Autonomous Car
-
Monitoring Remote Sensors

Author: Baeta Arribas, Francesc
Tutor: Herm, Ota
0 – Introduction

The *Autonomous Car Project* is a project started by *Department of Telecommunications Engineering* in *Czech Technical University in Prague (CVUT)*. The aim of this project is to build a car able to drive by itself, autonomously. The father of the idea was *Doc. Ing. Jiří Chod CSc.*, but the leader of this project is *Ing. Ota Herm*, I’m grateful for his support and help.


The project in CVUT is working in the same line, but not so big, due to the growing project has limited resources. The project is working with 1:8 scale car and some sensors like GPS or Camera, which is a start point. Thus the team must develop something presentable and then go to specialized applications and acquire more hardware.

The project has more applications than only competitions like car for disabled drivers, expeditions in dangerous environments and so on.

Due to at this time on the project are working 5 master students, each one on his specialized field, and like is written above this is a starting project, it is very important to remark that are some important requirements to fulfill.

- Open Source: More a recommendation than a requirement, using open source free software helps to be modular and to ensure that any future developer or user can use this software.
- Compatibility: All the implementations must be as compatible as possible between each other also it should be able to run over different operating systems.

- Easy implementations: The aim of implementations is to work, thus simplicity in the implementations would be appreciate.

- Standard: Go after standards like RFC it’s recommended but not forced if following the standards complicates too much the work.

- Clearly and Help documentation: Implementations should be programmed clearly and well documented for to help future developers.

0.1 - Application and Multimedia

This TFC will be focused in Application and Multimedia issues of the Autonomous Car Project trying to accomplish the proposed requirements.

One of the big needs of the project is to monitor the car from distance to know how the car drives and how respond on the road. The following goals are set up for this TFC.

1- Monitoring Video Camera Streaming: The camera is also a sensor of the car, but it will be treating separately because of the big weight in bytes that is a video streaming.
2- Monitoring Sensors Data Streaming.
3- Multimedia Portal: To view the monitored sensors.

In addition to the global requirements, MATLAB compatibility became almost the most important rule to carry out. Due to the advantages that brings MATLAB like simplicity, power, easy programming, and easy acquisition of data from hardware sensors.

MATLAB is a numerical computing environment and programming language. Created by “The MathWorks”, MATLAB allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. Although it specializes in numerical computing, an
optional toolbox interfaces with the Maple symbolic engine, allowing it to be part of a full computer algebra system.
1 - Monitoring Video Camera Streaming

One of the first aims of this project is to stream video, from a camera to Internet. Firstly the research was focused on find implemented software that allows to stream video data easily.

Some concepts should be introduced first.

a) A codec is a device or program capable of performing encoding and decoding on a digital data stream or signal. The word codec may be a combination of any of the following: 'Compressor-Decompressor', 'Coder-Decoder', or 'Compression/Decompression algorithm'.

b) A container format is a computer file format that can contain various types of data, compressed by means of standardized audio/video codecs. The container file is used to identify and interleave the different data types. Simpler container formats can contain different types of audio codecs, while more advanced container formats can support multiple audio and video streams, subtitles, chapter-information, and meta-data (tags) - along with the synchronization information needed to play back the various streams together.

1.1 - VideoLAN - VLC media player 0.8.6a - Research

VideoLAN is a software project for video that release free software under GNU License. We had a look on VLC media player 0.8.6a because of:

- It is a free cross-platform media player
- It supports a large number of multimedia formats, without the need for additional codecs
- It can also be used as a streaming server, with extended features (video on demand, on the fly transcoding, ...)

VLC can get data from files, disc (DVD, VCD, Audio CD), directly from network (UDP, RTP, HTTP, HTTPS, MMS, FTP, RTSP) or a properly installed video device.
These reasons realize that VLC fulfills the requirements. Thus we started to test.

1.1.1 - Video acquisition from Webcam
VLC has a friendly and intuitive Graphical Interface. However all the next reference on how to, are going to be write in the command line syntax.

How to get audio and video data from devices:

```
dshow:// :dshow-vdev="" :dshow-adev="" :dshow-size=""
```

How to send data (in addition the last sentence):

```
:sout=#duplicate{dst=std{access=udp,mux=ts,dst=127.0.0.1:}
```

This sentence means that the data stream will be duplicated and send it via UDP to the IP address 127.0.0.1 on port 1234. The audio and video data will be multiplexed in MPEG-TS format.

