A SIMULATION ENVIRONMENT FOR AUTOMATIC NIGHT DRIVING AND VISUAL CONTROL

Author: Fernando Arroyo Rubio
Supervisor and Examiner: Giacomo Spampinato

Bachelor Thesis, Computer Science 15 credits
April 13, 2012

School of Innovation, Design and Engineering
Mälardalen University, Västerås, Sweden
TABLE OF CONTENTS

Abstract.......................................................................................................................... 3
Installation Process ........................................................................................................ 4
Track Creation ............................................................................................................... 8
  Light-Markers Creation ............................................................................................ 15
  Fov Factor ................................................................................................................. 16
Visual Control and Automatic Driving System (ADS) ............................................. 17
  The Driver (Robot) .................................................................................................... 17
  The Track and Visual Control .................................................................................. 18
  The Car ...................................................................................................................... 18
  Light-Markers Detection Process ............................................................................ 19
Libraries ....................................................................................................................... 21
Code and Algorithms ................................................................................................... 22
  Declaration and Initializations ................................................................................ 23
  First Phase: Capturing the Screen .......................................................................... 24
  Second Phase: ROI .................................................................................................. 26
  Third Phase: Detecting the light-markers .............................................................. 26
  Fourth Phase: Sending Coordinates to Driver ...................................................... 30
  Fifth Phase: Steering ............................................................................................... 31
Simulation ...................................................................................................................... 33
References .................................................................................................................... 35
ABSTRACT

This project consists of developing an automatic night driving system in a simulation environment. The simulator I have used is TORCS. TORCS is an Open Source car racing simulator written in C++. It is used as an ordinary car racing game, as an IA racing game, and as a research platform. As we know, in a car racing game the vehicle can be driven using either a joystick or a keyboard. Nevertheless, TORCS has been designed to enable pre-programmed IA drivers to race against one another.

However, this thesis is not focused on competing against someone else as in a racing competition. The goal of this thesis is to implement an automatic driving system to control the car under night conditions using computer vision. A camera is implemented inside the vehicle and it will detect the reflective light-markers placed along the roadside. The car is oriented only with the lights the camera can see.

Imagine you are driving the car, if you fall asleep you may crash or get out of the road. Otherwise, if you have implemented this system, it could save your life.

Next chapters will show how I have implemented the Automatic Driving System besides the creation process of the track, algorithms, and so on.
INSTALATION PROCESS

TORCS runs on Linux (x86, AMD64 and PPC), FreeBSD, MacOSX and Windows. The source code of TORCS is licensed under the GPL ("Open Source") and is written in C++. TORCS installation is a difficult task which consist on a carefully process where is necessary be aware of what prerequisites and dependences are needed. Therefore, it is necessary install some packages, libraries and programs in order to get a successful installation.

In this project, it has been chosen Ubuntu v9.04 because of incompatibilities found with graphic card (ATI) and Ubuntu newest releases. Before installation TORCS, it is mandatory install some additional packages and update several parts of the operative system.

- Firstly, it is advisable to update the repositories because of they might be obsolete since the operative system chosen is not the newest release. The file that has the information regarding the repositories is /etc/apt/sources.list:

```
deb http://old-releases.ubuntu.com/ubuntu/ jaunty main restricted
deb-src http://old-releases.ubuntu.com/ubuntu/ jaunty main restricted
deb http://old-releases.ubuntu.com/ubuntu/ jaunty-updates main restricted
deb-src http://old-releases.ubuntu.com/ubuntu/ jaunty-updates main restricted
deb http://old-releases.ubuntu.com/ubuntu/ jaunty universe multiverse
deb-src http://old-releases.ubuntu.com/ubuntu/ jaunty universe multiverse
deb http://old-releases.ubuntu.com/ubuntu jaunty-security main restricted
deb-src http://old-releases.ubuntu.com/ubuntu jaunty-security main restricted
```

Secondly, it is necessary to install the OpenGL/DRI driver development tools and a few additional libraries such as the compilers gcc and g++, headers files for OpenGL, GLUT, GLU, XFree86, libc, ... As TORCS requires a 3D-accelerator with OpenGL support, it is compulsory to install the needed libraries.

