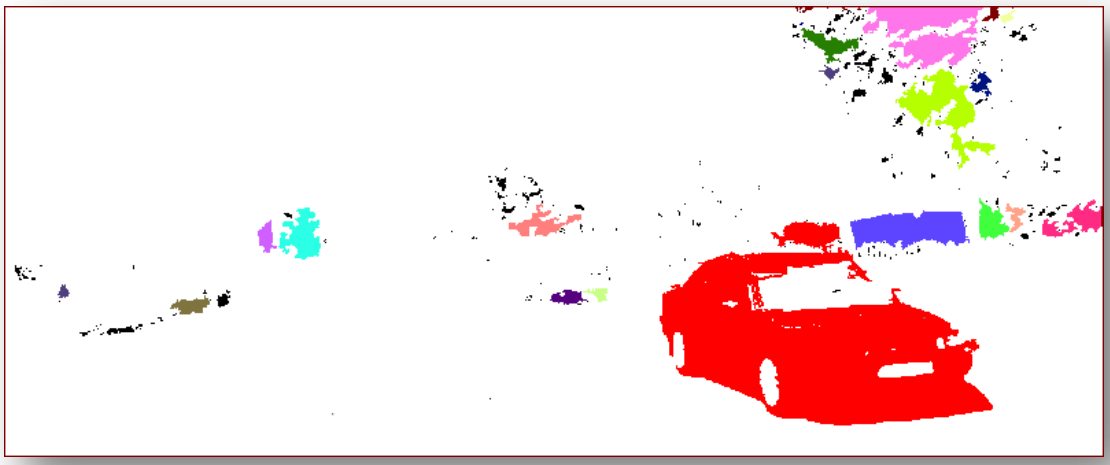
Description of the vehicle detection algorithm using Particle Filters

1. Most of the cars are composed of only one fundamental color. Initially the algorithm is searching for large shapes composed mostly from a single color. In the picture above can easily be identified 4 cars(two blue cars, one black car and one navy car). Using different color thresholds filtration we can identify where these cars are. The following image was obtained after the software binarized the image using the blue threshold = 105.

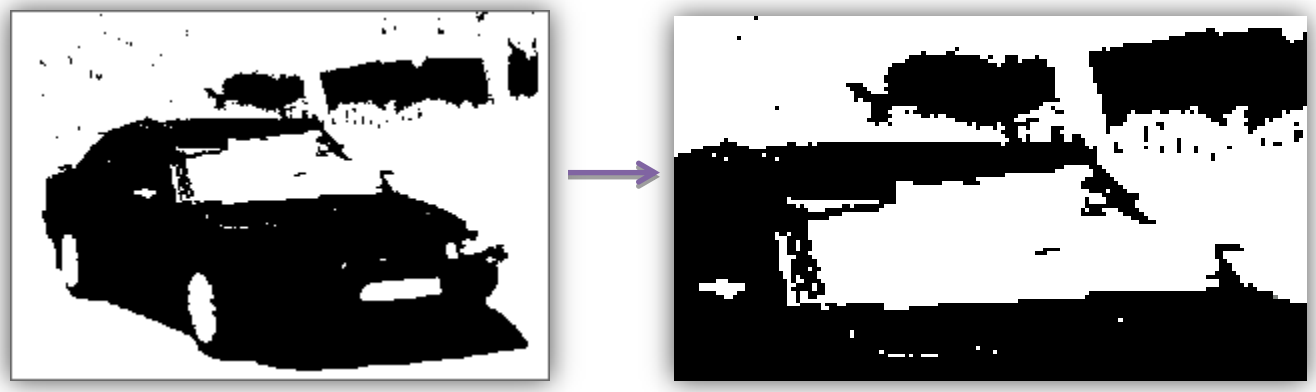


This picture was taken after thebinarization of the previous image using a blue color threshold = 105.

2. I am interested to do a segmentation to know how many big black shapes are in my pictures. The segmentation is just a simple fill. After the segmentation I will check if my black shape will be enough and fit for my car shape.



After this filtration with the color threshold, in many cases my car shape is joined to other noises from the background. In the previous image you can see that the car's color is the same with a small "cloud" of noise. In the following image, it is the same result from color filtration process. It can be noticed that the car shape is joined with a noise from the background.



In order to remove this joint problem, the image processing algorithm conceived by me will do 4 times an "open" morphological function.

3. Opening and closing are two mathematical morphology operators. The opening operation is derived from the dilation operator. The opening serves in computer vision and image processing for a morphological noise removal. The effects of the operator are intended to preserve the foreground region which has a similar shape to the structuring element (my car shape), or completely contains the structuring element, while eliminating all other regions(the background noise) of foreground pixels.

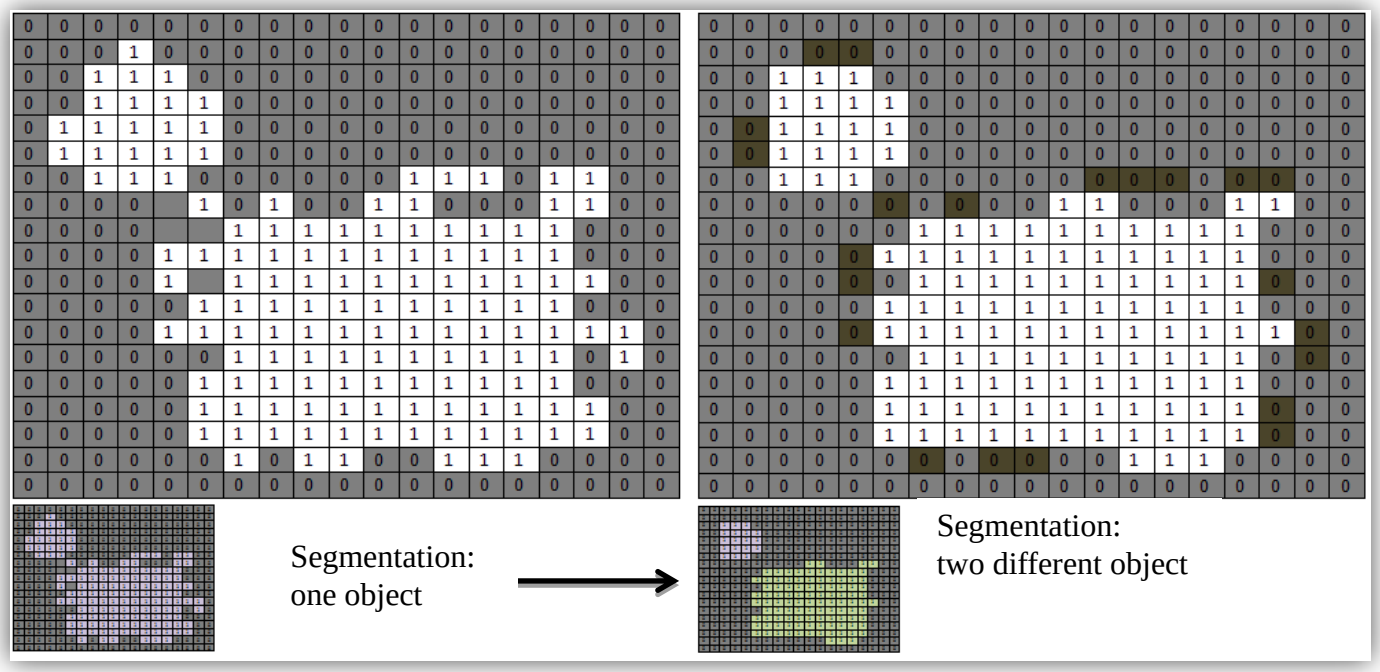
   In mathematical morphology, opening is the dilation of the erosion of a set A by a structuring element B:

$$A \odot B = (A \ominus B) \oplus B$$

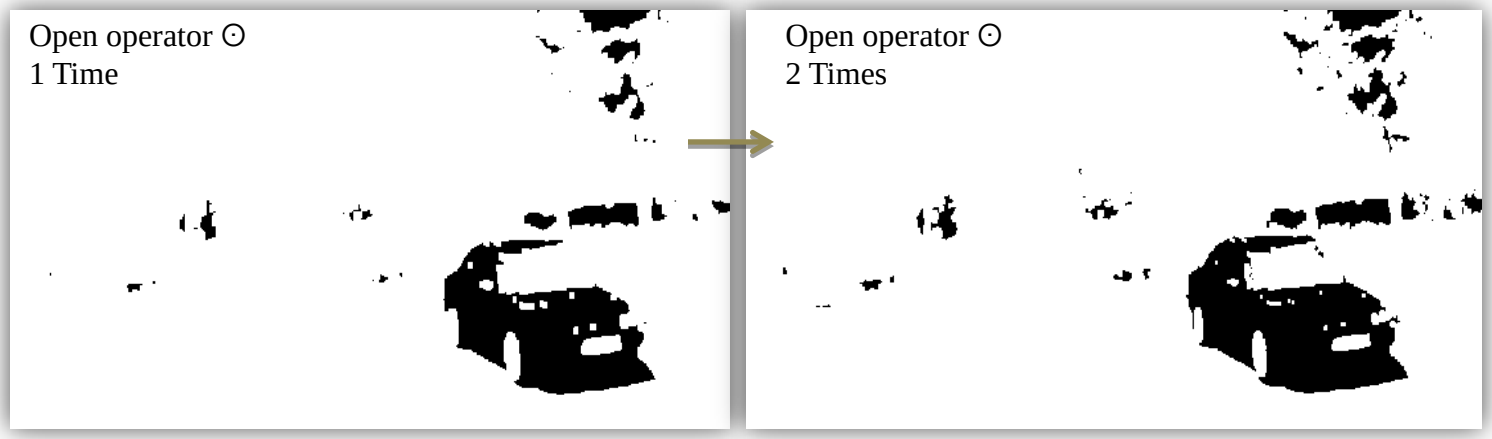
   where $\ominus$ and $\oplus$ denote erosion and dilation, respectively.

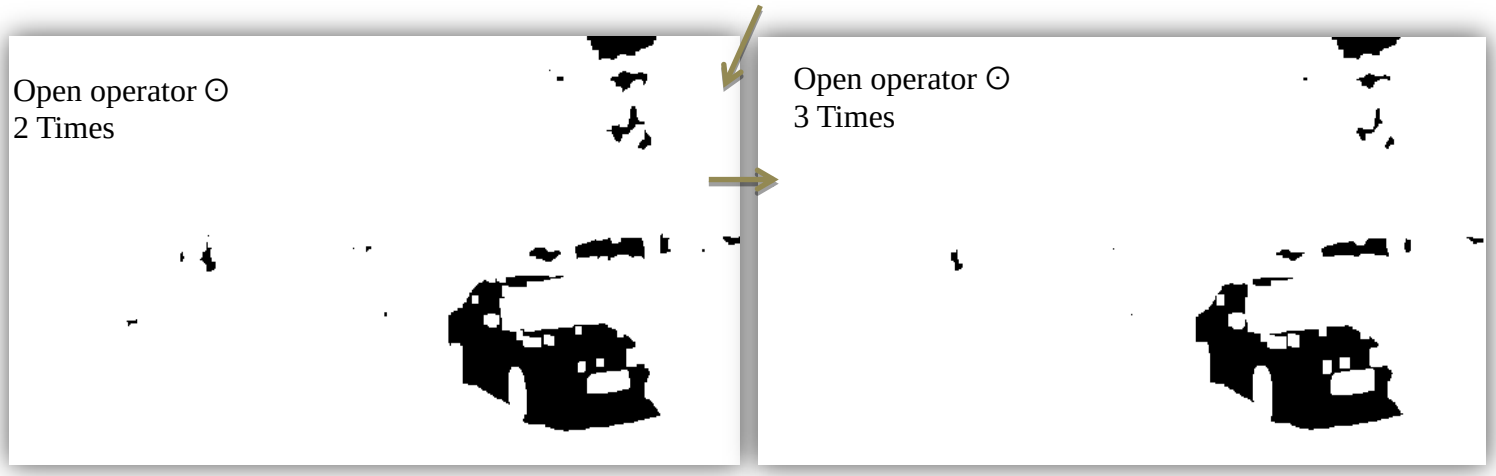
   In the most of the applications, the opening function is used to remove small objects from the foreground of an image, placing them in the background. This technique can be also used to find specific shapes in an image.

The effect of opening operation is like the erosion in that it tends to remove some of the foreground pixels from the edges of regions of foreground pixels. In the following image you can see that after one opening operator the two objects will be separated and the segmentation function will return two different objects.

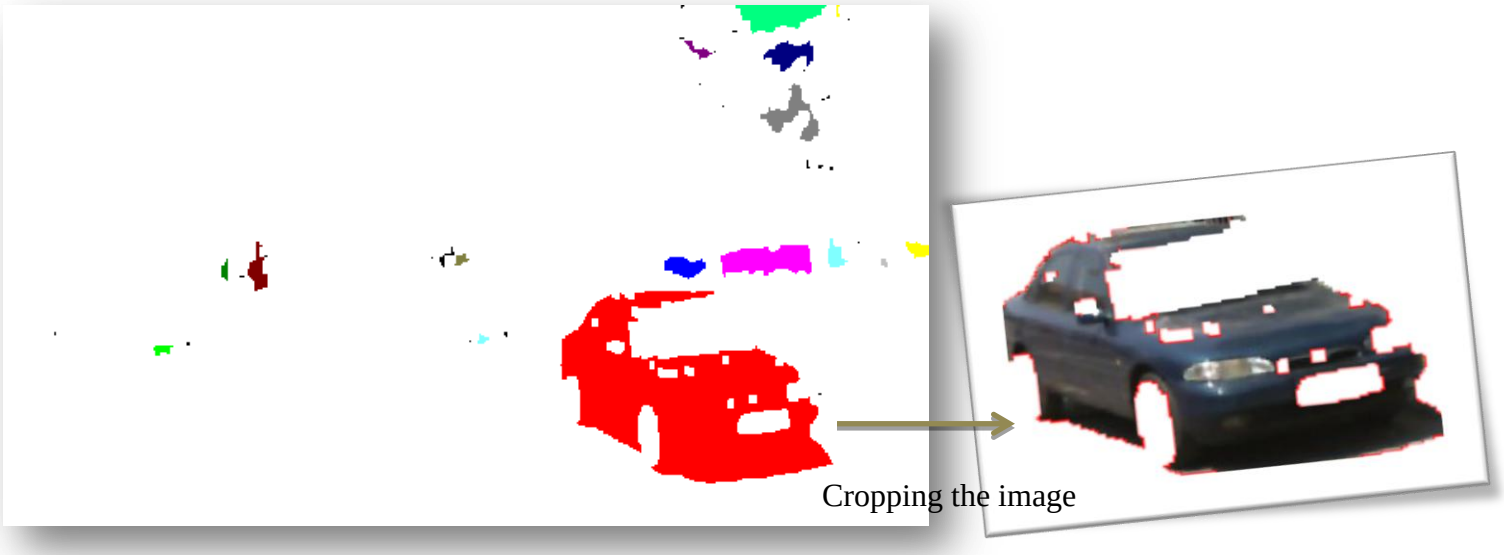


The results of 4 cascaded open morphological operator applied on the color threshold binarization.



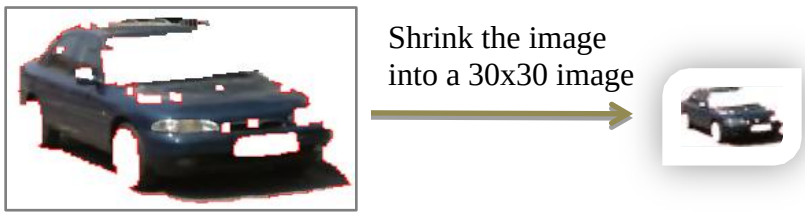


4. After the usage of a few open morphological operators the car's shape is totally different from all other noise shapes. The goal of image segmentation is to cluster pixels into salient image regions(corresponding to individual surfaces, objects, or natural parts of objects). In this step the goal of the image segmentation is used to recognize all image regions.