Output and encapsulation method formats able to choose, follows the next table:

1.1.2 –Codecs
To our surprise, when we tried to stream the data from camera to a unicast IP, it was impossible to reproduce data on the VLC client. That happened due to VLC doesn’t allow to send RGB raw data, because of encapsulation methods doesn’t expect raw data. On the other hand VLC is able to solve this problem giving a lot of different codes to transcodificate. Thus the container can understand given data.

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Encapsulation Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPEG-TS</td>
</tr>
<tr>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>MMS</td>
<td>Yes</td>
</tr>
<tr>
<td>RTP</td>
<td>Yes</td>
</tr>
<tr>
<td>UDP</td>
<td>Yes</td>
</tr>
</tbody>
</table>
1.1.2.1 - Codecs Test

Although VLC has more available codecs, we report these that works.

The tests have been done with the following description:

<table>
<thead>
<tr>
<th>Type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware details</strong></td>
<td></td>
</tr>
<tr>
<td>Video input</td>
<td>VIMICRO USB PC Camera</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Centrino Duo T5500 @ 1.66Ghz</td>
</tr>
<tr>
<td>RAM</td>
<td>1,00 GB</td>
</tr>
</tbody>
</table>

**Test 1 - Software details**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>24 RGB</td>
</tr>
<tr>
<td>Size [pixels]</td>
<td>800x600</td>
</tr>
<tr>
<td>Framerate [fps]</td>
<td>15</td>
</tr>
<tr>
<td>Bitrate (kb/s)</td>
<td>1024</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Codec</th>
<th>CPU rise</th>
<th>Subjective quality</th>
<th>Delay [seconds]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV2</td>
<td>35 %</td>
<td>Good</td>
<td>1,5</td>
</tr>
<tr>
<td>DIV3</td>
<td>40 %</td>
<td>Good</td>
<td>1,5</td>
</tr>
<tr>
<td>H264</td>
<td>65 %</td>
<td>Good</td>
<td>1,5</td>
</tr>
<tr>
<td>WM1</td>
<td>27 %</td>
<td>Good</td>
<td>1,5</td>
</tr>
<tr>
<td>WM2</td>
<td>30 %</td>
<td>Good</td>
<td>1,5</td>
</tr>
<tr>
<td>MJPG</td>
<td>30 %</td>
<td>Acceptable</td>
<td>1,5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(pixeled)</td>
<td></td>
</tr>
</tbody>
</table>

**Test 2 - Software details**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>24 RGB</td>
</tr>
<tr>
<td>Size [pixels]</td>
<td>320x240</td>
</tr>
<tr>
<td>Framerate [fps]</td>
<td>15</td>
</tr>
<tr>
<td>Bitrate (kb/s)</td>
<td>128</td>
</tr>
<tr>
<td>Codec</td>
<td>CPU rise</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>DIV2</td>
<td>13 %</td>
</tr>
<tr>
<td>DIV3</td>
<td>12 %</td>
</tr>
<tr>
<td>H264</td>
<td>30 %</td>
</tr>
<tr>
<td>WM1</td>
<td>12 %</td>
</tr>
<tr>
<td>WM2</td>
<td>15 %</td>
</tr>
<tr>
<td>MJPG</td>
<td>14 %</td>
</tr>
</tbody>
</table>

Notice that delay is always the same value. This is a problem for “real-time” transmissions given that just networks can increase too much the final delay. Delays problems consist into VLC buffers. -*We tried to reduce cache down to 5ms but that doesn’t solve anything*- We will bear in mind these 1.5 disappointing seconds for future usages.

Pay attention that if the same test is done with a slower CPU (working at 100%) the delay might be increase.

Once we have tested these codecs on VLC, we noticed that DIV2, DIV3, WM1, WM2 works almost the same. These codecs get a quite good relation among CPU consume, Bitrate and Subjective quality.

On one hand, h264 is better to codificate movement due to its implementation has focused in that. We could check this while we did the tests. On the other hand, h264 needs more CPU powerful.

