Real Time Vision System for a Small Size League Team

Luis A. Martinez-Gomez Alfredo Weitzenfeld
luis@mcculloch.itam.mx alfredo@itam.mx
Comp Eng Dept, ITAM
Rio Hondo 1, San Angel Tizapan
Mexico City, MEXICO, 01000

***Abstract*—The RoboCup *Small Size League* (SSL) is a challenging environment for computer vision applications. Ball and robots move at speeds up to 2 m/s and the need of tracking them precisely requires the implementation of a high performance vision system.**
We will present the vision system developed in the ITAM Small Size Team (Eagle Knights) explaining the main algorithms and technical decisions made in the implementation. We pay special attention to the details necessary to give new teams a head start in their own implementation based on knowledge gained in two years of experience in SSL.

I. INTRODUCTION

RoboCup[1] is an international joint project to promote AI, robotics and related field. In the Small Size League, two teams of five robots up to 18 cm in diameter play soccer on a 4 by 5.4 m carpeted soccer field.

The typical architecture of a team in the *Small Size League* (SSL) has four main components: the vision system, the AI system, five robots and the referee box.
The vision system digitally process two video signals from the cameras mounted on top of the field. It computes the position of the ball and robots on the field, including orientation of the robots in its own team transmitting the information back to the AI system.
The AI system receives the information and makes strategic decisions. The actions of the team are based in a set of roles (goalkeeper, defense, forward) that exhibit behaviors according to the current state of the game. To avoid collision with robots of the opposite team an obstacle avoidance module is used. The decisions are converted to commands that are sent back to the robots via a wireless link. The robots execute these commands and produce mechanical actions as ordered by the AI system. This cycle is repeated 60 times per second. Finally the referee can communicate additional decisions (infraction, goal scored, start of the game,etc.) sending a set of predefined commands to the AI system through a serial link. Figure 1 shows a schematic of the architecture.

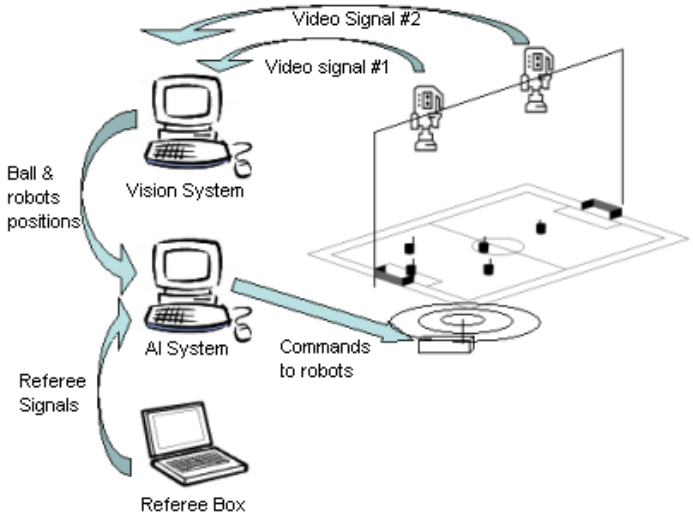

Fig 1. Typical architecture of a SSL team

The vision system is the only source of feedback in the whole architecture, if the data given by the vision system is wrong the overall performance of the team will be severely affected. That's why the vision system should be robust enough to compensate for any possible mistakes.

The main object characteristics used by the vision system are the colors defined in the rules [2] of the SSL. The ball is a standard orange golf ball. The robots of one team must have on top of them a 50 mm blue colored circle while the other team must have a yellow patch.

The main tasks of the vision system are:
1. Capture video from the cameras mounted on top of the field in real time.

2. Recognize the set of colors assigned in the rules to the objects of interest in the field (robots and ball).
3. Identify and compute the orientation and position of the robots in the team
4. Compute the position of the robots of the opposite team.
5. Transmit the information back to the AI system.
6. Adapt to different light conditions (color calibration procedure).

II. SYSTEM DESIGN

The system is modular to allow future updates and improvements. It has several modules, each module it's a functional block with a specific task. Figure 2 shows the vision system architecture.