- **OpenGL:** Depend on the graphic card installed in the computer it might be found some issues installing OpenGL. In this installation, it was found a big problem which solution is downgrade Xorg because of many problems were found with regard to the graphic card installed (ATI). Downgrade from 1.6 to 1.5.2 doing the following steps:

1) `#apt-get remove 'xserver-*'`
2) `#apt-get remove libdrm2 libdrm-intel1 libgl1-mesa-dri`
3) `#cp /etc/apt/sources.list /etc/apt/sources.list.jaunty`
4) substitute 'jaunty' with 'intrepid' everywhere in /etc/apt/sources.list
5) `#apt-get update`
6) `#apt-get install xserver-xorg-core`
7) `#apt-get install libdrm2 libdrm-intel1 libgl1-mesa-dri` → Actually, we only need to install lidrm-intel1 because the rest were installed with the installation of xserver-xorg-core. Note! It's necessary with the current repositories do apt-get install lidrm* or lidrm-dev*
8) `#mv /etc/apt/sources.list.jaunty /etc/apt/sources.list`
9) `#apt-get update`
10) `#apt-get install <non_xserver_packages_eliminated_in_step_1>`

---

Xorg is downgraded to the version 1.5.2. Therefore, OpenGL and the rest of the libraries can be installed:

1) `#apt-get install gcc g++. g++ and gcc are the compilers necessary for compile the code in C and C++ with which Torcs is programmed.`

2) `#apt-get install freeglut3-dev libalut-dev libraries of GLUT and GLU for rendering the graphics.`

3) `#apt-get install plib-dev libopenal-dev libalut-dev libxxf86vm-dev libXi-dev libXmu-dev libXrender-dev libXrandr-dev libz-dev different libraries necessary for Torcs runs properly.`

4) `#apt-get install libpng12-dev library for work with PNG images.`

5) `#apt-get install libcv-dev library used for the recognition of elements in images.`

Thirdly, having finished with all the prerequisites for TORCS installation, everything is ready to go on installing TORCS. The following steps explain how to make it:

- First step is download the source package for Linux on the TORCS’ website³

- Then, I extracted and decompressed the file in `/usr/src/torcs`:

```
# tar xfvj /usr/src/torcs/torcs-1.3.1.tar.bz2
# cd /usr/src/torcs/torcs-1.3.1
# ./configure
# make
# make install
# make datainstall
```

³ Reference: [http://prdownloads.sourceforge.net/torcs/torcs-1.3.1.tar.bz2?download](http://prdownloads.sourceforge.net/torcs/torcs-1.3.1.tar.bz2?download)
- Having installed the simulator, environment variables need to be created as LD_LIBRARY_PATH, TORCS_BASE and MAKE_DEFAULT in order to find easier the compilations in the future. By set up those variables is necessary to put them in the file .bashrc:

- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
- export TORCS_BASE=/usr/src/torcs/torcs-1.3.1
- export MAKE_DEFAULT=$TORCS_BASE/Make-default.mk

- Finally, It is necessary to reboot computer and in case of the computer ask for updating is mandatory to do it and reboot again.

Now, everything is ready and all set up correctly. Executing TORCS and seeing the next screenshot indicates that everything runs perfectly the installation was a success:
TRACK CREATION

In order to create a circuit according to the desired features, it is necessary to make a new track to implement the lights on the roadside. As it is already known, those lights are required to be detected by the camera in order to avoid collisions. Moreover, the circuit needs to be under night conditions. The following applications will help to make the track and the environment exactly as it is required:

- **Trackeditor**: This tool is provided with TORCS. It's the first step to create a track. This application is used to create the physical description of the circuit. It is able to make tracks adding straight or curves and configuring some parameters such as length, radius or banking. These information is stored in a XML file.

- **Trackgen**: This program is provided with TORCS also and is used to make tracks from the XML file generated by Trackeditor to an AC file which contents the 3D description of the track.

- **Blender**: Useful application to edit easily the 3D description of the track and modify the AC file. This software is able to add elements such as the markers on the roadside, insert textures, and so on.

- **GIMP**: This software is used to edit the images and textures in order to get an environment more realistic for the night conditions.