MJPG has not additional problems to codificate movement because of each frame is codificated separately using JPEG images codification. Therefore too much CPU powerful is required. Actually the biggest problem of MJPEG is that needs high bitrate to show a good subjective quality. If there is not a high bandwidth, pixeled problems are always appreciable.
1.1.3 – Conclusion

VLC is powerful software that brings lot of possibilities to reproduce, to save, to transcodificate or to stream video data.

To stream camera data the most important is to choose what codec will adjust to our requirements. Depending network bandwidth, difficulty implementation, CPU power or delays, it’s better to use one or other.

But 1.5 delay seconds are anyhow disappointing for a “real-time” worksheet.

1.2 - MJPEG format – Research

After understanding the VLC possibilities, MJPEG is studied because of the easiness over MATLAB to treat JPEG images.

1.2.1 - JPEG introduction

In computing, JPEG is a commonly used standard method of compression for photographic images. The name JPEG stands for Joint Photographic Experts Group, the name of the committee who created the standard. The group was organized in 1986, issuing a standard in 1992 which was approved in 1994 as ISO 10918-1. JPEG should not be confused with MPEG, the Moving Picture Experts Group, which produces compression schemes for video.

JPEG itself specifies both the codec, which defines how an image is compressed into a stream of bytes and decompressed back into an image, and the file format used to contain that stream. The compression method is usually lossy compression, meaning that some visual quality is lost in the process, although there are variations on the standard baseline JPEG which are lossless. There is also a "progressive" format, in which data is compressed in multiple passes of progressively higher detail. This is ideal for large images that will be displayed whilst downloading over a slow connection, allowing a reasonable preview before all the data has been retrieved. However, progressive JPEGs are not as widely supported.

The file format is known as 'JPEG Interchange Format', as specified in Annex B of the standard. This is often confused with the JPEG File Interchange Format (JFIF), a minimal version of the JPEG format that was deliberately simplified so that it could be
widely implemented and thus become the de-facto standard. Most image editing software programs that write to a "JPEG file" are actually creating a file in JFIF format.

Image files that employ JPEG compression are commonly called "JPEG files". The most common file extension for this format is .jpg, though .jpeg, .jpe, .jfif and .jif are also used. It is also possible for JPEG data to be embedded in other file types, such as TIFF format images.

JPEG/JFIF is the format most used for storing and transmitting photographs on the World Wide Web. For this application, it is preferred to formats such as GIF, which has a limit of 256 distinct colors that is insufficient for color photographs, and PNG, which produces much larger image files for this type of image. The compression algorithm is not as well suited for line drawings and other textual or iconic graphics, and thus the PNG and GIF formats are preferred for these types of images.

The MIME media type for JPEG is image/jpeg (defined in RFC 1341).

1.2.2 - MJPEG formats

Motion JPEG (M-JPEG) is an informal name for multimedia formats where each video frame or interlaced field of a digital video sequence is separately compressed as a JPEG image. It is often used in mobile appliances such as digital cameras.

Motion JPEG uses intraframe coding technology that is very similar in technology to the I-frame part of video coding standards such as MPEG-1 and MPEG-2, but does not use interframe prediction. The lack of use of interframe prediction results in a loss of compression capability, but eases video editing, since simple edits can be performed at any frame when all frames are I-frames. Video coding formats such as MPEG-2 can also be used in such an I-frame only fashion to provide similar compression capability and similar ease of editing features.

Using only intraframe coding technology also makes the degree of compression capability independent of the amount of motion in the scene, since temporal prediction is not being used. (Using temporal prediction can ordinarily substantially improve video compression capability, but makes the compression performance dependent on how well the motion compensation performs for the scene content.) Because of this, it is
used in surveillance cameras which only take one frame per second, in which time there could be large amounts of change.

The bitrate falls between uncompressed formats (like RGB, compression 1:1, and YCbCr, compression 1:1.5 to 1:2.5) and MPEG (1:100). Data rates in the range of 29 Mbit/s are very high quality, but also result in comparatively large file sizes.

There isn’t a standard container for MJPEG consequently two private implementations have been released. Microsoft defined it own implementation inside the RIFF (Resource Interchange File Format).

For Quicktime formats, Apple has defined two types of coding: MJPEG-A and MJPEG-B. MJPEG-B no longer retains valid JPEG Interchange Files within it, hence it is not possible to take a frame into a JPEG file without slightly modifying the headers.