1. CAPTURE MODULE. Assign a physical capture device to the cameras, the type of connection (IEEE1394, S-Video, Composite, etc), the resolution of the image, and the frame rate.
2. PREPROCESSING MODULE. Modify the quality of the image, such as brightness, contrast, gamma, etc.
3. OBJECT CALIBRATION MODULE. This module is a tool to establish the thresholds of each component according to the space color for every object of interest (robots and ball).
4. SEGMENTATION MODULE. Separate each pixel of the images into object classes. The module consist of two segmenters, each one using the thresholds values assigned to the camera for every object of interest.
5. BLOB BUILDER MODULE. Connects the segmented pixels into blobs. Before reaching this module the image is composed of separate pixels; when a blob is constructed useful information is computed such as the area, centroid, bounding box, etc. A joint list of blobs for the two cameras is generated for each color.
6. ACTIVATION/DESACTIVATION MODULE. Enables or disables the use of a particular robot. Sometimes a team can play with less robots so this information is useful to avoid unnecessary searching processes.
7. RECOGNITION MODULE. Selects the regions that adjust better to the objects searched. It has a selection criteria for every kind of object. For the ball we select the orange blob that is nearest to an area of 85 pixels (with an image resolution of 640x480). For the robots of the opposite team the selection criteria consists in selecting the blobs of the corresponding color of the central patch with an area nearest to 115 pixels (the area of the patch is bigger than the ball). The number of blobs selected are determined in the Activation/ Desactivation module. For the team the procedure is similar to the one used for the robots of the opposite team, but additional to the central patch, a search for extra patches is necessary. The extra patches are employed for identification and orientation computation.
8. GEOMETRIC CALIBRATION MODULE. This module computes the internal and external parameters of the cameras using the Tsai method [3]. This parameters are used to correct the distortion produced by the lenses of the camera.
9. LOCALIZATION MODULE. Computes the position of all objects in the field. It uses the camera parameters obtained in the Geometric Calibration module to undistort