First of all, the circuit is designed with Trackeditor. The track is divided into segments. There are three kinds of segments: left turns, right turns and straight segments. The segments are usually short, so a turn that looks like a big turn or a long straight is most often split into much smaller segments. A straight segment contains itself information about its width and its length. A turn has information about its width, its length and its radius. I decided draw a track not quite long because the longer the circuit is, the more light-markers will be needed.
After having designed the circuit, Trackeditor generates automatically an XML file which contains information about the description of the track. This XML file is mandatory: save it in the path: torcs/tracks/C/TN/TN.XML where ‘C’ is the category of the track and ‘TN’ is name of the track. The structure of this path is compulsory to respect since Trackgen always tries to go to this path to find the XML.

Then, Trackgen will generate the ac files and create the track through XML file. There are two methods of generate the track with Trackeditor:

- # trackgen -c category -n name

  It is used to test the track. Usually, this command is used when you are checking or testing the track and just creates the track and not the terrain.

- # trackgen -c category -n name -E <n>

  It is used to generate the final track version because will create the track and the terrain.
Category indicates what kind of track is. For instance, categories such as road, oval, dirt... and the parameter -E is for the elevation of the terrain. The number 3 is used for indicates there is no slopes on the track so the terrain is flat. Finally, the command used is and below there are the information of the creation process:

```
# trackgen -c dirt -n mdh-1 -E 3
name      = mdh-1
author    = Fer
filename  = tracks/dirt/mdh-1/mdh-1.xml
nseg      = 210 version  = 4
length    = 828.316895
width     = 10.000000
pits      = 0
XSize     = 319.000031
YSize     = 219.001221
ZSize     = 0.000000
Delta X   = 0.000595
Delta Y   = -0.001282
Delta Z   = 0.000000
Delta Ang = -0.000009 (-0.000536)
=== Indices Array Size   = 16400
=== Vertex Array Size    = 49200
=== Tex Coord Array Size = 32800
=== Indices really used = 5026
GenerateMesh: right, reverse order, interior
Relief: 0 vtx, 0 seg
Load Chains
Segments = 210
xmin = 19.017883   ymin = 19.000000
xmax = 299.982147   ymax = 199.999939
Chains = 1
3885 Nodes
Renum Nodes, Triangles, Sides
GenerateMesh: left, normal order, exterior
Relief: 0 vtx, 0 seg
Load Chains
Segments = 214
xmin = -50.000000  ymin = -50.000000
xmax = 369.000031  ymax = 269.001221
Chains = 2
796 Nodes
Renum Nodes, Triangles, Sides
Generating Elevation Map tracks/dirt/mdh-1/mdh-1-elv3.png (1024, 779)
100% |***********************************************| row 779
```
Now, Trackgen has created the ac file and has generated some textures by default. Furthermore, TORCS is able to read this ac file and is possible to check out if the track fulfills the wanted requirements. There is an example in the next screenshot:

Notice the environment in the image. It doesn’t seem to have the wanted night conditions, light-makers, realistic environment, and so on. It is necessary change some textures and parameters on Blender since Trackgen have set them up by default. Some needed changes are:

- There are no light-makers on the roadside.
- The textures by default are not the appropriated to simulate night conditions. For instance:
  - It is day and is necessary a nocturnal sky.
  - Become darker the road, signals, kerbs or curbs, grass etc
- Remove the sun and elements that emits lights. Remove other elements useless.
- Numerous modifications which are necessary to get an environment as realistic as possible and always under night conditions.

Most problems are fixed editing the ac file with blender. We can find the ac file in the same folder as the XML file. In this case the name of the track ac file is mdh-1.ac.

Firstly, to open the ac file is necessary import the track to blender and will be loaded into. Notice it is important copy all the textures of the track (generate by Trackgen) in the same path of the mdh-1.ac to be loaded by Blender. They can be found on torcs/data/textures. This is an example of the track loaded with Blender:

Secondly, it is necessary to edit the track and change the textures of the sky, asphalt, grass, mountains, barriers and kerbs for another one darker and more realistic. First of all, to edit the textures is necessary use GIMP. Having them edited, the new textures are saved in mdh-1.ac folder in order to avoid problems:
- **Barriers**: the file used is `tr-barrier.rgb`. Just getting darker.

- **Asphalt**: the file used is `tr-asphalt-l-nmm.rgb`. The marks of the road have been changed to yellow in order to avoid problems detecting the light-markers which are white.

- **Curbs or Kerbs**: the file used is `tr-curb-r.rgb`. Just getting darker.