1.2.3 - Summary RFC 2035/2435 - RTP Payload Format for JPEG
These RFCs explains how to encapsulate images, codified by JPEG, into a RTP packet. Thus to create a video stream of JPEG frames might send it over IP to travel into a network. This stream called JPEG-movie or M-JPEG has some restrictions comparing to JPEG possibilities.

Remember that JPEG codec has four operating modes:

1) Sequential DCT mode
2) Progressive DCT mode
3) Lossless mode
4) Hierarchical mode

But, only the **sequential DCT** operating mode and restrict the set of predefined RTP/JPEG type codes to *single-scan, interleaved images* are assumed. Thus means in JFIF format that our image database must have, into Start of Scan marker (FFDA) header, the *N_components* set to 03.
JPEG over RTP must not use JFIF format due to these RFCs recommends a particular format to stream.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| Ver | P | X | CC | M | PT | Sequence numbers |
| Timestamp |
| SSRC |
| CSRC [0..15] … |
| Type-specific | Fragment offset |
| Type | Q | Width | Height |
| [ Optional Header ] |
| JPEG payload |

Legend: **RTP Header**  **JPEG Header**  **Optional Headers**

*Figure: Headers formats*

1.2.3.1 - RTP Header

There are some specification parameters that rarely changes from frame to frame within the same video stream. Hence JPEG data is written in abbreviated format. Each frame begins immediately with the single entropy-coded scan.

These invariant parameters like the frame and scan headers is represented within RTP/JPEG header.

Otherwise Huffman tables and color space hardly ever are modified during the video stream.

Quantization tables or images size might vary (e.g. to implement rate-adaptive transmission or allow a user to adjust the "quality level" or resolution manually). Each value is stored into JPEG header.

Due to JPEG frames are usually larger than MTU (Maxim Transfer Unit), each frame must be fragmented into several IP/RTP packets. The last packet of a JPEG frame must be set the Marker Bit from 0 to 1, within the RTP header.
The RTP timestamp use 90000Hz units. Each RTP packet from a frame must have the same timestamp value.

1.2.3.2 - JPEG Header
Type-specific value depends on the application. But, if no interpretation is specified, his field must be zeroed on transmission and ignored on reception.

Fragment offset is the position of the current fragment over the hole frame, measured in bytes unit.

The Type field specifies the information that would otherwise be present in a JPEG abbreviated table-specification as well as the additional JFIF-style parameters not defined by JPEG.

<table>
<thead>
<tr>
<th>Value</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-63</td>
<td>Reserved as Fixed</td>
</tr>
<tr>
<td>64-127</td>
<td>Restart Marker - Reserved as Fixed</td>
</tr>
<tr>
<td>128-255</td>
<td>Free – Session Setup Protocol</td>
</tr>
</tbody>
</table>

Currently, of the group of fixed mapping only 0 and 1 value are defined. The 64 and 65 values correspond at the same specifies of 0 and 1, but they indicates that the restart marked must be.
Numbers among 2-5 are reserved and should not be used.

RTP/JPEG types describe below (using YUV or YCbCr sample format in JPEG):

<table>
<thead>
<tr>
<th>Types</th>
<th>Component</th>
<th>Horizontal Sample Factor</th>
<th>Vertical Sample Factor</th>
<th>Quantization Table Number</th>
<th>Sample Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 64</td>
<td>1 (Y)</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4:2:2</td>
</tr>
<tr>
<td></td>
<td>2 (U)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 (V)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Quantization Table Numbers refers to the quantization tables from Annex K from JPEG specifications that have been recommended by the ITU-R 601 standard.

The Q field defines the quantization tables for this frame.

<table>
<thead>
<tr>
<th>Value</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-127</td>
<td>Q.T. Algorithm determined by Type field</td>
</tr>
<tr>
<td>128-255</td>
<td>Q.T. Header appears – Specifying Q.T. in-band</td>
</tr>
</tbody>
</table>

Width and Height fields encodes the size of the frame, each value is encoded in 8-pixel multiples (e.g. 40 denotes 240 pixels). The maximum value is 2040 pixels.

1.2.3.3 - Optional Headers
These optional headers must be between JPEG header and JPEG data.
Restart Market Header or Quantization Table Header or both can lie between the corresponding layers.