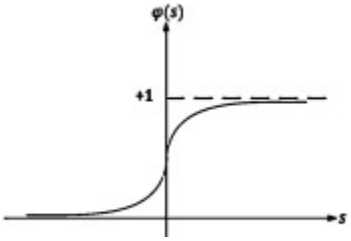
The retrieved image has a huge probability that it can be a car

5. In order to check whether there is a car in the retrieved image from the segmentation, the algorithm uses a Multi-Layer-Perceptron Neural Network. As the resolution of the cropped image is very high, (for computers high resolution images are mostly redundant), the algorithm conceived by me will shrink the image into a small image by 30x30.



6. The Feed-forward Neural Network will be able only to classify and recognize whether in that image it is a car, so the colors are redundant.

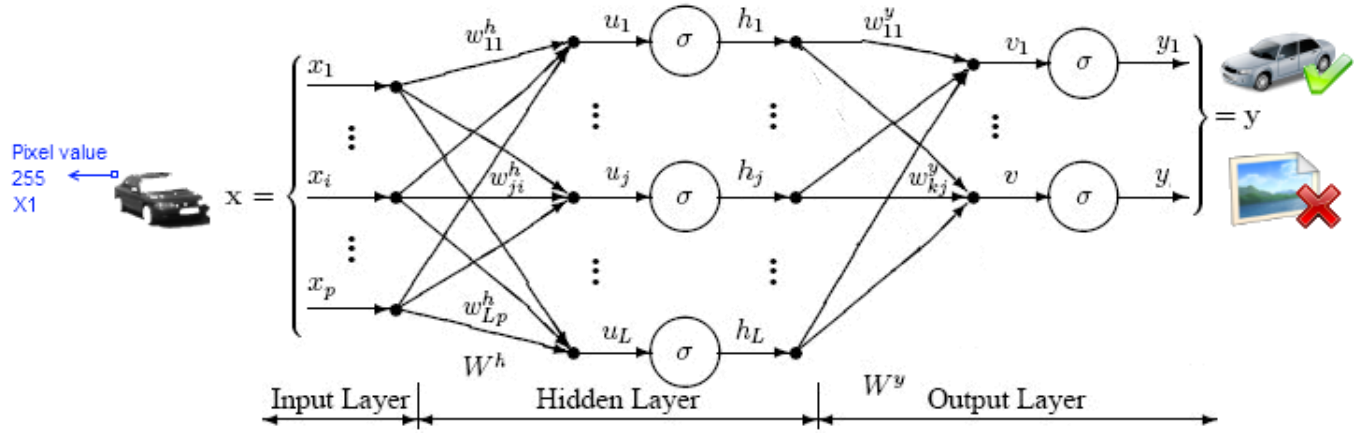
The Neural Network topology is quite simple, it is: [900,50,2]. There are 900 inputs(for each pixels from the minimized image, the value are normed to 1000), 50 perceptrons in the first hidden layer with the logsig (the graph on the right) as activation function, and only two perceptrons on the output layer with the logsig as activation function.One output is used to describe whether the input(the image) belongs to the cars' cluster or to the rejection cluster. The second cluster is used because the MLP Neural networks are not able to make rejections properly.



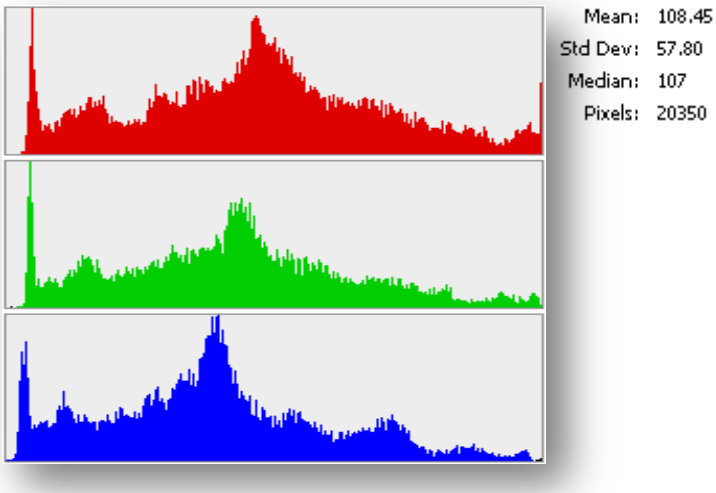
$$logsig(x) = \frac{1}{1+e^{-x}}$$

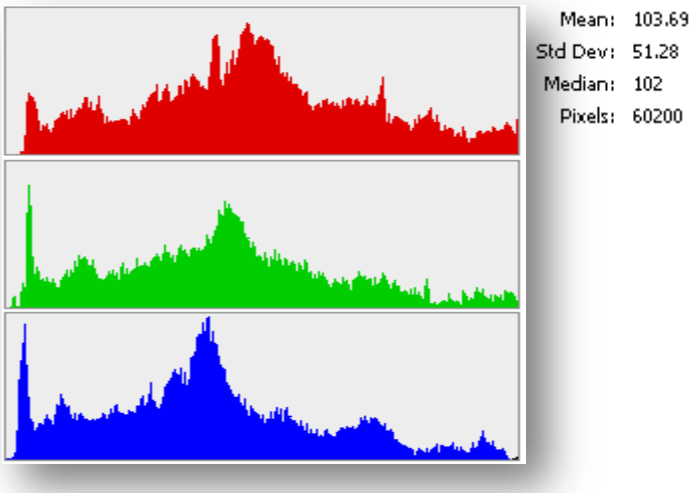
logsig is only a transfer function. Transfer functions calculate a layer's output from its input.

In the following image you can see the feed forward neural network's topology (architecture). Each output of the neural network has values between [0...1]. If The value is closer to 1, it means it's a large probability to be in that cluster (the cluster of cars or in the cluster of rejections).



7. If the neural network's result is satisfactory, then the particle filter algorithm tries to track the car in the next frames. The principle is quite simple: the car should not disappear in the next frames. For the particle filter implementation I am using the Kalman Filters. Initially the weight of the cluster is small, but if in many consecutive frames I can still recognize the car and the position of the car is changing (the car is moving) then the weights of the particles will increase. In order to calculate the posterior probability and track the cars my Kalman Filter algorithm uses the histogram. The concept that the software is able to track the cars using a histogram is: the colors of the cars are not changed dramatically in consecutive frames. Let's look at an example.





Calculating the difference of the histograms:

In the following picture can be seen the difference (the black pixels from the following histograms) between these two consecutive frames. In conclusion, you can understand why such a particle filter using a histogram will be able to recognize and track vehicles.

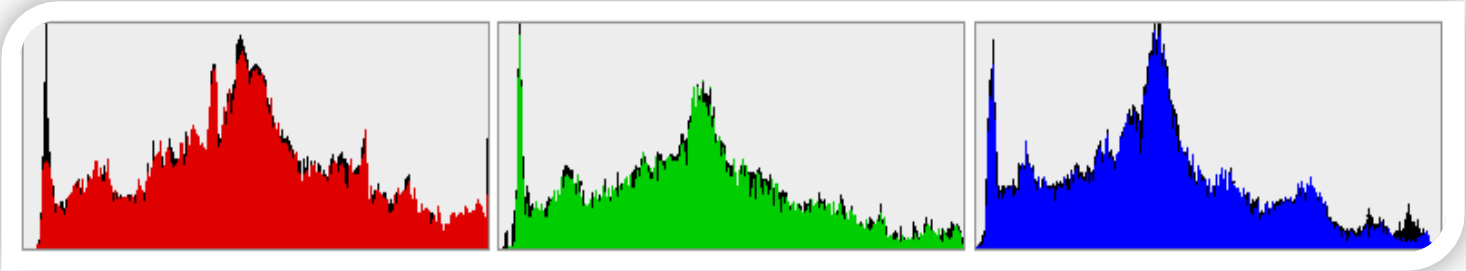
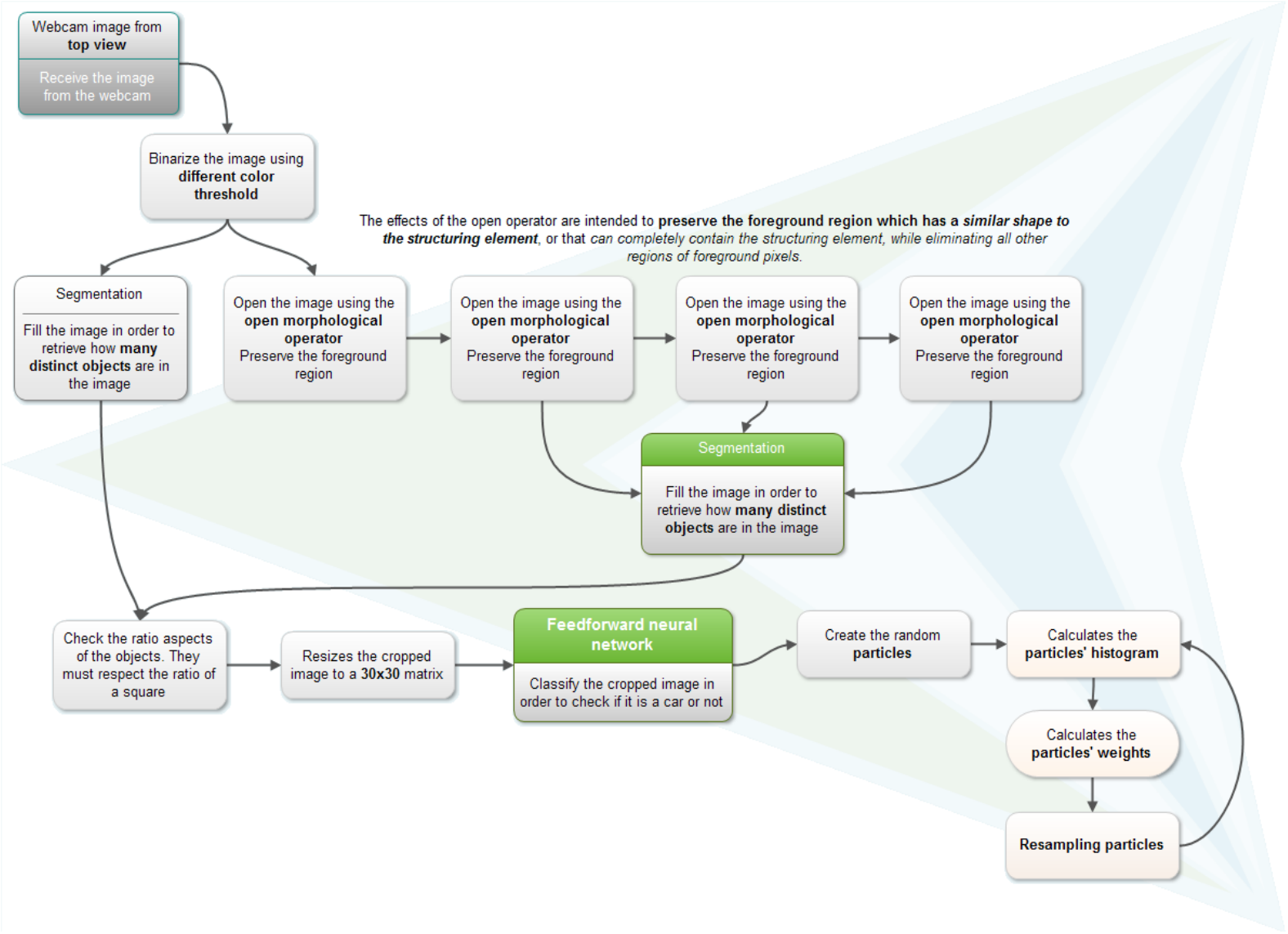


Diagram of the algorithm described.

The diagram below describes the vehicles detection and recognition tracking algorithm using particle filters (Kalman filters). Initially the algorithm tries to recognize car shapes. If such shapes are recognized, the particle filter tracking is enabled.

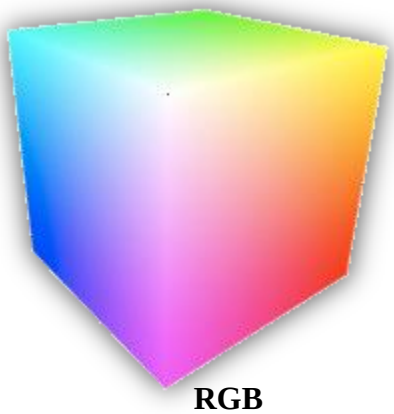


HSV Filters - filtration for colors.

By simple observations anyone figures that all of the traffic signs are surrounded by elementary graphical figures like blue circles, red triangles, etc… In order to recognize the traffic signs from images,the first step is to locate where the traffic signs color signs are. Using the RGB (Red, Green, Blue) representation of the colors it is very difficult to write a piece of software that filters colors.

HSV is a cylindrical-coordinate representation of points in an RGB color model. The representation rearranges the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation. HSB stands for **hue**, **saturation**, and **value/brightness/lightness.**

HSB

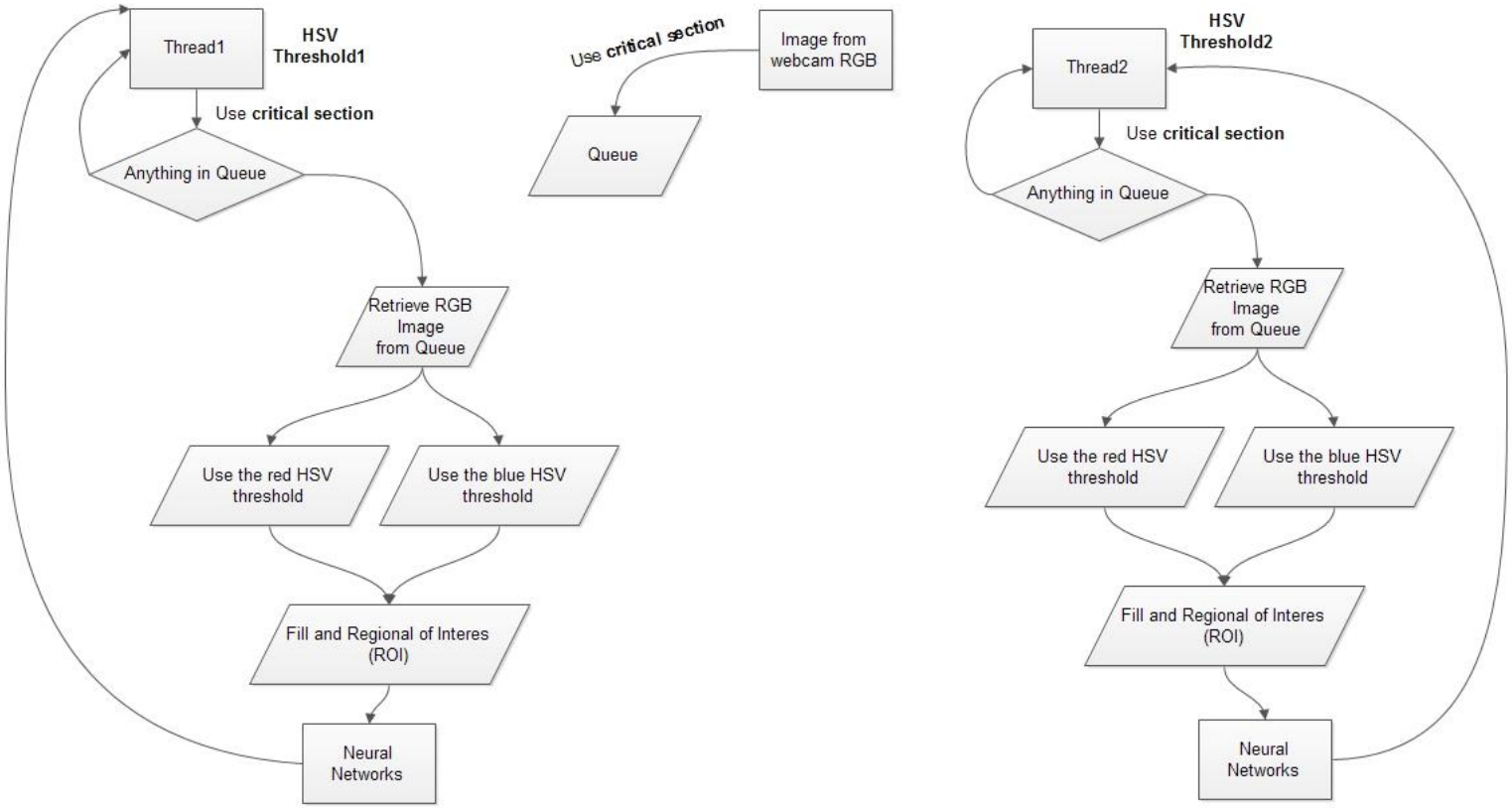


RGB

From the webcam images I am interested to recognize the elementary colors. Using the HSV it is easier because the Hue component returns the color and saturation and brightness returns the intensity of that color.