- **Grass**: the file used is `tr-grass7.rgb`. Just getting darker.
Background: the file used is background.png. The stars, the mountains, clouds and the night effects have been created also.

Most of them just need to replace the texture in Blender. However the background needs to modify the mdh-1.XML to link the image with the 3d description code:

```xml
<section name="Graphic">
  <attstr name="3d description" val="mdh-1.ac"/>
  <attstr name="background image" val="background.png"/>
  <attnum name="background type" val="4"/>
  
  <attnum name="light position x" val="-10000"/>
  <attnum name="light position y" val="-10000"/>
  <attnum name="light position z" val="10000"/>
</section>
```

Notice last 3 lines contents information about the position of the sun. It seems that it is not possible remove the sun so I set up negatives values in order to hide it.
LIGHT-MARKERS CREATION

As it is already known, the light-markers indicate the border of the track and they will be detected by the camera of the car in order to avoid collision. For this reason, it has been decided implement numerous light-makers to be easier the detection of them, the number of the markers is over 3000 distributed all along the roadsides. The markers gap is around 1 meter one another. Next illustration shows an example of a light-marker:

Firstly, to make a new object it is advisable to create first the object in a new blender project and then append the element to the track. The lights are composed of 4 planes connected each other in their extremes with a special texture simulating a reflective marker.

Secondly, the markers are put all along the roadside. In order to get a realistic effect is modified the parameter Emit which is used for establish the light is able to emit an object.
Finally, in order to get the scenario more realistic is necessary decrease Fov Factor. The Fov Factor is used to indicate how much background is wanted to see when you are driving. In other words, how much depth is wanted to see through the camera.

**FOV FACTOR**

Decreasing Fov Factor gets an environment more realistic since under night condition is impossible to see beyond the light of the car show. Therefore, the objects that are far away will not appear unless the car gets close to them. Moreover, decreasing Fov Factor is good also for the throughput since it is not necessary to render as many elements as if everything was shown at the same time. To decrease it is necessary to add next line to the XML file in the graphic section:

```xml
<attnum name="fov factor" val="0.15"/>
```
VISUAL CONTROL AND AUTOMATIC DRIVING SYSTEM (ADS)

THE DRIVER (ROBOT)

The robot is the program that is able to drive a car. It is used by TORCS and gets as input information about the current status of its car and the situation on the track. Based on this it can compute how much it wants to steer, brake or accelerate, which gear is necessary and so on. The robot returns the data to TORCS and the next simulation step will be performed. But the robot doesn’t have just to fight with the control of the car, apart from this the robot with the camera’s aid have also to avoid collisions, to overtake and to resolve other situations.

First of all, it is necessary to generate the robot skeleton. Next, it is used the robotgen script (which is provided with TORCS) to generate the initial set of files for the robot. Robotgen is a script of TORCS which is able to create robots. Then, the robotgen.gz is saved in $TORCS_BASE folder. Finally, is necessary to indicate the robots name, the author, the car chosen, and optional a description and if you want GPL headers.

```
# ./robotgen -n "bt" -a "Fernando" -c "sc-f1" –gpl
```

Generation of robot bt author Fernando
Generating src/drivers/bt/Makefile ... done
Generating src/drivers/bt/bt.xml ... done
Generating src/drivers/bt/bt.cpp ... done
Generating src/drivers/bt/bt.def ... done
Generating src/drivers/bt/bt.dsp ... done

To check if everything works is necessary compile and install the driver. To compile and install the robot just is necessary do the following.

$ cd $TORCS_BASE/src/drivers/bt
$ make
$ make install

Now the robot is properly installed. Inside the robot there is bt.cpp. This file is the most important part for the driving. This file contains the function drive which will be the responsible for handle the car during the race. This is the main function of the robot and is locate in bt.cpp:

```
static void drive(int index, tCarElt* car, tSituation *s)
{
    car->ctrl.steer = 0.01*((float)car->ctrl.Ry-(float)car->ctrl.Ly); //Steer
    car->ctrl.gear = 1; // first gear
    car->ctrl.accelCmd = 0.2; // 30% accelerator pedal
    car->ctrl.brakeCmd = 0.0; // no brakes
}
```
THE TRACK AND VISUAL CONTROL

As it is already known, the track is partitioned into segments. The segments are organized as linked list in the memory. A straight segment has a width and a length, a turn has a width, a length and a radius, everything is measured in the middle of the track. The structure tTrack is defined in $TORCS_BASE/export/include/track.h.