1.2.3.3.1 - Restart Market Header
Like said above, if type field is among 64-127 this optional header must be after that main JPEG header. Given that, the data stream containing restart markers is able to be decoded.
Restart markers are useful for transmitting an image over an untrustworthy network. If a file gets corrupted, it will only be corrupted up to the next restart marker. Restart markers add a small amount to the file size of an image.

The document “Restart marker regulation technique for progressive JPEG image coding in mobile communications” say that unfortunately, misinterpreted markers may cause serious error damage due to the error propagation. Therefore, a restart marker regulation technique is proposed in this document to preprocess restart markers at the
decoding end. All erroneous restart markers are corrected and rearranged in the correct order. After decoding, isolated erroneous restart intervals are detected and further processed by the error concealment to reduce image degradation. The simulations demonstrate that the proposed scheme does significantly improve the image quality in error-prone environments.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| Restart Interval | F | L | Restart Count |

Restart Interval field express the number of MCUs that appears between restart markers. This 16bit value is the same that would be in the mark of JFIF segment header. This value must not be zeroed.

The F (first), L (last) and Restart Count fields are used if the restart intervals in a frame are not guaranteed to be aligned to the packet boundaries (the F and L bits must be set to 1 and the Restart Count must be set to 0x3FFF. This indicates that a receiver must reassemble the entire frame before decoding it).

For more details of partial frame decoding: RFC 2435 - 3.1.1 - 4th paragraph.

1.2.3.3.2 - Quantization Table Header

This optional header must appear after JPEG header when Q value is among 128-255. Also be after Restart Interval Header, if present. It specify a customize quantization table associated with the Q value on the JPEG header.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| MBZ | Precision | Length |

Quantization Table Data

....
Length field determine the length in bytes of the Quantization Table Data that follows. If received length value is larger than remaining bytes on the packet, this packet must be discarded.

Each table is an array of 64 values given in zigzag order, identical to the format used in a JFIF DQT marker segment.

Precision field indicates the size of the coefficients in the table. Therefore the next fields depend on that value.

<table>
<thead>
<tr>
<th>Value</th>
<th>Coefficient Size</th>
<th>Table length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8 bits</td>
<td>64 bytes</td>
</tr>
<tr>
<td>1</td>
<td>16 bits</td>
<td>128 bytes</td>
</tr>
</tbody>
</table>

Obviously packets must not contain a Q value among 128-255 and Length is zero.

1.2.3.4 - JPEG Payload
This data is a whole single scan set by entropy-encoded words. It ends either with an EOI (end of image) marker, or implicitly when the image is fully parsed.

1.2.3.5 - Implementation
The RFC 2435 through some Appendix helps to implement given C source codes.

- Appendix A: to create a quantization table from a Q factor.
- Appendix B: to create the JPEG marker segments corresponding to the table-specification data that is absent from the RTP/JPEG body.
- Appendix C: to illustrate the RTP/JPEG packet fragmentation and header creation
1.3 - Stream MJPEG Solution (MATLAB Toolbox)

After MJPEG have been studied we chose neither follow the RFC 2035/2435 nor Apple solution nor Microsoft solution. From here, we are going to understand MJPEG as continuous images in JPEG format, without headers or overhead between them. Thus we carry out the opened description of MJPEG and VideoLAN understand the format like this, because of JPEG format define a mark which identifies the end of an image.

The scenario where the implementation will work is:

1º) MATLAB get image by image form the webcam.
2º) After MATLAB gets one image, it is sent to the connection.
3º) Then VideoLAN receives the image that is showed on screen.

For more details of implementation: help documentation in Matlab Toolbox inside CD.

Also with this method we are able to reduce the delay to 500ms, due to the JPEG codification is done in MATLAB that it’s quicker than VideoLAN codification.

*Picture: streaming a milliseconds clock.*
2 - Monitoring Sensors

The next aim of this project is to stream data from different sensors. The scenario it’s similar than the webcam streaming scenario.

![Diagram of sensors, Matlab, Servlet, and Browser](image)

2.1 - Sensor to XML (eXtensible Markup Language)

The first step is to store periodically in XML format all the information acquired by the sensors. Due to not all the sensors are ready to work we are going to implement this part most opened as possible.