All of the traffic signs are filled with Red, Blue and Green. The filtration is intended to search initially the position of the traffic signs using a filtration of the colors.

Image no. 1.9. The following program flowchart shows the multi-threading solution implemented by me for the HSV filter. It is used to search for the traffic signs color.



Initially, the image taken from webcam will be processed for colors using HSV filters

Using the HSV to filter after red color

Using the HSV to filter after blue color

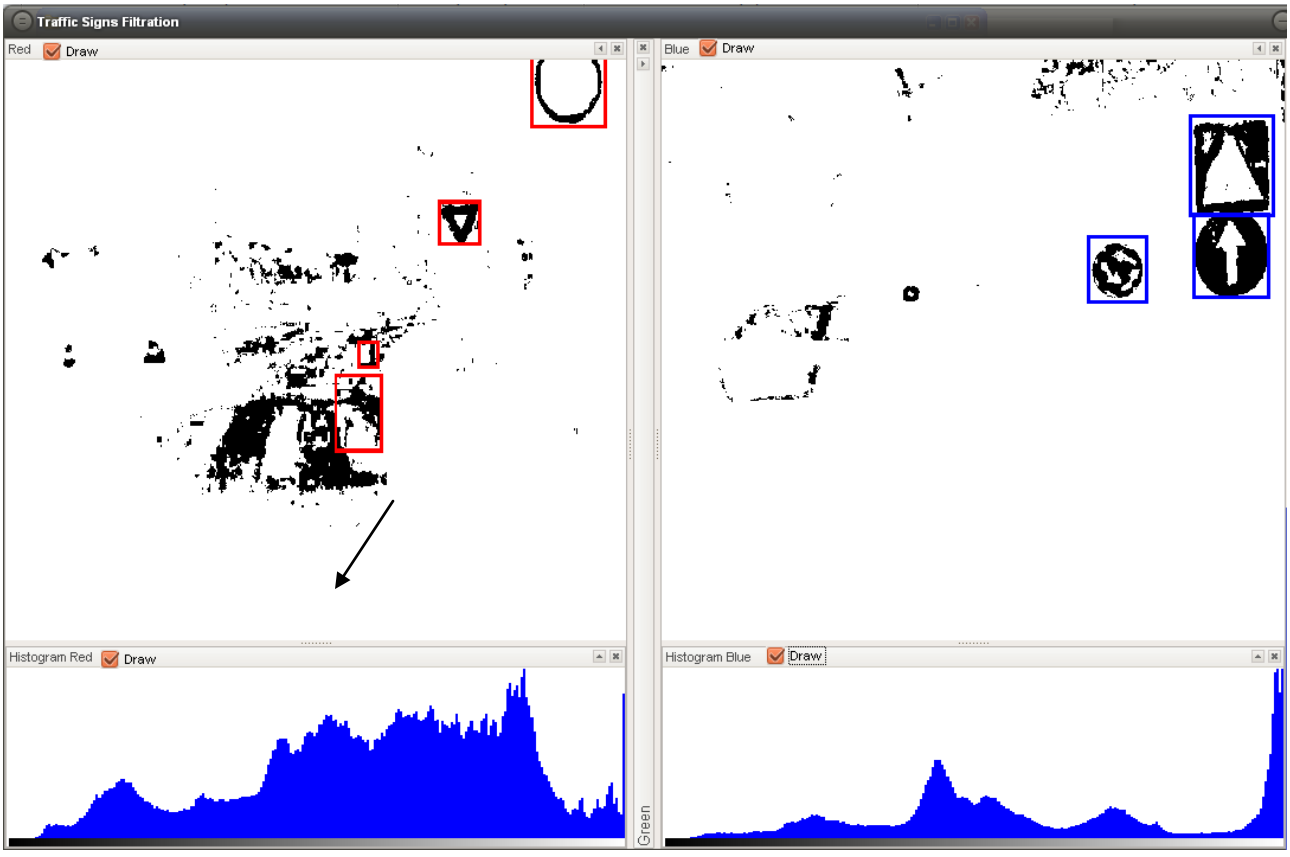
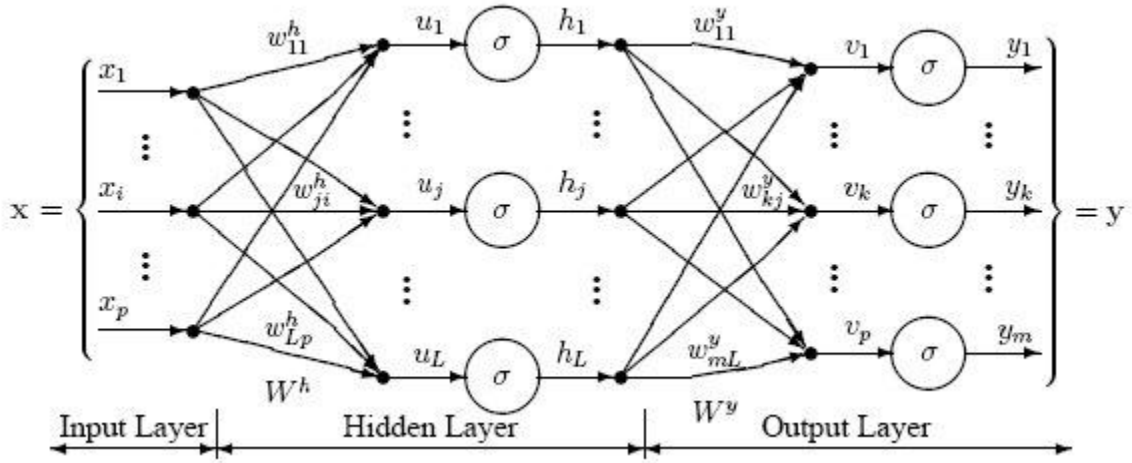


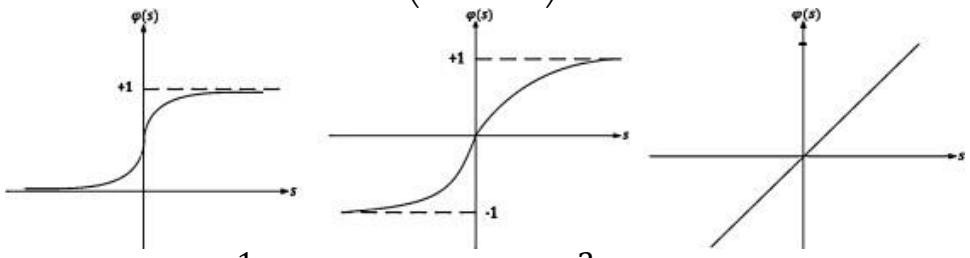
Image No. 1.10. The result of a frame filtered using only one HSV Threshold.

Neural Network

Artificial Neural Networks are information processing systems composed of simple processing units, interconnected and acting in parallel. These elements are inspired from the biological nervous systems. As in nature, the network function is determined by the connections between the network's elements. Weights of connections between the perceptrons are those that remember the information learned by the neural network. The networks are trained by adjusting these weights, according to a learning algorithm.



$$h_j = \sigma\left(\sum_{i=1}^{p} x_i w_{ji}^h\right) \qquad y_j = \sigma\left(\sum_{i=1}^{L} h_j w_{ji}^y\right)$$



$$logsig(n) = \frac{1}{1+e^{-n}} \quad tansig(n) = \frac{2}{1+e^{-2*n}} - 1 \quad pureline(n) = n$$

$$logsig'^{(n)} = n*(1-n) \quad tansig'^{(n)} = 1-n^2 \quad pureline'(n) = 1$$

Artificial Network Networks characteristics:

- A distributed representation of the information: The information is stored in a distributed network (in the network's weights), which means that any inputs and outputs depend on all weights from the neural network.
- The ability of generalization from some training data to some unknown situations. This feature depends on the number of weights and the topology, for example the size of the neural network. It is found that increasing of the neural network size leads to better memorizing the training data but to lower performance on test data, which means that ANN has lost the ability to generalize. Determining the optimal number of perceptrons from the hidden layers, is a key step in designing an ANN. It can be done by choosing different topologies of the Artificial Neural Networks until it begins to decrease performance on the test set.

- Tolerance to noise: an ANN can be trained, even if the data are affected by noise, - obviously decreasing its performance.
- Resistance to partial destruction: because the representation of the information is distributed through the entire neural network, it can still operate even if a small part of the network was destroyed.
- Fast calculus: learning an Artificial Neural Network using a training data it's time consuming, but once trained it will quickly calculate the network output for a given input(test).

Anyone can train an ANN in order to perform/approximate a particular function by adjusting the values of the connections (weights) between elements.

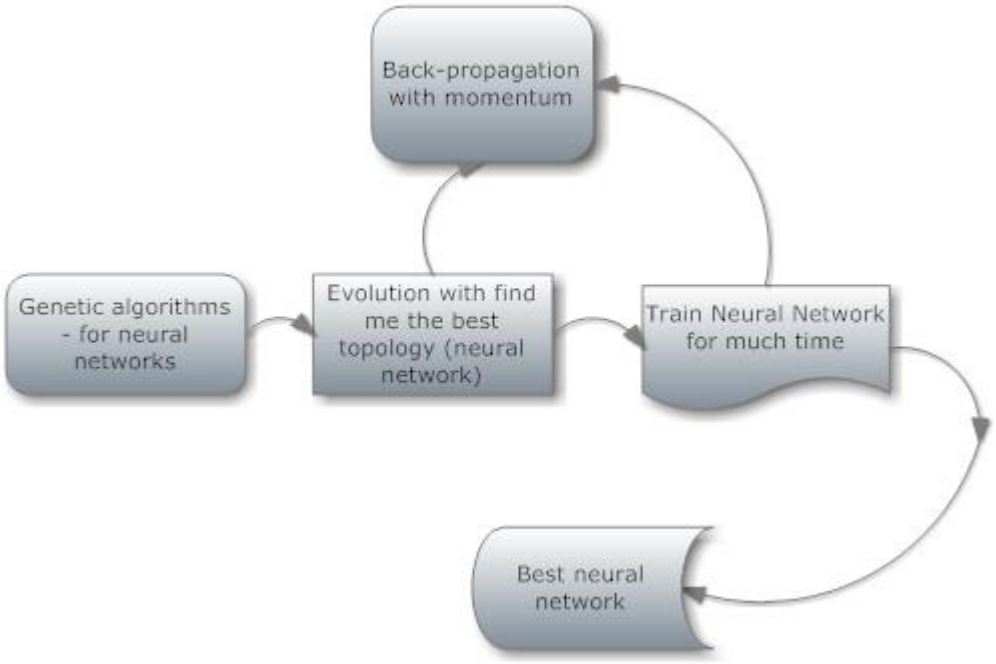
The ANN is usually adjusted (trained) so that a signal at the entrance to involve a certain output (target). The network is adjusted on the basis of comparison with the target response, until the network output matches the target. Because the learning algorithm of this neural network is supervised I need to give it several pairs input / output.

Applications of the Multilayer Perceptron Artificial Neural Networks

Neuralmethodscanbesuccessfully appliedinsoftwarewithincompletedata, erroneousvaluesaround at some valuescalledclusters.It's a visual class which allows you to create Multi-Layer-Perceptron Artificial Neural Network.The networks are trained with some representatives of the clusters. On the simulation the network must be also able to recognize other values near to the cluster. The neural networks manage the classification of the inputs in clusters and it is a supervised learning**.** Again this is not a pattern-matching or hamming distance, but a way to work with uncertain data with perceptron paradigm. It has really good results in many problems. A new method is probabilities using Hidden Markov Models. I haven't implemented a HMM method so far, but by the time of the contest, I will have implemented a HMM for better results.

Training the neural networks

To train the neural networks I have used the Back Propagation with momentum. In most of the cases I also used Genetic Algorithms to find the best topology of the neural network.



The algorithm

Many neural networks were createdin the application, exactly Multi-Layer-Perceptron Neural Networks, which were trained using the **Back-Propagation** algorithm. The networks were trained with some cluster represents and during simulation assigned with values from cluster, so that the software could generalize the clusters (the traffic signs). I want to remind you that this is not a pattern-matching or a metric distance(or Hamming), it is a method to work with uncertain data

Along with this software, I implemented a few functions that allow extracting the content from the image- without these it will be very hard to recognize the tiles. Software like: How many apples are in this photo? is very hard to write and achieve. There is necessary a series of steps, of image processing.

Using Genetic Algorithms to find the best topology of the neural network

I used Genetic Algorithms to find the best topology. All of the Neural Networks are made of chromosomes. The differences between the chromosomes consist in the configuration of the network (functions used, number of the layers, and number of the perceptrons on each of the layers)

The purpose is to learn and find the best neural network topology in order to obtain the smallest error. So the fitness function of the genetic algorithm is the last error after the training of the neural network. I used the wheel of fortune and crossover of the networks to obtain another type of network from the best and the weakest chromosome (to make a crossover with best and weakest). I also used the tournament in the idea of obtaining a better chromosome next epoch.

The genetic algorithm evolved for about 23 hours in order to find the best topologies for the neural networks.