But, how does the track looks for the robot? In normal conditions the robot will use the track information to know about its position respect to it. But in this case, it is necessary to implement a visual control so it is not possible use such information. The robot must be able to know where is every time and just with what the camera can see.

This visual control is implemented using a camera in the front of the vehicle. The aim of the camera is to detect the light-makers which are put on the roadside. Then, the camera stores the information about the markers in a structure where is read by the driver who is the responsible for the control of the car, steering, braking, accelerating and so on.

THE CAR

The structure of the car is tCarElt. is defined in $TORCS_BASE/export/include/car.h. This is the main car structure, used everywhere in the code.

```c
typedef struct CarElt
{
  int index; /**< car index */
tInitCar info; /**< public */
tPublicCar pub; /**< public */
tCarRaceInfo race; /**< public */
tPrivCar priv; /**< private */
tCarCtrl ctrl; /**< private */
tCarPitCmd pitcmd; /**< private */
struct RobotItf *robot; /**< private */
} tCarElt;
```
The struct tCarCtrl not only is the responsible for the control of the vehicle but also is the place where the driver reads the information sent by the camera. Such info tries to give to the driver a reference about where the light-makers are and consists of 4 int variables called Lx, Ly, Rx and Ry. L means Left, R means Right, x and y are the coordinates respect to (0,0) of the position of the lowest white pixel found in each side (Left and Right). This information is useful for the driver to control the car, recognize the size of the road, its situation respect to the track and so on. The info is updated each capture that the camera takes.

**LIGHT-MARKERS DETECTION PROCESS**

As is it already known, TORCS can facilitate many information about the track such as its shape, geometric, characteristics or what is the position or orientation of the car respect to the circuit for each time. But, in the real life when you drive the car outside you don’t have that information.

The main objective is to do the simulation as realistic as possible. In the real life when someone drives under night conditions, the driver is able to see only the lights or the reflective markers of the road. Therefore, instead of using the track information that TORCS provides, it will be used the light-markers situated on the roadsides. Those light-markers are detected by the camera in order to orient the vehicle on the road as in the real world.
The process to detect the light-markers is the following:

If the position of the light-markers respect to the car is known, then it will be know where the car is every time and therefore the driver is able to steer the vehicle in a secure way. In the real life if someone who is driving falls asleep he may crash but if he has this system it could be save his life.
First of all, be aware that is necessary use some graphics libraries to capture the screen, pattern recognition. Those libraries are OpenGL and OpenCV.

**OpenGL** (Open Graphics Library) is a standard specification for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.

OpenGL is used by TORCS to draw all the 3D scenes. But in addition, in this case, we are going to help us with the OpenGL functions especially to capture the screen of the camera each time.

**OpenCV** (Open Source Computer Vision) is a library of programming functions for real time computer vision. We need to use OpenCV because of we have to process the images that the camera gives. Some OpenCV features that are necessaries for us are:

- Image data manipulation (allocation, release, copying, setting, conversion).
- Matrix and vector manipulation and linear algebra routines.
- Various dynamic data structures (lists, queues, sets, trees, graphs).
- Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, color conversion, morphological operations, histograms, image pyramids).
- Object recognition.

And the OpenCV modules that we are going to use are:

- cv - Main OpenCV functions. → #include <opencv/cv.h>
- cvaux - Auxiliary (experimental) OpenCV functions. → #include <opencv/cvaux.h>
- cxcore - Data structures and linear algebra support. → #include <opencv/cxcore.h>
- highgui - GUI functions. → #include <opencv/highgui.h>
CODE AND ALGORITHMS

TORCS provides a function which consists on capturing the screen every frame to record a video of the race. This function is very useful since to implement the camera and the algorithms that are necessary to capture the screen as well. For that, the function is modified to carry out the required aim. The function name is called recapture and is located on the raceengine.cpp. When the race starts, this function is called every frame.