The XML file must follow the next DTD (Document Type Definition) structure:

```xml
<!DOCTYPE SENSORSET [
<!ELEMENT sensorset (sensor)+>
<!ELEMENT sensor (name, value) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT value (#PCDATA) >
]>
```

Therefore the XML file has to be like it’s showed above:

```xml
<?xml version="1.0" encoding="utf-8"?>
<sensorset>
  <sensor>
    <name>Clock</name>
    <value>17/6/2007-15:3:29.906</value>
  </sensor>
</sensorset>
```

This document is created and stored by MATLAB periodically. For more details of implementation: help documentation inside the CD
2.2 - Displaying XML information

The second step is to publish or to send these sensors data into an IP network where through website browser software we will be able to see the data information. Browser software has been chosen due to nowadays almost all the operating systems include this software that allows representing data easily.

The web application will use AJAX technology. AJAX is a web development technique used for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is intended to increase the web page's interactivity, speed, functionality, and usability.

The name is an acronym standing for Asynchronous JavaScript and XML. Ajax is asynchronous in that loading does not interfere with normal page loading. JavaScript is the programming language that Ajax function calls are made in. Data retrieved using the technique is commonly formatted using XML, as reflected in the naming of the XMLHTTPRequest object from which Ajax is derived.
Ajax is a cross-platform technology usable on many different operating systems, computer architectures, and Web browsers as it is based on open standards such as JavaScript and XML, together with open source implementations of other required technologies.

At the present time a lot of web applications often explode this technology like Google Maps, Mapy, Google suggests, etc. To use AJAX technology and to develop our viewing application we will use GWT (Google Web Toolkit) due to Google Development products keep on growing with a good documentation as APIs or help documents and examples.

2.3 - Autonomous Car Web Application

This application has two different panels: Sensors Pane and Video Pane. The development tries to follow the HTML+CSS model. Thus, HTML own data and CSS gives graphical format.

2.3.1 - XML sensors

Here is where the data from sensors will be displayed. This is organized through two tabs. First tab shows the data sensors into a table and it is switch on by default. Into second tab XML raw data is showed.

<table>
<thead>
<tr>
<th>Sensor Panel</th>
<th>Sensors Pane</th>
<th>XML Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td>17/7/2007-17:21:18:546</td>
<td></td>
</tr>
</tbody>
</table>

```xml
<?xml version="1.0" encoding="utf-8"?>
<sensorset>
  <sensor>
    <name>Clock</name>
    <value>17/7/2007-17:19:33.546</value>
  </sensor>
</sensorset>
```
2.3.2- VLC video plug-in

VideoLAN developers implemented also a plug-in to embed VLC objects into websites. Mozilla, Firefox, Safari and Explorer browsers are supported. The VLC object is embed to the HTML website and it’s controlled through JavaScript.

Our VLC website implementation just controls the following options:

- Play
- URL
- Full-size

For more detailed information look the source codes of “vlcplayer.html” or consult “VLC plug-in” website.
2.3.3 - Website design

Design format is written in the CSS document "styles.css". The colors agree Google Applications style due to GWT has been used to develop this web application. Finally the Autonomous Car – Monitoring Sensors look like:

Figure: Web Application Appearance
3 - Bibliography

- Restart marker regulation technique for progressive JPEG image coding in mobile communications - http://cat.inist.fr/?aModele=afficheN&cpsidt=865182
- VideoLAN information - http://www.videolan.org
- Que es AJAX - http://www.programacionweb.net/articulos/articulo/?num=317
Annex A

Quantization Tables recommended on ITU-R 601 showed below:

{16, 11, 10, 16, 24, 40, 51, 61,
  12, 12, 14, 19, 26, 58, 60, 55,
  14, 13, 16, 24, 40, 57, 69, 56,
  14, 17, 22, 29, 51, 87, 80, 62,
  18, 22, 37, 56, 68, 109, 103, 77,
  24, 35, 55, 64, 81, 104, 113, 92,
  49, 64, 78, 87, 103, 121, 120, 101,
  72, 92, 95, 98, 112, 100, 103, 99}

*Figure: Luminescence Q.T.*

{17, 18, 24, 47, 99, 99, 99, 99,
  18, 21, 26, 66, 99, 99, 99, 99,
  24, 26, 56, 99, 99, 99, 99, 99,
}

*Figure: Chrominance Q.T.*