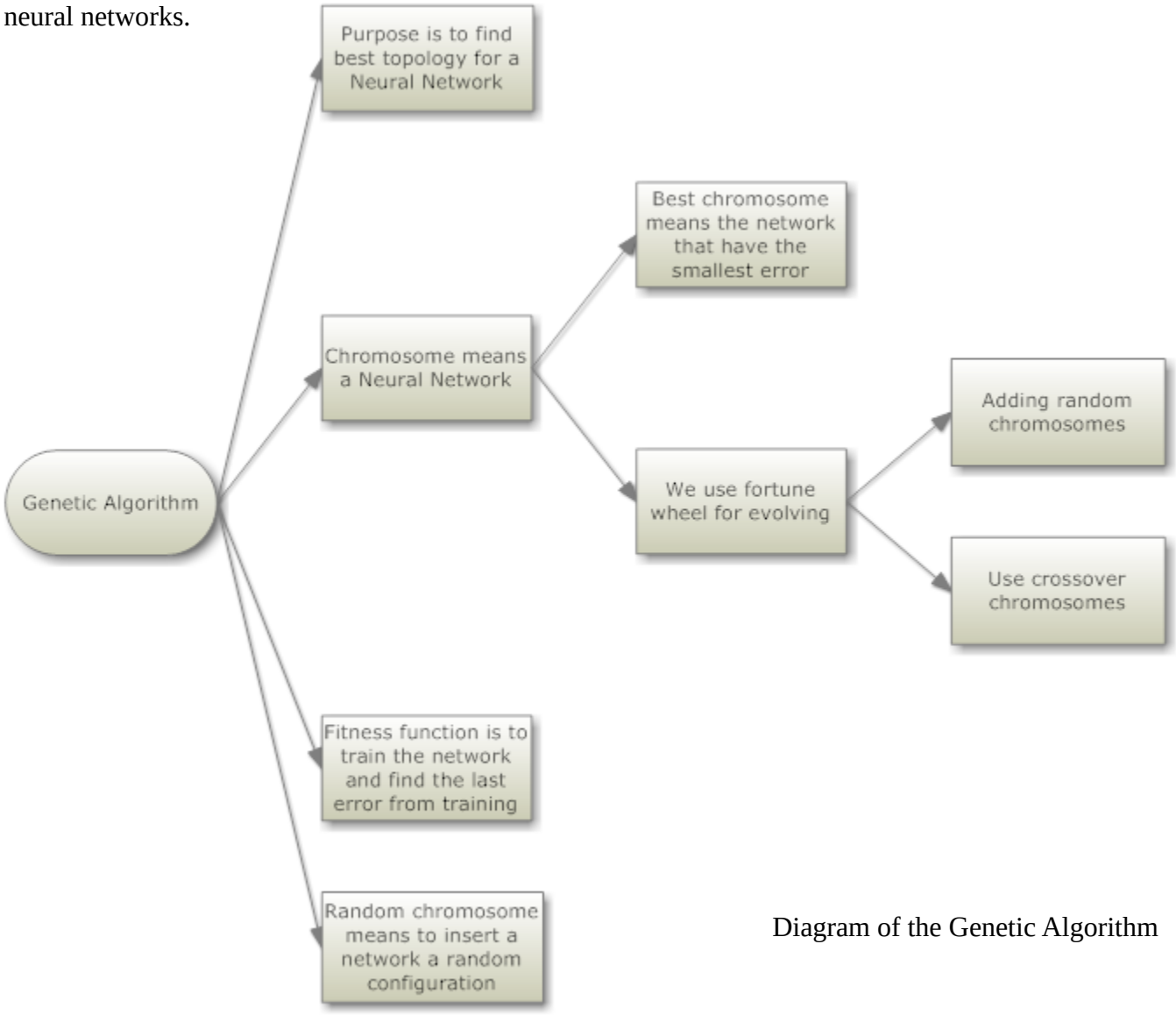


Diagram of the Genetic Algorithm

BackPropagation (without momentum)

In order to calculate the layer deltas anyone needs to **work backwards through the network**.Assuming a linear error function(a pureline activation function), such as the following:E = (a - i).

The layer deltas are calculated with the following formula:

$$\delta_i \begin{cases} -Ef'_i & ,ouputs\ nodes \\ f'_i \sum_k w_{k,i}\delta_k & ,inputs/hidden\ nodes \end{cases}$$

To sumarize for each output of the neural network take the inverse of the error multiplied by the derivative of the sum of the node. For all other layers we need to multiply by the derivative of the sum of the node by the sum of each weight multiplied by the node delta from the next layer. The following formula can be used for each weight from the neural network in order to compute the gradient from the weight k to weight i.

$$\frac{\delta E}{\delta w_{i,k}} = \delta_k o_i$$

Notations used in the Back-Propagation

q number of training sets

p_Q – input, from Q set

t_Q – output(target), from q set

E – error calculated in the current step

k – step (epoch)

m – layer (1th layer is input, [2nd .. m-1] hidden layers, [m] output layer)

$b_i^m(k)$ – ithBias Perceptron from **m**thlayerat kthstep

$w_{i,j}^m(k)$- Weight perceptron ith from **m**thlayer and perceptron j from the **m-1** layer at k step

$\forall$- learning rate

s_i^m - sensitivity perceptron ifrom**m**layer at current step

n_j^m - value perceptronj from **m**layer at current step

a_j^m - lay out perceptron j from **m**layer at current step = $F(n_j^m)$

ϑ- variable learning rate, from Darken Moody algorithm

$($ - coefficient moment from Back-Propagation with momentum

$F^{m\prime}(a^m)$is the Jacobian Matrix

$a,b,w,F,s,n$are matrix

i,j,k, $\forall$,ϑ ,$($ are numbers(scalar)

Training set

$\{p_1,t_1\},\{p_2,t_2\},\{p_3,t_3\},\dots,\{p_Q,t_Q\}$

Mean square error

$F(\boldsymbol{X}) = E[e^2] = E[(t-a)^2]$(I have used $E = (t-a)$, not the Mean Square Error)
Weight updates:

$$W_{i,j}^m(k+1) = W_{i,j}^m(k) - \forall s_i^m a_j^{m-1}$$
$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) - \forall \boldsymbol{s}^m \left(\boldsymbol{a}^{m-1}\right)^T$$

Bias update:

$$b_i^m(k+1) = b_i^m(k) - \forall s_i^m$$
$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - \forall \boldsymbol{s}^m$$

Jacobian Matrix

$$\frac{\partial \boldsymbol{n}^{m+1}}{\partial \boldsymbol{n}^m} = \boldsymbol{W}^{m+1}\boldsymbol{F}^{m\prime}(a^m)F^{m'}(a^m) = \begin{bmatrix} f^{m\prime}(a_1^m) & 0 & \cdots & 0 \\ 0 & f^{m\prime}(a_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f^{m\prime}(a_{s^m}^m) \end{bmatrix}$$

Finally, back-propagation sensitivity,

For last layer$\boldsymbol{s}^m = -2\,\dot{F}^M(n^M)(t-a)$

For all others layers $\boldsymbol{s}^m = -2\,\dot{F}^M(n^M)(W^{m+1})^T s^{m+1}$, $\quad m = M-1, \ldots, 2, 1$

So weights update:

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) - \forall \boldsymbol{s}^m \left(\boldsymbol{a}^{m-1}\right)^T$$

Biases update:

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - \forall \boldsymbol{s}^m$$

- **Back - Propagation with momentum -** The weight & bias updates will be overload with the following formula:

$$\boldsymbol{W}^m(k+1) = ()\boldsymbol{W}^m(k-1) - (1-()\,\forall \boldsymbol{s}^m \left(\boldsymbol{a}^{m-1}\right)^T$$

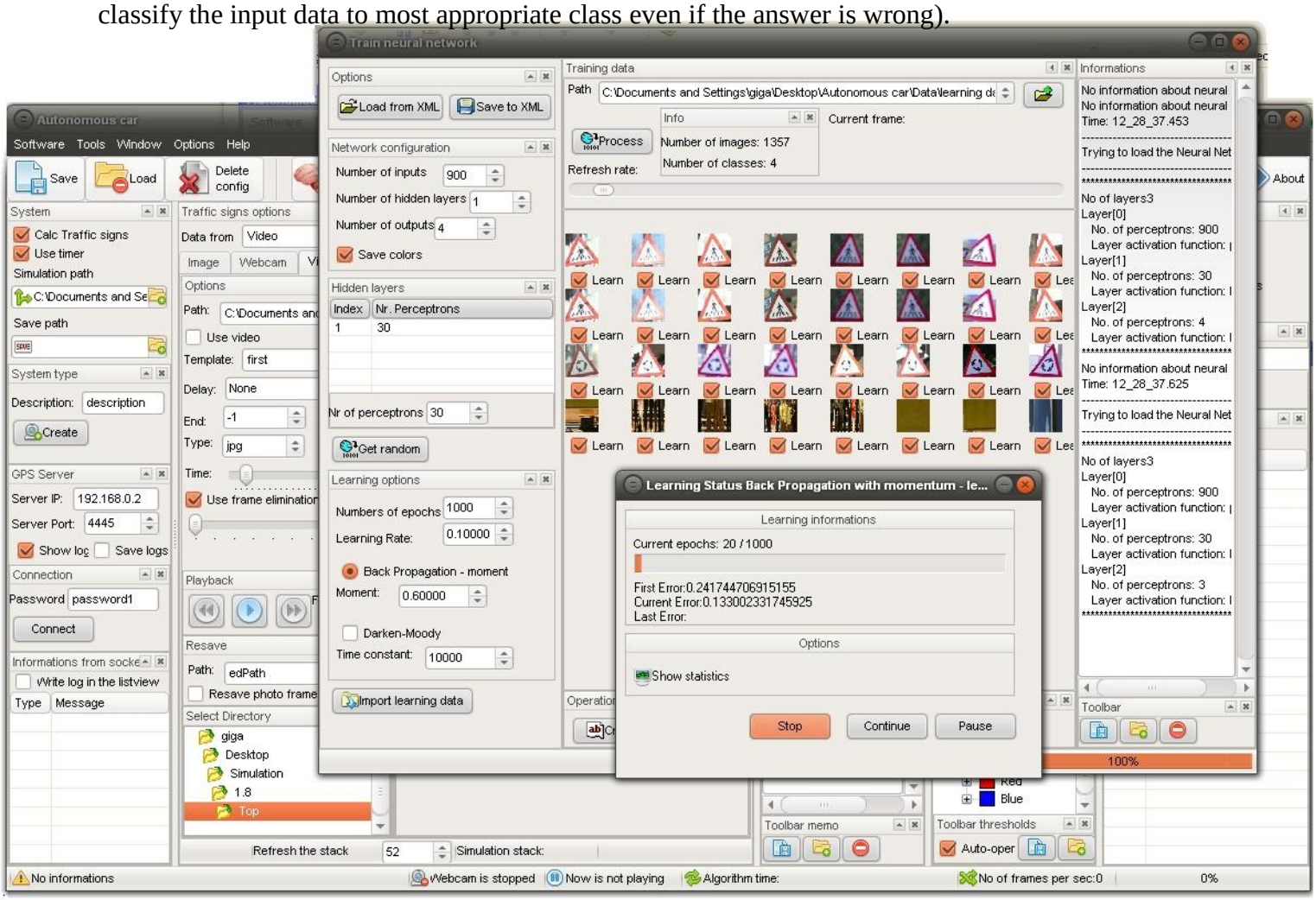
$$\boldsymbol{b}^m(k+1) = ()\boldsymbol{b}^m(k-1) - (1-()\,\forall \boldsymbol{s}^m$$

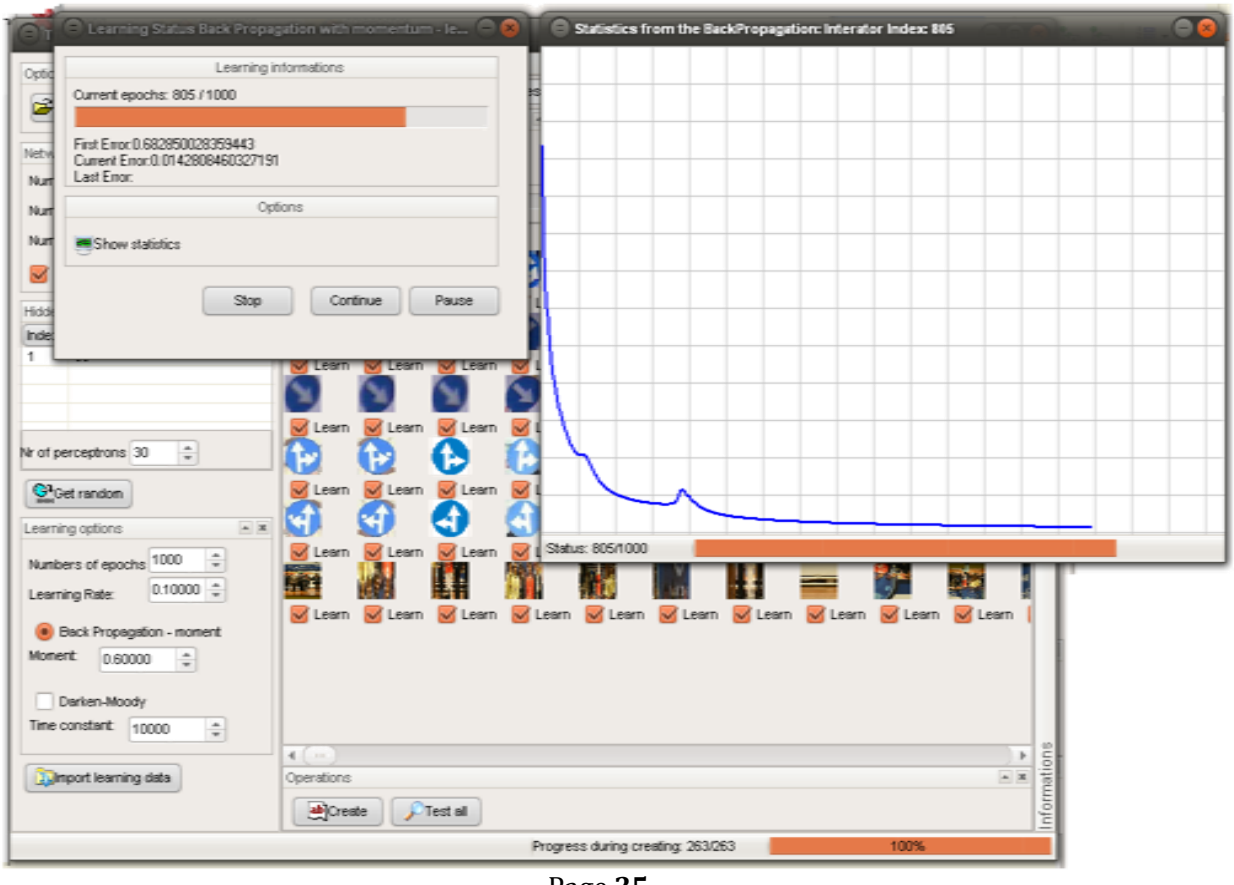
- **Darken Moody** – it allows the chaning value of the learning rate(∀). The learning rate chaning is calculated using the variable learning rate 𝔉:

$$\alpha(k) = \frac{\alpha_0}{1 + \frac{k}{\tau}}$$

Neural methods can be applied successfully in software with incomplete data, erroneous values around at some values called clusters. The networks are trained with some representatives of the cluster and on the simulation time the networks will be assigned with near values from the cluster. The method can be used for detecting other objects from the image for instance pedestrians or cars. To recognize only the blue traffic signs forms I used a [900, 40, 6] configuration, so the input layer was formed by all image pixels(30x30). On the hidden layer I used 40 perceptrons -I tested it with various types but the highest results were obtained with 40 perceptrons . The output layer is formed of 6 perceptrons (first 5 classes are different blue traffic signs forms, the last class is used for rejection), because MLP doesn't manage to make the rejection automatically(the MLP

classify the input data to most appropriate class even if the answer is wrong).





Statistics about the recording algorithm.

The software components were designed to be able to save offline images for further simulation. For this purpose, the project saves the simulation data for traffic signs and traffic lanes detection using multiple computers and parallel programming. There are three webcams and two different laptops. One computer saves the images from a top view position with a resolution of 800x600 and the other laptop saves the images from the other two images which have the resolution of 352x288.