Basically, the aim of this function is to detect the light-markers on both sides of the road. Firstly, the screen is captured. Then, the capture is processed and is established the ROI (Region of Interest), One ROI for the left side and another for the right one. Thirdly, white pixels are searched in both ROIs storing the position of the lowest white pixel in each side. Next, those positions are stored in a struct to be sent to the driver. Finally, Driver reads the information about the position of the light-markers and decides what to do, turn left, turn right, go straight...
static void reCapture(void){

DECLARATION AND INICIALIZATIONS

    int sw, sh, vw, vh;                //They are used to know the size of the screen
    int ph_start = 0, pw_start = 0, pw_end = 640, ph_end = 237; //They establish the first ROI (Region of Interest) of the Screen in pixels from 0,0 (bottom right of the image) to 640,270. In OpenGL the reading way is bottom up.

    int height = ph_end-ph_start;      //The height of the capture
    int width = pw_end-pw_start;       //The width of the capture

    unsigned char *captureGL;          //It is the pointer of the capture used for the OpenGL functions

    IplImage* captureCV;               //It is the pointer of the capture used for the OpenCV functions

    int widthStep, channels, ROIwidth=90, i,j; //They are used to control the size of the ROIs

    uchar *Aux;                        //Pointer to auxiliary space of memory used for try to locate the light-markers

    int Lx, Ly, Rx, Ry;                //L means Left, R means Right, x and y are the coordinates respect to (0,0) of the position of the lowest white pixel found in each side (Left and Right).

    tSituation *s = ReInfo->s;         //Structs to identify the car.
    tCarElt* car = s->cars[0];
FIRST PHASE: CAPTURING THE SCREEN

OpenGL is used to capture what the camera is seeing. First of all, a piece of memory is allocated for the capture. Then, OpenGL functions are used to read the pixels of the screen and are saved in captureGL. Finally, there is a possibility to store the capture in a PNG file.

```c
GfScrGetSize(&sw, &sh, &vw, &vh); //OpenGL function necessary to get the size of the screen

captureGL = (unsigned char*)malloc(vw * vh * 3); //A piece of memory is allocated for the capture

if (captureGL == NULL) {
    printf("Error allocating memory");
    return;
}

glPixelStorei(GL_PACK_ROW_LENGTH, 0); //OpenGL sets pixel storage modes that affect the operation of subsequent glReadPixels

glPixelStorei(GL_PACK_ALIGNMENT, 1); //OpenGL sets pixel storage modes that affect the operation of subsequent glReadPixels

glReadBuffer(GL_FRONT); //OpenGL sets the target frame buffer to read specifying the color buffer as the source for subsequent glReadPixels, as we are working with a single-buffered configuration we use GL_FRONT.

//OpenGL read the pixels from the frame buffer

glReadPixels(pw_start, ph_start, pw_end, ph_end, GL_RGB, GL_UNSIGNED_BYTE, (GLvoid*)captureGL);

//This function is already implemented in TORCS and is used just in the test period to check the captures. It converts the captured image in PNG and store in disk.
```
OPENGL → OPENCV

OpenCV is used to manipulate the image. OpenCV is unable to read directly an image generated by OpenGL. To use properly the OpenCV library is necessary to change the type of the capture to another accepted type such as IplImage*.

//Allocating memory for a copy of the capture of the screen (the capture generated by glReadPixels) but type IplImage* necessary for works with OpenCV functions.
captureCV = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);

//This loop is used because is necessary to copy the data of captureGL to captureCV which has the adequate format to work properly with OpenCV. It will be necessary invert the array because OpenGL reads in opposite way, OpenGL reads bottom up and OpenCV reads top down. Data are copied row by row.
for (i = (height-1), j=0; i >=0 ; i--,j++){  
    memcpy((unsigned char*)(captureCV->imageData + i * captureCV->widthStep),captureGL + j * width * 3, width * 3);
}

//Parameters of captureCV
height = captureCV->height;  //Image height in pixels
width = captureCV->width;    //Image width in pixels
widthStep = captureCV->widthStep;  //Size of aligned image row in bytes
channels = captureCV->nChannels; //The channels of the colors that we use. In this case are 3, our data layout of a color image is: b0, g0, r0, b1, g1, r1 ... We have to realize that is different in OpenCV is BGR and is not RGB as we are get used.
 Aux = (uchar *)captureCV->imageData;

// Pointer to aligned image data. It is going to be the auxiliary where we are storing the ROI

SECOND PHASE: ESTABLISHING THE REGION OF INTEREST (ROI) - LEFT SIDE

The Region of interest (ROI) is a rectangular area in an image, to segment object for further processing. Once the ROI is defined, most OpenCV functions will perform on that particular location. The ROI is the most likely area of the capture where there can be light-markers. Thus, two ROI will be established, one for the left side and another for the right one. Each ROI will be used to help find the light-markers in an efficiently way.