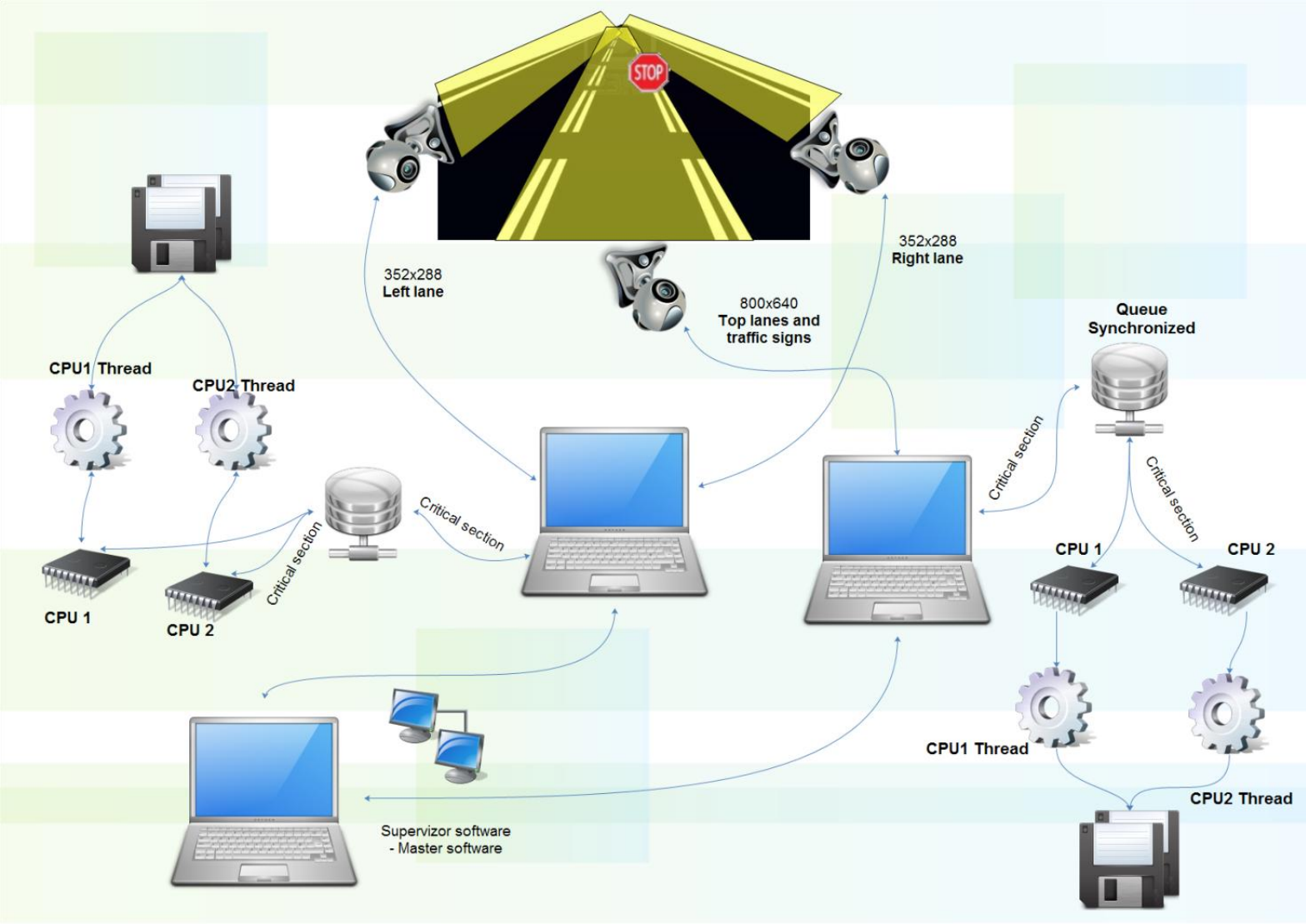


Image no. 2.1. This diagram shows how the software applications were designed in order to save the images using distributed programming and parallel programming.

No of frames recorded	Frames per second	Frame resolution	No of cores
12,343	~30	800x600	**2**
9,432	~29	800x600	**2**
10,323	~29	800x600	**2**
11,123	~30	800x600	**2**
8,145	~18	960x800	**2**

Table no. 1.2. This table shows how many different traffic signs can be recognized by the cascaded

Neural Networks from the software.

Statistics about the traffic signs recognition algorithm.

In order to improve the algorithms performances, each time I made major improvements to the algorithm and I tested the algorithms with multiple testing data. The offline simulations consisted of images coming from the three different webcams and the GPS coordinates. The below table displays the statistics of the traffic signs detection algorithm.

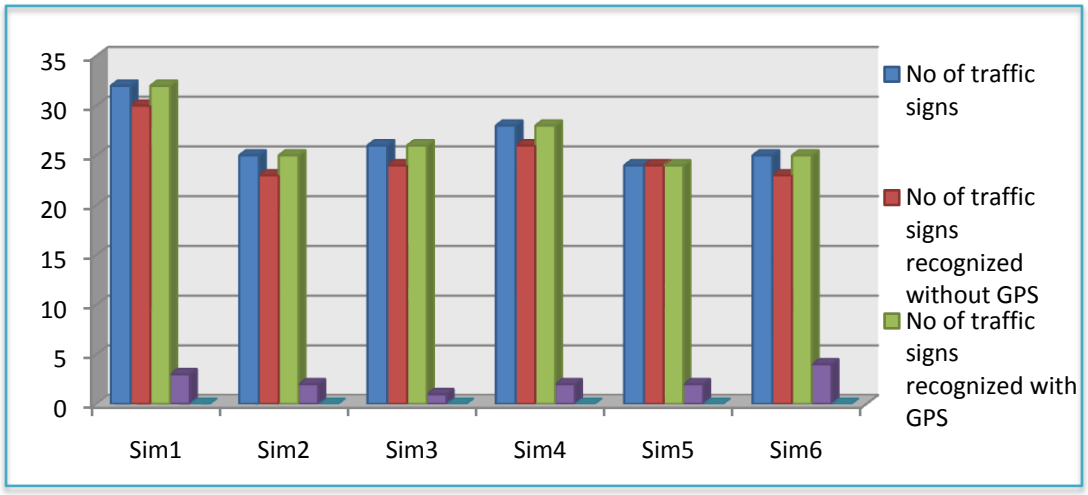
No of frames in the simulation	The lightness	Frames per second	Frames processed per second	No of **cores**	No of traffic signs in the simulation data	**No of traffic signs recognized**
12,343	Good	~30	~29	2	**32**	**31**
9,432	Good	~29	~29	2	**25**	**24**
10,323	Good	~29	~29	2	**26**	**25**
11,123	Good	~30	~29	2	**28**	**26**
8,145	Good	~18	18	2	**24**	**24**

Table no. 1.3. This table shows how many different traffic signs can be recognized by the cascaded Neural Networks from the software and the error recognition.

No of traffic signs in the simulation data	No of traffic signs recognized	**Percent of recognition**	No. of frames recognized from multiple frames	Percent of recognition from multiple frames
32	30	**93.75%**	26	81.25%
25	23	**92%**	21	84%
26	24	**92.30%**	22	84.61%
28	26	**92.85%**	24	85.71
24	24	**95.83%**	20	83.33

Table no. 1.4. This table shows the percent of detection by the software in order to recognize the traffic signs. Moreover, this table shows the percent of traffic signs recognized from multiple frames.

The use the GPS Solution can improve the traffic signs detection to over 99,5%.



Threading and multi-threading code

For reaching the maximum performance from the computers, the software components were designed using **parallel programming.** Traffic Signs Detection algorithm, Traffic Lanes Detection algorithm and Data Save algorithm (images from webcams and the GPS data) were built using

multi-threading solutions.

Electronic parts

This project presents a hardware version of a LIDAR – a 3D radar and a software for creating a 3D environment in which the car navigates. By using it, the car will take the decision to avoid obstacles. The 3D radar helps the entire software system to increase the confidence of decision. A stepper motor is used to spin the PCB with the photodiodes, alimentation and a PIC16F877. A photodiode is a type of photodetector capable of converting light into either current or voltage. My photodiode is a very special one and is used only in rangefinder; it is an **APD – Avalanche Photodiode Detector –** source**at200 V.**

No of data received: No of APDs * No of spins per second * No. of spin grades * No. bits

16 x 10Hz * 180 grades(or 360 or 18000) * 10 bits = 288,000 bits of data/s

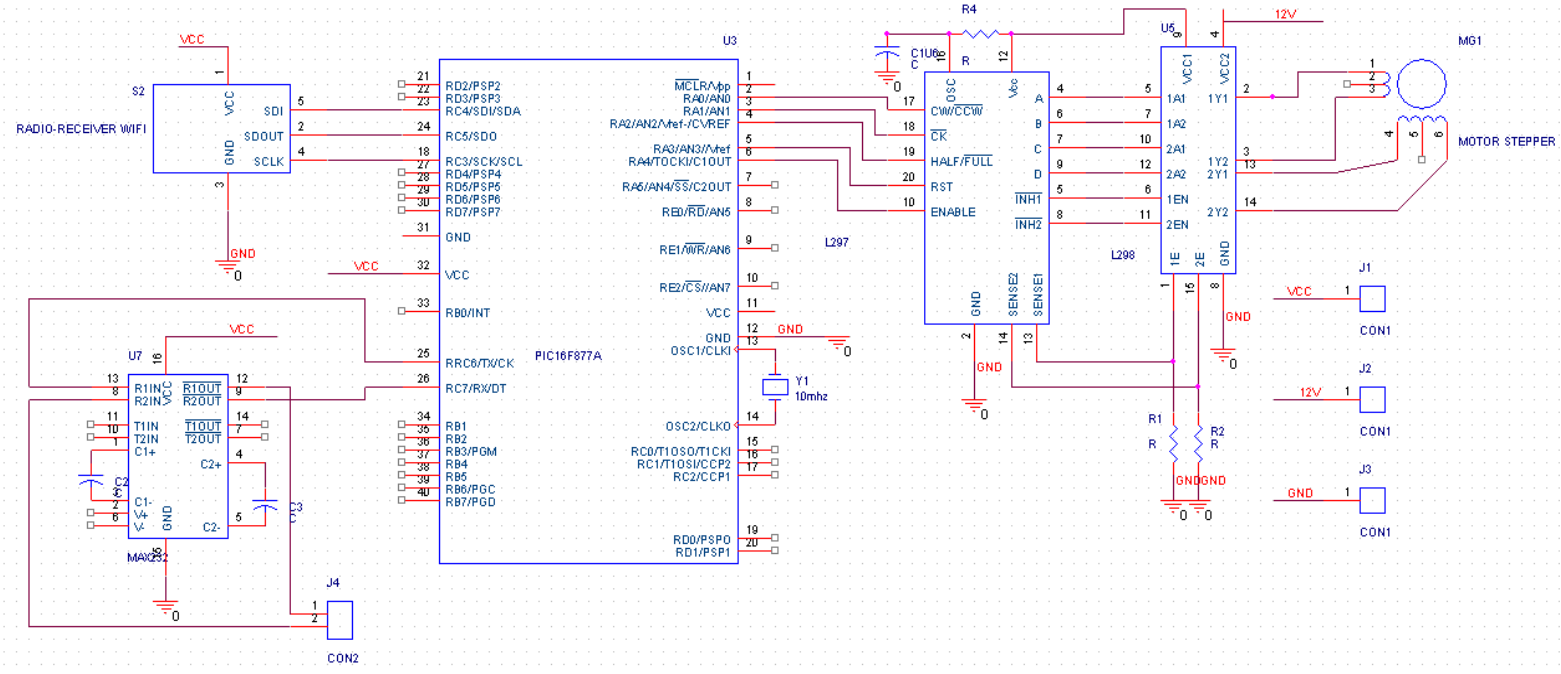
=28.800 pixels with the resolution of 10 bits

There are only 6 electronic parts:

1) Base microcontroller, driving stepper – LIDAR, interface for RS232
2) LIDAR microcontroller – fast convertor analogical digital SPI, wireless transmitter
3) Convert the ToF(Time of Fly) into an analogical signal.
4) Clock generator – 4 kHz, 0.001%, length of duration ~ 40 ns
5) Pulsed laser generator, 35A/40-60 ns
6) APD photodiode, comparator and high speed A stable

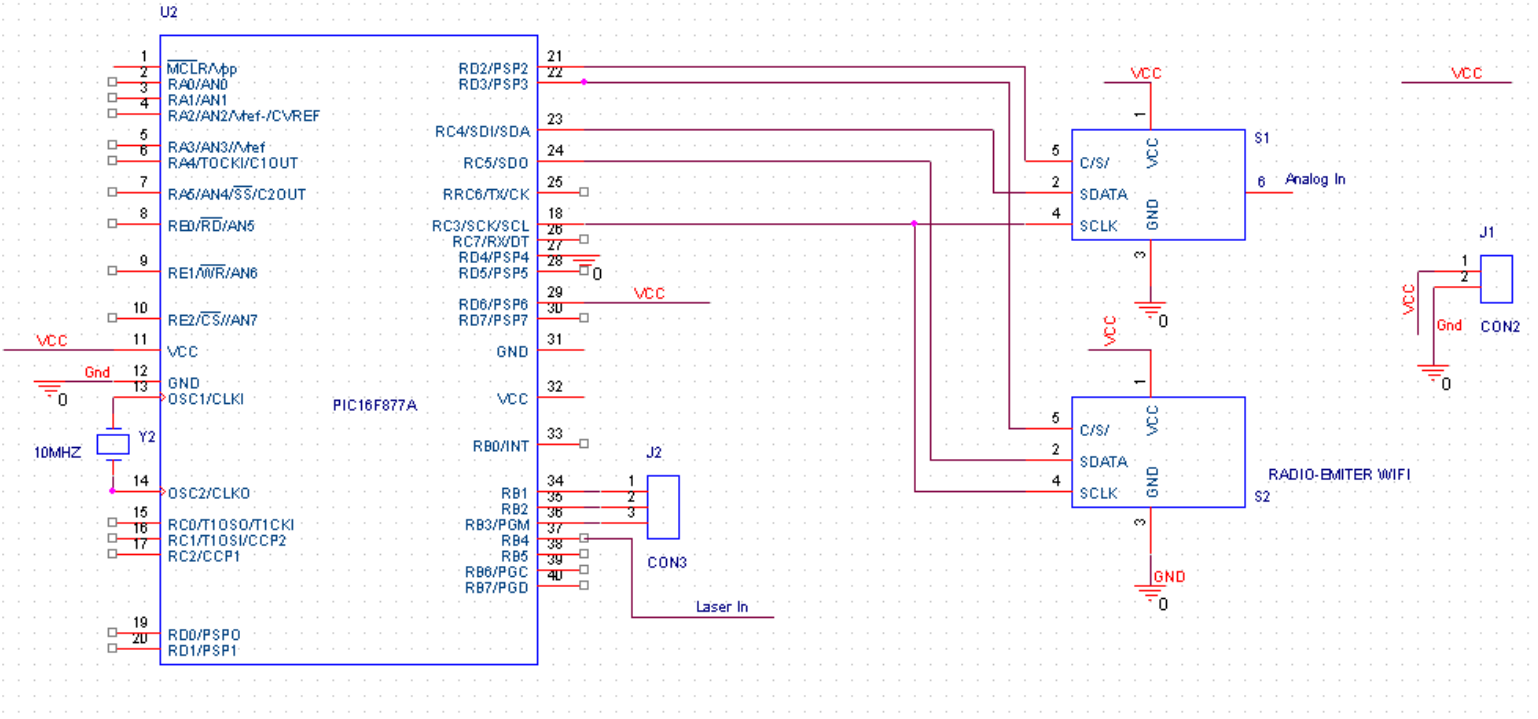
[1]. Base microcontroller, driving stepper – LIDAR, interface for RS232

This diagram shows the computer interface and stepper motor control. The electronic circuit diagram contains a receiver that collects data from the rotating LIDAR. The transmission is via a 2-GHz transceiver. After receiving the information, the printed circuit board sends the data to the laptops via the serial RS232 port, though the circuit of MAX232 . This circuit provides impulses for the next step of the LIDAR.The rotation speed is 20 Hz (20 pulses per second), the motors allow 1.8 degrees per pulse. A complete turn means 200 impulses. So the generator will generate 4000 pulses per second.



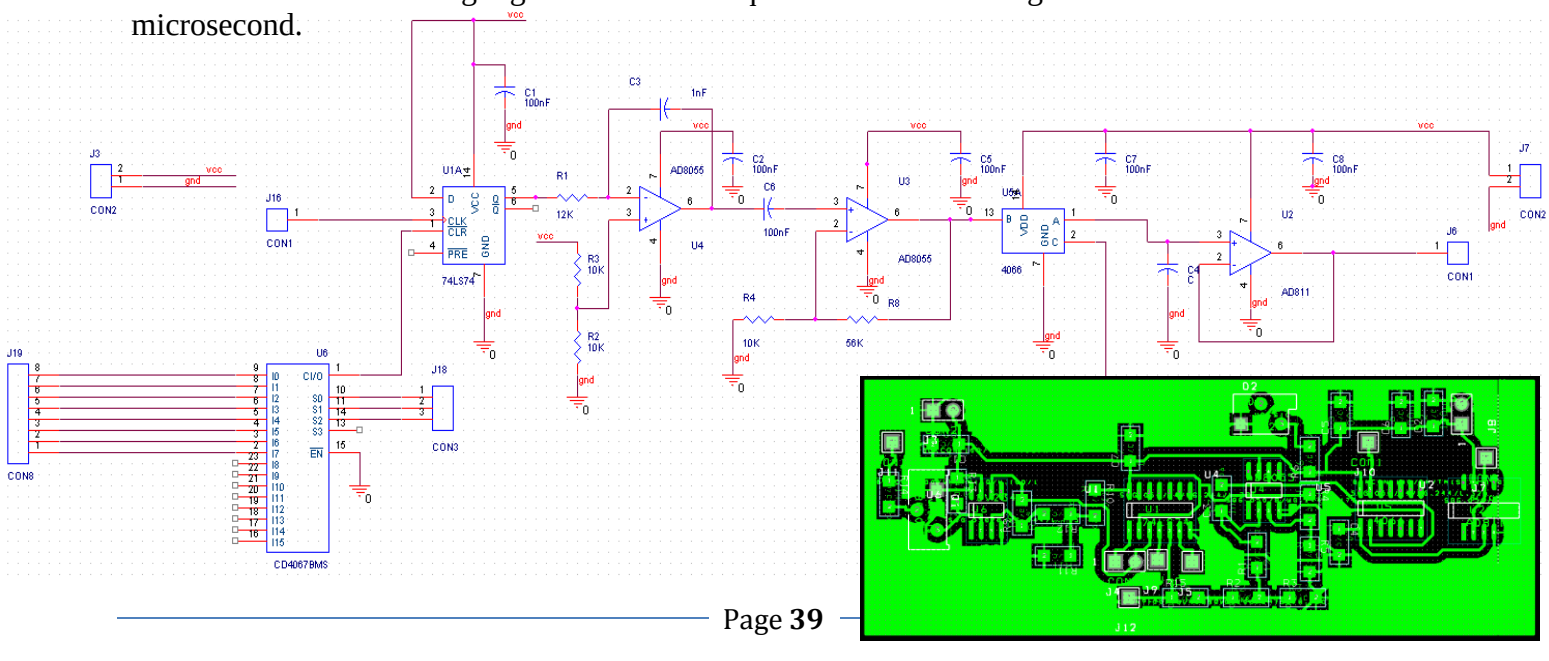
[2]. LIDAR microcontroller – fast convertor analogic digital SPI, wireless transmitter

This controller is placed on the rotating radar and provides analog-numerical conversion of the Time to Fly signal, along with switching signals of the four photodetectors cells and lasers beams.The acquired signal is transmitted to the PIC microcontroller from the stable board (that with the stepper motor) through a transceiver. The data and the information of the photodiode cell is transmitted at a resolution of 10 bits. The converter has a resolution of only 10 bits transmitted though SPI (Serial Peripheral Interface Bus).



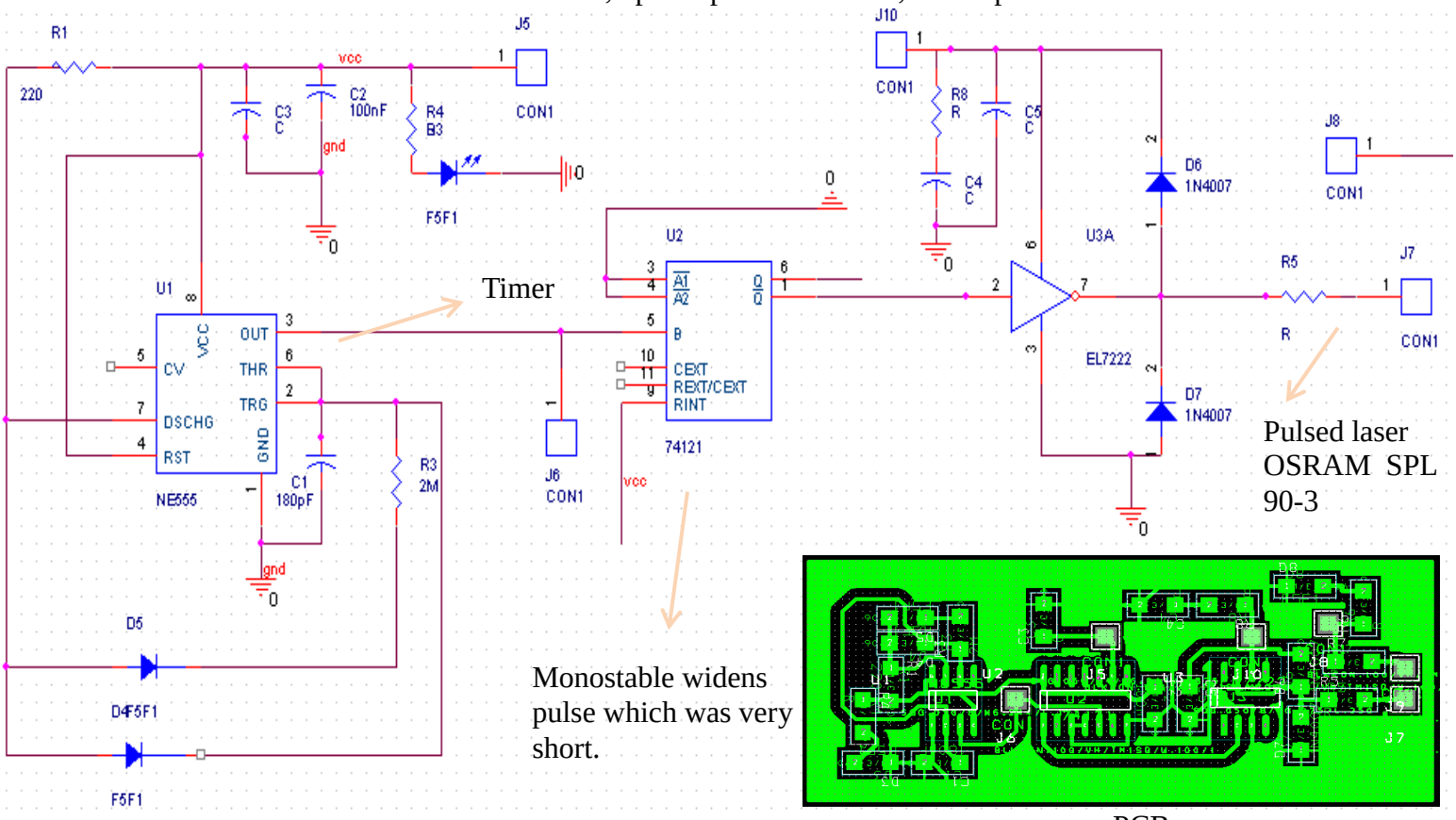
[3]. Convert the ToF(Time of Fly) into an analogic signal.

The electronic circuit calculates the time difference between the pulse and reception of the reflected signal from the target. For this I used a D flip-flop. The resulting signal is integrated with a high-frequency integrator, then is applied to a sample and hold circuit created from a switch CD4067 and a repeater. The circuit returns a voltage proportional to the pulse duration. I used this method because the analog digital converters require conversion voltage to be stable at least for one microsecond.



[4]. Clock generator – 4 kHz, 0.001%, length of duration ~ 40 ns

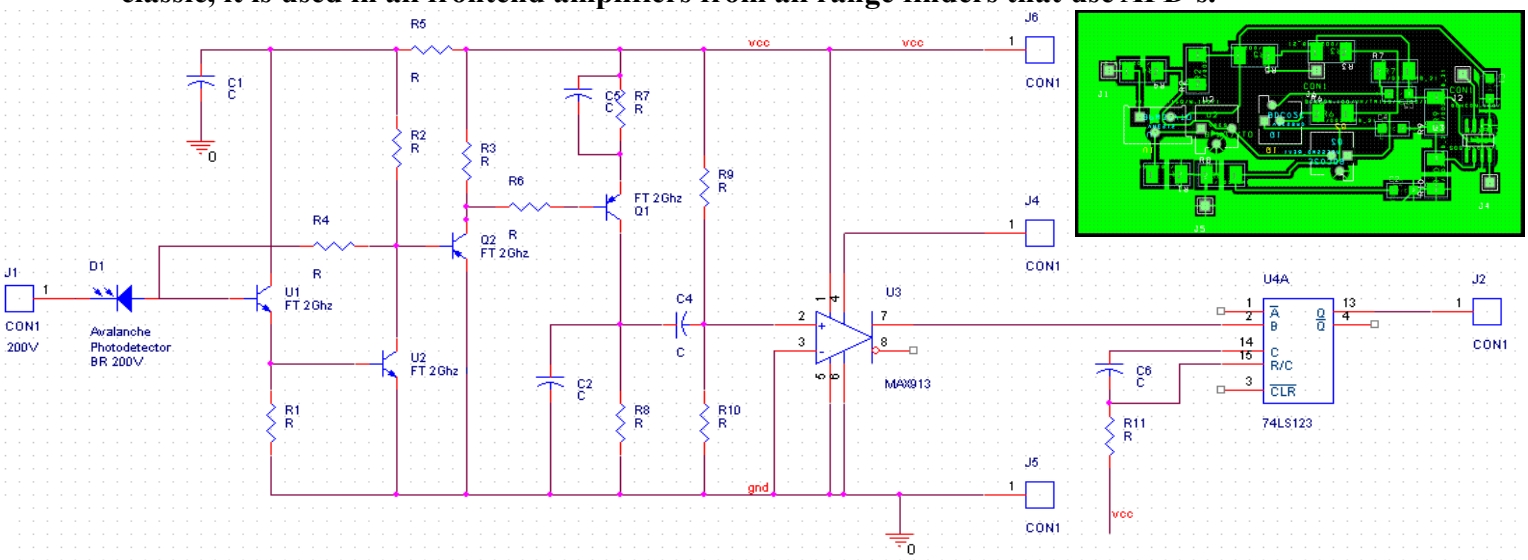
Laser pulse transmission method is called Pulsed Laser and involves the use of large currents in the tens of amperes of short duration, 40-60 nanoseconds, with a maximum repetition rate of 3 kHz. Ensuring optical power level of 70 Wats. Energy is very high but because the pulse duration is very short, the reflected signal is much higher than the ambient light. The latest version uses an infrared laser OSRAM SPL 90-3, optical power 75 Watts, 35 Amperes.



PCB

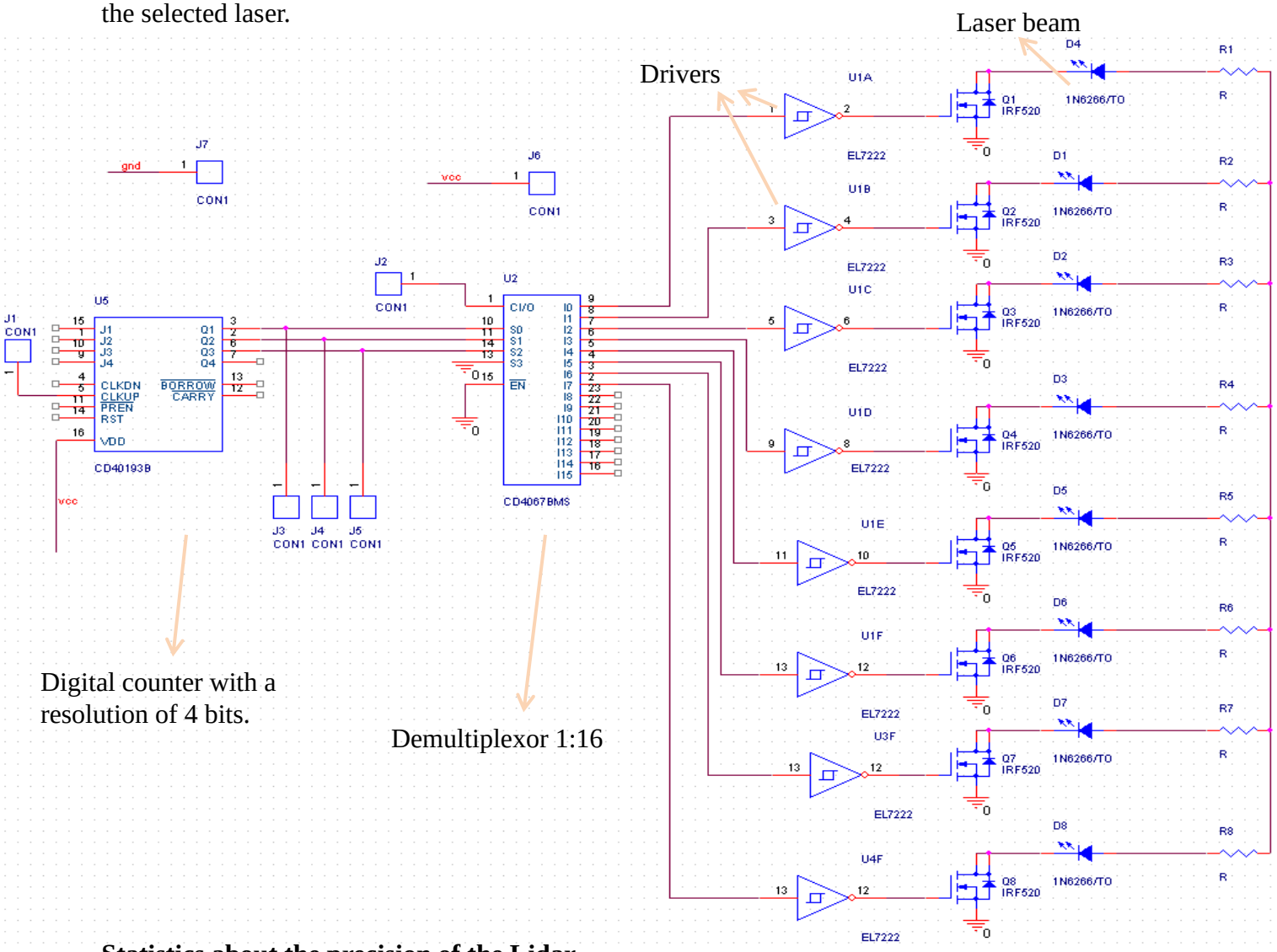
[5]. APD photodiode, comparator and high speed monostable

The circuit uses an APD (Avalanche Photodiodes) diode which converts the reflected light into an electrical impulse. The amplifier used by me is called transimpedance amplifier and amplifies this signal to a TTL (Transistor-Transistor Logic) level. The signal is compared by a threshold comparator which decides if the signal was reflected from the target. **The scheme is classic, it is used in all frontend amplifiers from all range finders that use APD's.**



[6]. Pulsed laser generator, 35A/40-50 ns

The electronic circuit ensures the selection of all 8 lasers using a digital counter CD4067 and a demultiplexer CD40193B. Followed by some power drivers, and a MOS transistor that acts the selected laser.