// LEFT ROI is set on the image captureCV. The size will be ROIwidth x height -> 90x237.

 cvSetImageROI(captureCV, cvRect(0, 0, ROIwidth, height));

THIRD PHASE: DETECTING THE LIGHT-MARKERS

As I mentioned before, everything is developed under a nocturnal environment so that most of the elements that are possible to find will be dark. Thus, the way to detect the light-markers will consist on reading the matrix of the image trying to find the pixels with high values in RGB scale so that these pixels have many chances of being the light-markers.

Each capture is a matrix which consists of 640x237 pixels. Each pixel contains the information of its level of red, green and blue in RGB scale so that a pixel which values are 0, 0, 0 respectively will belong to the black color. Otherwise, if the pixel has values such as 255, 255, 255 (these are the maximum) it will be a white pixel. Therefore, those pixels whose values are higher than 100 in RGB scale will be recognized as a pixel of a light-marker:
- red>100
- green>100
- blue>100

Nevertheless, those pixels whose values are lower than 100 in the RGB scale won’t be recognized as a pixel of a light-marker.
DETECTING LIGHT-MARKERS THROUGH ROI (Region of Interest)

- **Red Area**: Left ROI.
- **Blue Area**: Right ROI.
- **Green Pixel**: Indicates that pixel was the lowest pixel of a light-marker in its ROI at a given moment but now there is another pixel lower.
- **Red Pixel**: the lowest light pixel position in Right ROI.
- **Blue Pixel**: the lowest light pixel position in Left ROI.

The ROI are used to help to find in an efficiently way the position of the lowest white pixel of the light-markers in each side. Each ROI is read right to left and top down.

Red and Blue pixels indicate they are the lowest white pixels in its respective areas. Those pixels whose color is green indicates that pixel was the lowest white pixel in its ROI at a given moment but now there is another pixel lower. In other words, if it is found a white pixel ‘A’ and it is the lowest so far, the pixel is marked internally as blue or red (depend on its side) but it is found another white pixel ‘B’ and is lower than ‘A’, ‘A’ will change to green and ‘B’ will change to either red or blue.

All of these pixel and areas modifications take place just in a test file called: ROI.png. But, during the simulation, those modifications are transparent. When it is finished scanning each ROI, the last white pixel located is the lowest pixel of a light-marker. In order to identify them on the ROI.png we change the pixel color with a blue pixel in left ROI and a red pixel in right ROI.
Those variables will keep the values of the position of the lowest white pixel of the light-makers. L means left ROI, R means right ROI. X and y are the position in pixels indicating the lowest position of the ROI where has been found a marker. For example Lx and Ly values will be modify when is found a white pixel with the value of its position if it is the lowest so far and in the left ROI

\[
\begin{align*}
Lx &= -1; \\
Ly &= -1; \\
Rx &= -1; \\
Ry &= -1;
\end{align*}
\]

This loop finds the lights on the left ROI. The pixel whose RGB values are Red>100, Green>100 and Red>100 it will be stored its position as the lowest white pixel in Lx and Ly.

\[
\begin{align*}
\text{for}(i=0; i<\text{height}; i++)& \\
\text{for}(j=0; j<\text{ROI width}; j++)& \\
\text{if } & ((\text{Aux}[i*\text{widthStep}+j*\text{channels} + 0]>100) \& \& \\
& (\text{Aux}[i*\text{widthStep}+j*\text{channels} + 1]>100) \& \& \\
& (\text{Aux}[i*\text{widthStep}+j*\text{channels} + 2]>100))&
\end{align*}
\]

\[
\begin{align*}
\text{if}(i>\text{Ly}) & \\
Lx &= j; \\
Ly &= i;
\end{align*}
\]

\[
\begin{align*}
\text{end width loop}& \\
\text{end height loop}&
\end{align*}
\]
SECOND PHASE (AGAIN) : ESTABLISHING THE REGION OF INTEREST (ROI) – RIGHT SIDE