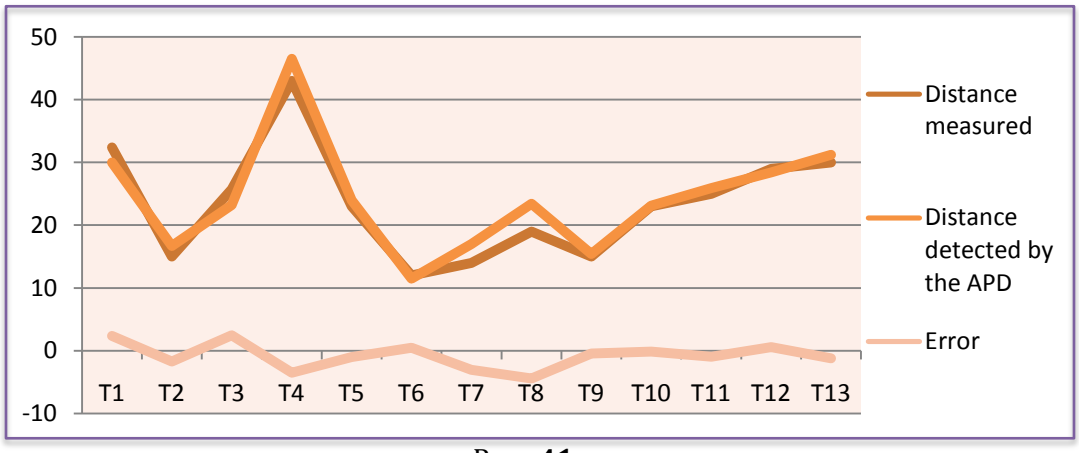


Statistics about the precision of the Lidar

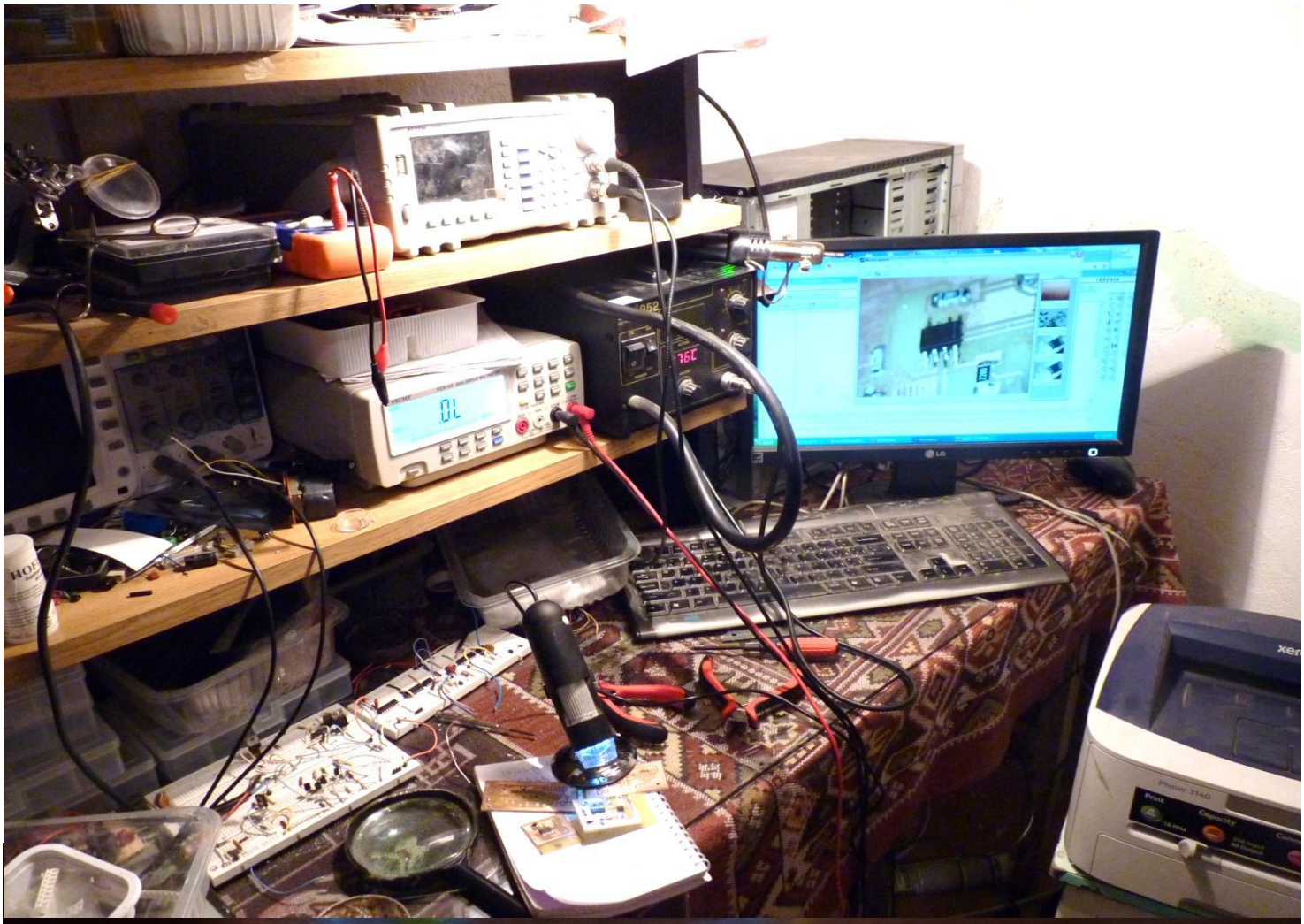
The precision is on 10 bits, so $2^{10} = 1024$

One step is $\frac{80}{1024} = 0.078125\ meters = 7.8125\ cm$

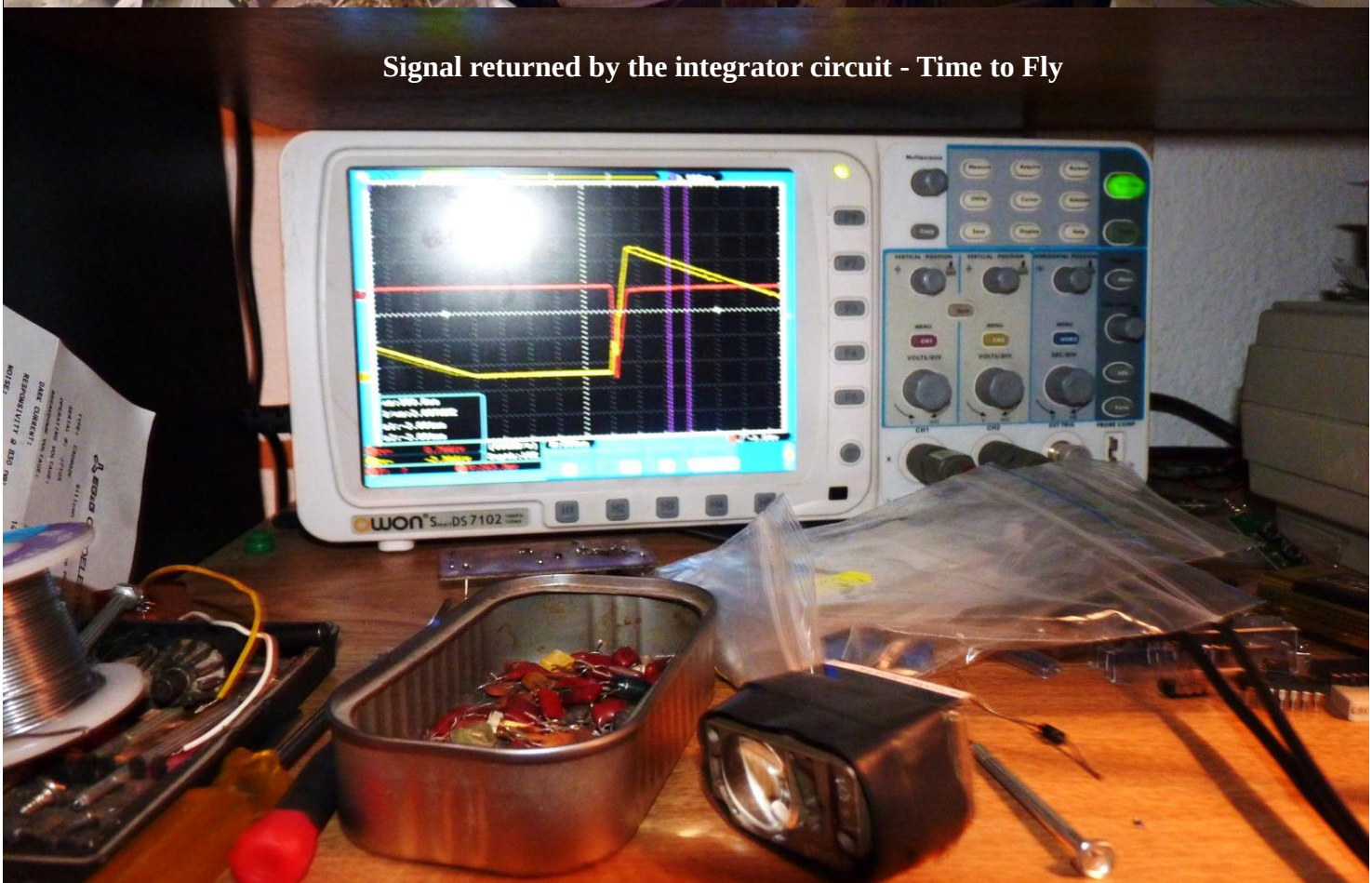
The error between two consecutive steps: $0.078125\ meters = 7.8125\ cm$

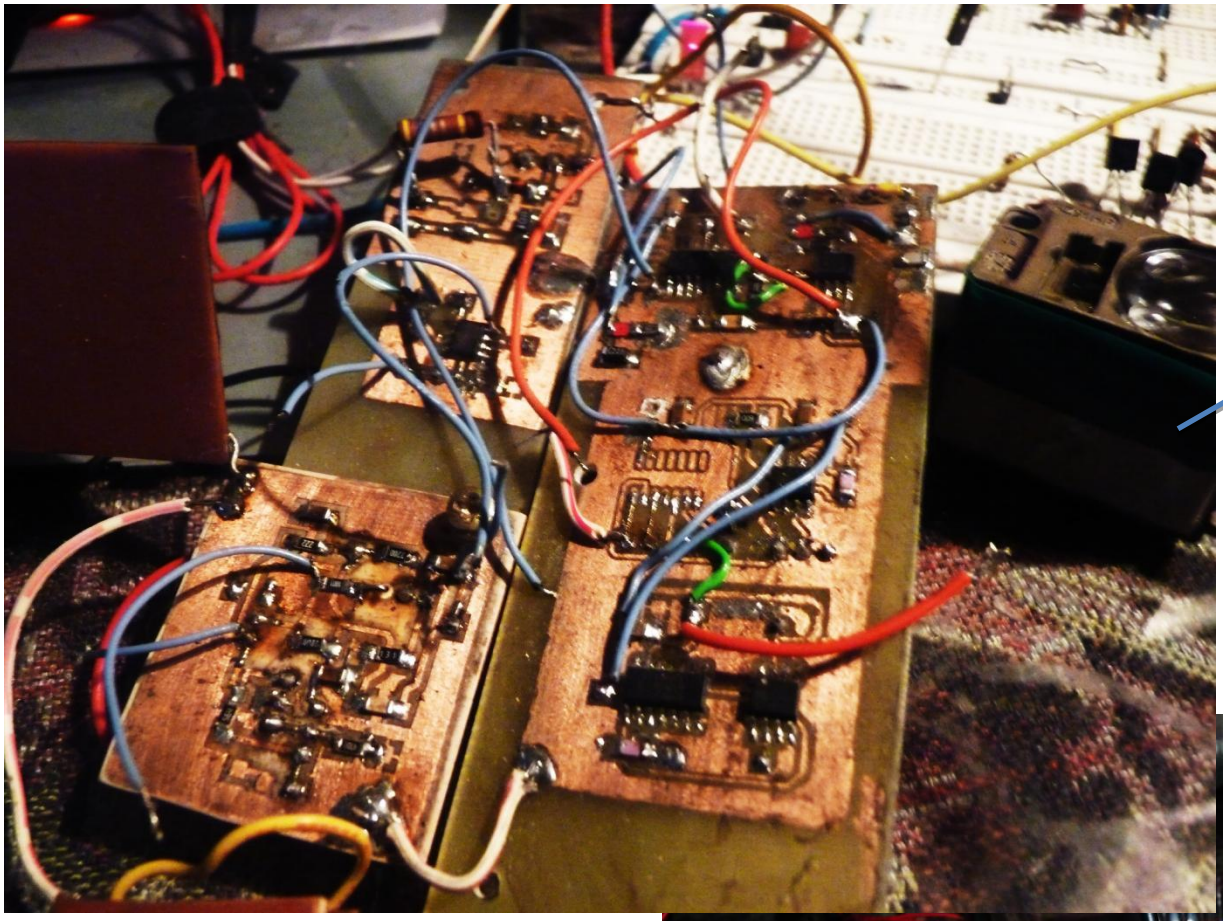



Intel
ISEF
2011
intel
ISEF
2010

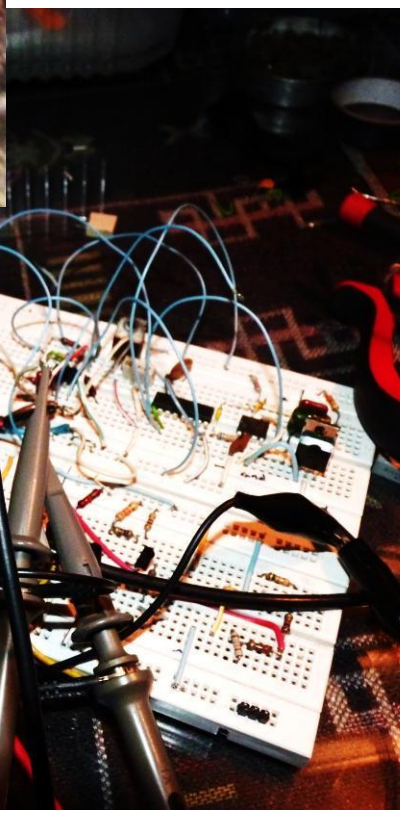


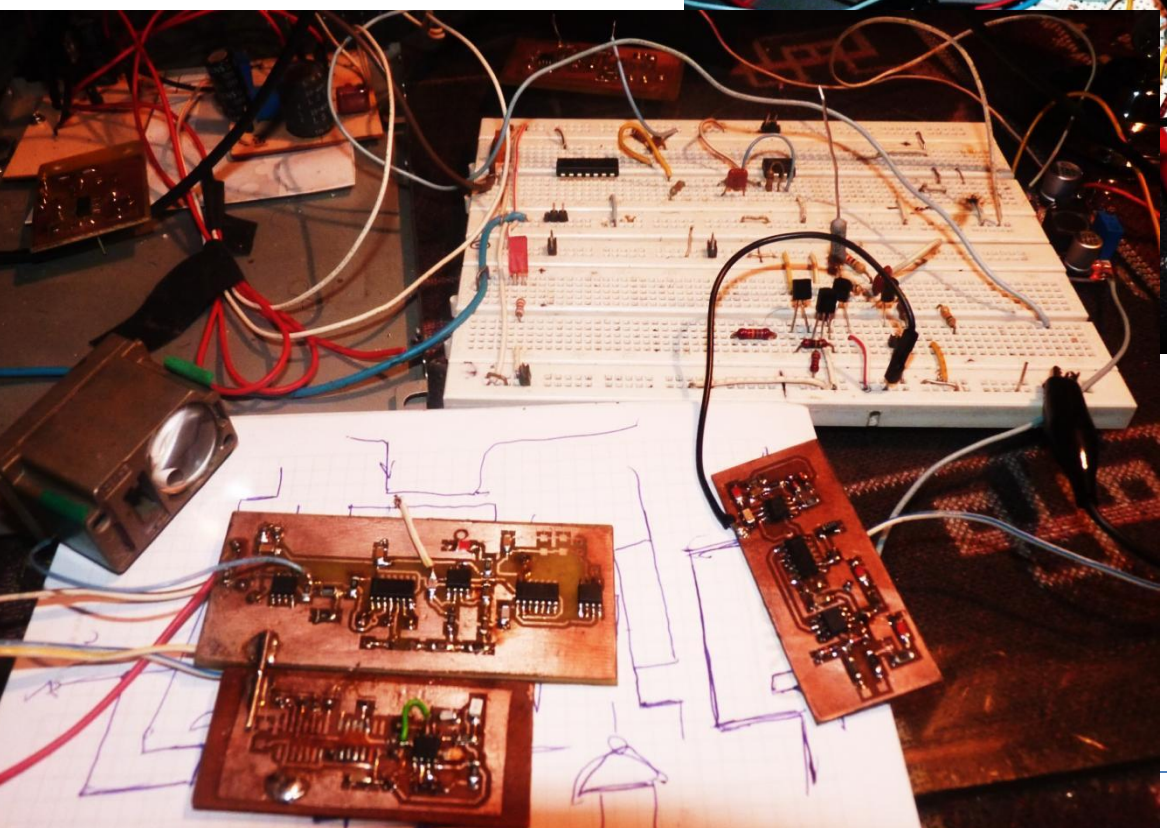
Signal returned by the integrator circuit - Time to Fly





Avalanche photodiode





Project status

This project was built by creating different software components and electronics. The next step was connecting all of the project's parts one to each other. Most of the components have already been done. Here's is a simple list of the components involved in the project.

- **Traffic signs detection** **90%**
  - **Multi-threading algorithm** for detection
  - **HSV Filter thresholds** (searching for better thresholds)
  - Offline simulations
  - Real time simulations
  - Communication with the **GPS and Google Maps Software** component
    - Creating the **Common Traffic Signs Database**
      - Giving **feedbacks** for the Common Traffic Signs Database
    - **Use** the Common Traffic Signs Database
  - Communication with the Supervisor Software using a local network
- **Traffic lanes detection** **90%**
  - **Multi-threading algorithm** for lane detections
  - **Multiple thresholds**
  - Offline simulations
  - Real time simulations
  - Communication with the Supervisor Software using a local network
- **GPS and Google Maps Software** **90%**
  - Reading the data from **GPS**
  - **Mashup** using the **Google Maps API v3.0**
  - Integrating the **GPS with Google Maps**
  - Offline simulations
  - Real time simulations
  - Communication with the **Traffic Signs Detection Software** component
    - Creating the **Common Traffic Signs Database**
      - Giving **feedbacks** for the Common Traffic Signs Database
    - **Send and use**the Common Traffic Signs Database
  - Communication with the Supervisor Software using a local network
- **3D Radar – LIDAR8 APD – 10 Hz**
  - **APD – Avalanche Photodiode Detector –** source **200 V**
  - **APD – Trans Impedance – Amplifier**
  - Timer generator
  - **Laser pulse circuit**
  - Circuit for calculus for **Time to Fly(TOF)**
  - **Conversion integrator**
  - **Amplifier**
  - **Sample and Hold**
  - **Circuit PIC16F877**
    - **Transmission**
    - **Reception**
    - **Circuit driver stepper motor**

- **3D Radar receiver Software**
  - Reading the data from **PIC16F877** using **COM Port** or **RS232**
  - **OpenGL**
    - Create a **3D world from the 3D data**
    - Save the data for further offline simulations using timestamps
  - Offline simulation
  - Real time simulations
  - Communication with the Supervisor Software
- **Supervisor software 70%**
  - Communications with the all other software components
  - Control all other software components **50%**
  - Communication with **PIC16F877**
  - Making a decision and send the **steering movements** to **PIC16F877**
  - Offline simulations 70%
  - Real Time simulations

In the below table you can see the project's current status - what project's components were done and some information about the finished components.

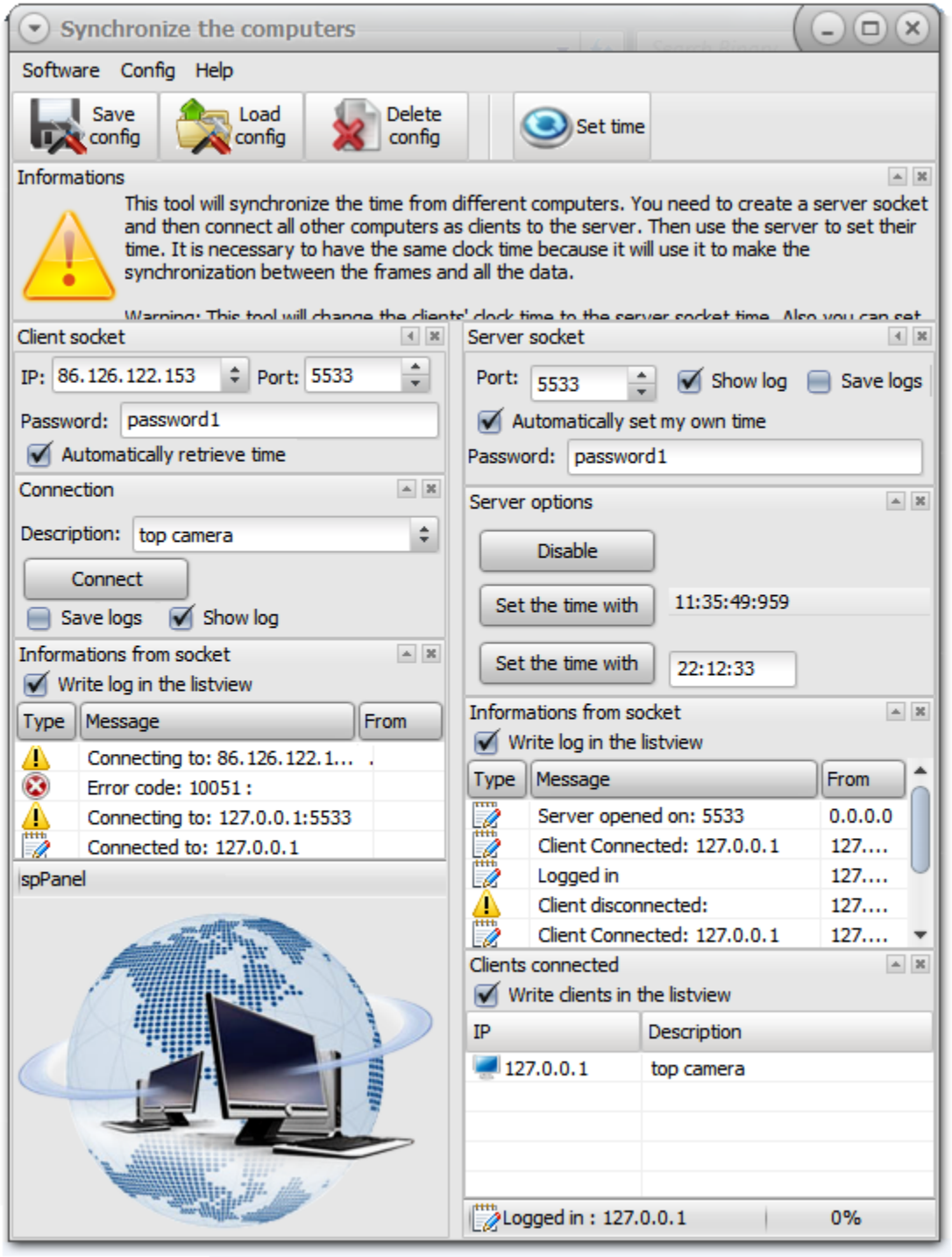
Project's component name	Component type: software/electronic	Percent of finalization	**Mentions**
Traffic signs detection using Cascaded Neural Networks	Software	90%	**Multi-threading algorithm**, communication with the supervisor software, offline simulations, real time simulations, communication with the GPS and Google Maps Software component
Traffic lanes detection using Particle Filters – Kalman Filters	Software	90%	**Multi-threading algorithm,** display the traffic lanes, communication with the supervisor software, send the traffic lanes to the master software, offline simulations, real time simulations,
GPS and Google Maps Software	Software	90%	**Reading the data from the GPS, using the GPS coordinates in the Google Maps,**offline simulations, real time simulations, communication with the Traffic Signs Detection software and the supervisor software.
Supervisor software	Software	70%	Collecting the data from all software components, collecting statistics about computers' performances(CPU and Memory Usage), display the traffic signs and the traffic lanes.
Communication between the Traffic Signs detection and GPS Software	Software	90%	The software is able to create the Traffic Signs common database using the traffic signs and the GPS coordinates. Each time the software gives positive and negative feedbacks to the traffic signs from the common database.
3D Radar - LIDAR	Electronic device	75%	**Schematic** and**Printed Circuit Boards** for the LIDAR in the **Cadence OrCAD**.The photodiode was tested and it works.
OpenGL to create the 3D map from LIDAR	Software	90%	The software can read the 3D data and create a 3D map about the environment. It can save data and continue to use it in offline simulations.

Table no. 1.4. This table shows the percent for each component of the project. The field called

"mention" is used to describe the parts done for each component.

Synchronization for distributed calculus

The project offers the possibility to make offline simulations in order to understand the car's performance and to make the necessary improvements. To do this offline simulation it was necessary to save the entire data using timestamps. The data file is very complex and is formed of many webcam images and the GPS data. Because there are many different software components running on different computers(laptops) with different performance, each software must synchronize with all others software components.



Each computer connects through this protocol in order to make the synchronization between all computers. The synchronization is made using the same time clock.

Image no. 2.5. This is the software that makes the synchronization between the computers. All computers connect to the supervisor software and set the computer's clock using the supervisor

time.

Contact

- **BudisteanuIonutAlexandru:**
  - Email address: ibudisteanu@acm.org**or** budisteanu@giga-software.com
  - More details about this project here website**: www.giga-software.com**
  - Personal website:www.budisteanu.net

Bibliography:

[1] ACM Digital Library

[2] Digital Integrat Circuits, Gheorghe Stefan, Polytechnic Romania

[3] Digital Integrat Circuits, SandaMaican, Polytechnic Romania

[4] ACM Digital Library

[5] Cadence OrCAD – editing different schemes

[6] Cadence OrCAD realization of Printed Circuit Boards

[7] Pspace Manual Cadence

[8] From TTL gate to microprocessor, Gheorghe Toacse, West University Romania

[9] Neural Networks - Răzvan Andonie

[10] Neural Networks – Applications Catalin-Daniel Caleanusi Virgil Tiponut.

[11] Reteleneuronale - Arhitecturisialgoritmi de Stefan Holban

[12] Neural network training based on examples, CalinEnachescu

[13] Solving different kind of problems using feed forward neural network Daniela Zaharie West University of Timisoara

[14] Knowledge representation, NicolaeTandareanu, University of Craiova

[15] Image processing and analyzing - ConstantinVertan, University of Bucharest

[16] Artificial Intelligence, DoinaZaharia, West University of Timisoara

[17] Mathworks MATLAB documentation

[18] NeurosLab documentation www.giga-software.com, IonutAlexandruBudisteanu

[19] AILab – scripting language for Artificial Intelligence documentation, www.giga-software.com, BudisteanuIonutAlexandru

[20] Artificial Intelligence, Dan Rotar – University of Bacau

[21] Systems based on knowledge, Doru Adrian Pãnescu, University AlexandruIoanCuza Iasi

[22] Artificial Intelligence, IoanDziţac, University AurelVlaicu

[23] Artificial Intelligence, CatalinStoian, University of Craiova

[24] Knowledge database and expert systems, NicolaeȚăndăreanu

[25] Genetic algorithms, G. Oltean, Tehnical University of ClujNapoca

[26] Rule-based tracking of multiple lanes using particle filters, Stefan Vacek, Stephan Bergmann, Ulrich Mohr, and RuudigerDillmann. IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems

[27] Neural and evolutionarz computing, D. Zaharia, West Universitz of Timisoara

[28] Machine learning, Daniel Morariu, Lucian Blaga University, Sibiu

[29] Artificial Intelligence, University of Berekeley

[30] Introduction to Artificial Intelligence,PallabDasgupta, Indian Institute of Technology Kharagpur

[31] Road-marking analysis for autonomous vehicle guidance, Stefan Vacek, ConstantinSchimmel, RudigerDillmann,

[32] Sequential Monte Carlo Methods in Practice. Springer.,Doucet, A.; De Freitas, N.; Gordon, N.J. (2001).

[33] A lane detection vision module for driver assistance, KistijanMacek, Brian Williams, SaschaKolski, Roland Siegwart, IEEE trans. Pattern Analysis and Machine Intelligence

[34] Artificial Intelligence: A Modern Approach, Stuart Russell and Peter Norvig, Berkeley Univ.

[35] Neural Network http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html
[36] Stuttgart Neural Network Simulator - introduction http://www.ra.cs.uni-tuebingen.de/SNNS/
[37] Neural Network Toolbox from Matworks MATLAB http://www.image.ece.ntua.gr/courses_static/nn/matlab/nnet.pdf
[38] An introduction to neural network http://www.ibm.com/developerworks/library/l-neural/
[39] Machine learning course http://www.ml-class.org/course/auth/welcome
[40] Introducing to Artificial Intelligence http://www.ai-class.org/PhD. Sebastian Thrun and PhD. Peter Norvig, Stanford University
[41] Nicholas Apostoloff. Vision based lane tracking using multiple cues and particle filtering. PhD thesis, Department of Systems Engineering, Research School of Information Science and Engineering, Australian National University
[42] www.LaserComponents.com – Article: "Avalanche Photodiodes"
[43] Technical Information SD-28 Avalanche Photodiodes , HAMAMATSU
[44] Silicon Avalanche Photodiodes, A. Stoykov&R.Scheuermann, Laboratory for Muon Spin Spectroscopy, Paul ScherrerInstitut, CH-5232 Villigen PSI, Switzerland
[45] R.J.McIntyre, IEEE Trans. Electron Devices, ED-19, 703
[46] Sensor Solutions, Datasheet, Silicon Avalanche Photodiodes, C30902 Series, High Speed APDs for Analytical and Biomedical Lowest Light Detection Applications
[47] UdacityOnline free classes - http://www.udacity.com/PhD. Sebastian Thrun
[48] Avalanche Diodes (APD) Detectors: Introduction and Recent Advances , Alfred Q. R. Baron, Harima Riken, ,International Symposium on the Development of Detectors for Particle, Astro-Particle, and Synchrotron Radiation Experiments "SNIC"
[49] Characterization of silicon avalanche photodiodes for photon correlation measurements, Robert G. W. Brown, Robin Jones, John G. Rariry, and Kevin D. Ridley,
[50] LaserComponents.com Pulsed Laser Diodes Avalanche Photodiodes
[51] An overview of avalanche photodiodes and pulsed lasers as they are used in 3D laser radar type applications, Bruno Dion CMC Electronics, Nick Bertone OEC,
[52] Three-Dimensional Imaging Laser Radars with Geiger-Mode Avalanche Photodiode Arrays , Marius A. Albota, Brian F. Aull, Daniel G. Fouche, Richard M. Heinrichs, Lincoln Laboratory MIT,
[53] *Pulsed Time of Flight laser range finder techniques for fast, high precision measurement applications,* Ari Kilpela, Department of Electrical and Information Engineering, University of Oulu
[54] Laser Fluorometry Using an Avalanche Photodiode a Visible Semiconductor Laser for Capillary Electrophoresis , Hirofumi KamazuMI'J, oonMyongsOng,Takanori Inoue and Teiichiro Ogawa, Kyushu University,
[55] The Capacity of Avalanche Photodiode-Detected Pulse-Position Modulation , J. Hamkins
[56] For the datasheets of my chips, http://www.alldatasheet.com/
[57] "WHO | World report on road traffic injury prevention" and "WHO Global Status Report on Road Safety" from WHO onwww.who.int.
[58] Near-Infrared Laser Range Finder, using kHz Repetition Rate , Josef Kölbla, Michael Fröschla, Adam Seedsmana, Peter Sperberb, EOS Optronics GmbH, University of Applied Sciences,
[59] *Range Finding Using Pulse Lasers*, Opto Semiconductors, OSRAM Inc.
[60] "The USA 2009 Statistical Abstract: Motor Vehicle Accidents and Fatalities". http://www.census.gov/compendia/statab/cats/transportation/motor_vehicle_accidents_and_fatalities.html