//Reset image ROI. Now the capture doesn’t have ROI.
cvResetImageROI(captureCV);

// RIGHT ROI is set on the image captureCV. The size will be ROIwidth x height -> 90x237.
cvSetImageROI(captureCV, cvRect(0, 0, ROIwidth, height));

THIRD PHASE (AGAIN) : DETECTING THE LIGHT-MARKERS – RIGHT SIDE

for(i=0;i<height;i++){
    for(j=((width-1)-ROIwidth);j<width;j++){
        //height loop
        //width loop
        if ((Aux[i*widthStep+j*channels + 0]>100)&&(Aux[i*widthStep+j*channels + 1]>100)&&(Aux[i*widthStep+j*channels + 2]>100)){
            if(i>Ry){
                Rx=j;
                Ry=i;
            }
        } //end width loop
    } //end height loop
} //end width loop
} //end height loop

cvResetImageROI(captureCV); // Reset image ROI. Now the capture doesn’t have ROI.
if((Lx==-1)&&(Ly==-1)) {
    if((Rx==-1)&&(Ry==-1)) {
        printf("Light not found Both Sides!\n", Lx, Ly);
    } else {
        printf("Light not found Left Side!\n", Lx, Ly);
    }
} else {
    if((Rx==-1)&&(Ry==-1)) {
        printf("Light not found Right Side!\n", Lx, Ly);
    } else {
        printf("%f\n", 0.01*(float)car->ctrl.Ry-(float)car->ctrl.Ly);
    }
}

FOURTH PHASE: SENDING COORDINATES TO DRIVER
The coordinates of the position of the light-markers are stored in the struct of the car so that the driver can access that info and be able to steer safely the car

// those positions are stored in a struct to be sent to the driver.

car->ctrl.Lx=Lx;
car->ctrl.Ly=Ly;
car->ctrl.Rx=Rx;
car->ctrl.Ry=Ry;
//Free space of memory of each image
Having finished scanning both ROI, the position of the lowest white pixels of each side are stored. In the image they are marked with a blue pixel in left ROI and a red pixel in right ROI. Then, theirs heights are compared and the difference of of Ly and Ry multiplied by a constant $\alpha$ will be the steering. $\text{Steering} = (R_y - L_y) \cdot \alpha$

If $R_y > L_y$ the driver must turn the wheel to left. However, if $R_y < L_y$ it must turn the wheel to right. Finally, when $R_y = L_y$ the steering will be 0 so the car will keep going straight.
static void drive(int index, tCarElt* car, tSituation *s)
{
    car->ctrl.steer = 0.01*((float)car->ctrl.Ry-(float)car->ctrl.Ly); // Steering = ( Ry – Ly ) · α / α=0.01
    car->ctrl.gear = 1; // first gear
    car->ctrl.accelCmd = 0.2; // 20% accelerator pedal
    car->ctrl.brakeCmd = 0.0; // no brakes
}
This graphic shows an example of a lap in one of the oval circuits created. The simulation consists of activating and deactivating the ADS in some parts of the circuit. The blue dark line is the reference path. In other words, it represents the middle of the road and it is the ideal path where the car should go if the ADS is activated. The red one represents the path when ADS is activated and the light blue represents the path when the ADS is deactivated.

When the ADS is deactivated the steering wheel is blocked and therefore the car gets out of the route. However, when the ADS is activated the driver tries to stabilize the steering wheel in order to get back to the route as we can see in the sinusoid.

Next page, contains the simulation in another circuit with the same information.
The path is traveled in the clockwise direction.
REFERENCES


Track Tutorial [http://usuarios.multimania.es/f1torcs/](http://usuarios.multimania.es/f1torcs/)


OpenCV 1.0 API [http://cgi.cs.indiana.edu/~oleykin/website/OpenCVHelp/](http://cgi.cs.indiana.edu/~oleykin/website/OpenCVHelp/)

Basic Structures — OpenCV 2.0 C Reference [http://opencv.willowgarage.com/documentation/basic_structures.html](http://opencv.willowgarage.com/documentation/basic_structures.html)

Introduction to programming with OpenCV [http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html#SECTION000250000000000000000000](http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html#SECTION000250000000000000000000)