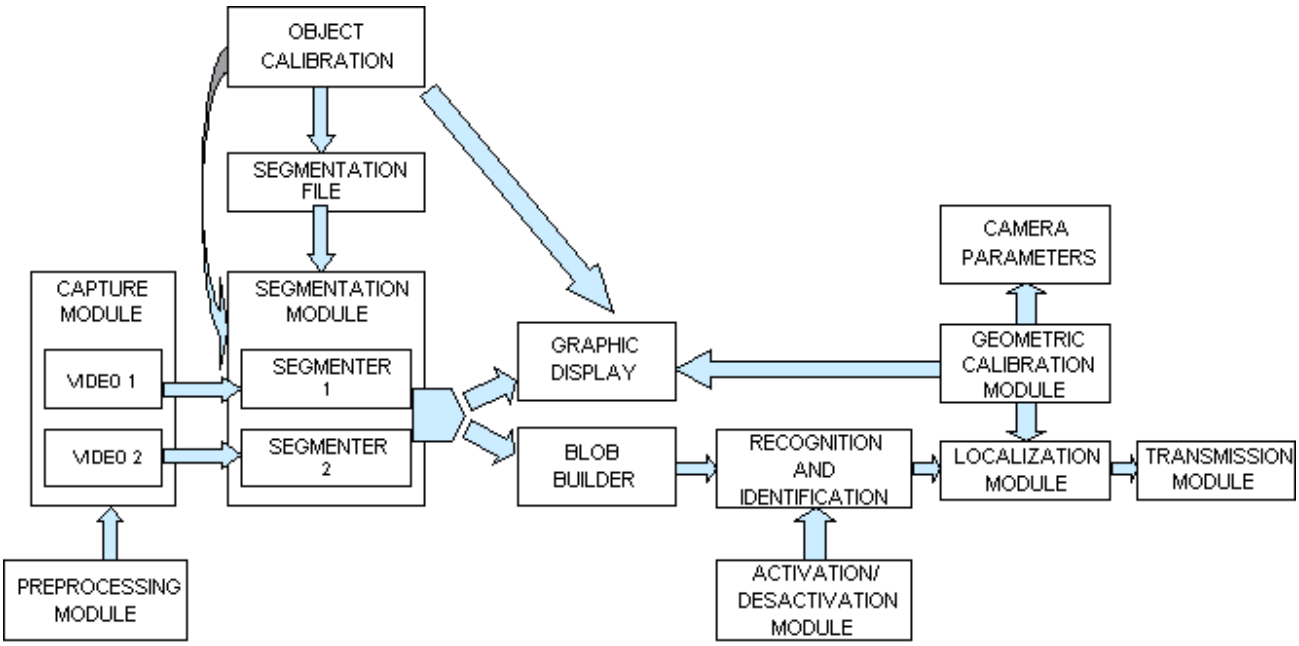

Figure 2. Vision system architecture

the image. Also computes the orientation of robots in the team.

10. GRAPHIC DISPLAY MODULE. It's responsible for displaying video images in the screen and for generating basic drawing functions such as lines, circles, etc. in the video image.
11. TRANSMISSION MODULE. A UDP network link is setup for the communication between the vision system an the AI system. The module builds a structure appropriate for data transmission. In practice the vision system can perform communication with two or more hosts allowing for distributed AI system processing if necessary.

Some modules such as the capture and graphic display modules rely on the operating system chosen for the implementation because they are tightly coupled with the hardware. We choose Windows 2000 because it is a well known operating system used by many as the primary programming platform. Nevertheless the algorithms of the rest of the modules can be programmed in any platform. In the next section we present the main algorithms of the vision system.

III. MAIN ALGORITHMS

So far we have explained the modules in the vision system but haven't explain the details of the algorithms used for video processing. This section gives a general overview with references given to further detailed explanation.

A. SEGMENTATION ALGORITHM

The most important step in video processing is assigning to each pixel in the image a class (object) that belongs to. This is equivalent to subdivide the image into the parts or objects that constitutes it [4]. The goal of the segmentation algorithm is to make this classification in real time. We use a method proposed in [5] that can be best described as a constant thresholding in a projected color space. The approach involves the use of thresholds in a three dimensional color space, such as RGB, YUV or HSI. We chose YUV because is more robust to changes in light intensity and is supported by most of the video capture cards.

Each class (object of interest defined by a color) is specified as a set of six thresholds values: two for every color space component. That is the maximum and minimum values for the range of each component in the color space.

The classification operation evaluates if a pixel belongs to a class or not, thus it is important to pay special attention to its implementation because this operation will be repeated for every pixel in the image; so with a 640x480 resolution the operation is done 307,200 times.

One naïve approach to evaluate if a pixel with values Y,U,V belongs to a class is:

```
if((Y >= Ymin)
   AND (Y <= Ymax)
   AND (U >= Umin)
   AND (U <= Umax)
   AND (V >= Vmin)
   AND (V <= Vmax))
   Pixel_color = color_class;
```

Unfortunately this approach is quite inefficient because needs up to 6 conditional branches to determine membership of the pixel to the color class.

The algorithm employed uses a boolean valued decomposition for the operation. The decomposition is stored in arrays, one array for every color component in the color space with one array element for each value of the color component. With this approach the membership of a pixel to a class can be evaluated using a bitwise AND operation of the elements of each array indicated by the values of the pixel:

```
pixel_in_class  = Yarray[Y]
                  AND Uarray[U]
                  AND Varray[V];
```

The resulting value in `pixel_in_class` indicates if the pixel belongs to the class or not. The next example helps to illustrate this approach.

The color space is discretized in 10 levels for each component. Then a color "O" might be represented assigning the next values for every element in the arrays:

```
Yarray[] = {0,1,1,1,1,1,1,1,1,1}
Uarray[] = {0,0,0,0,0,0,0,1,1,1}
Varray[] = {0,0,0,0,0,0,0,1,1,1}
```

To verify if a pixel with values (1,8,9) belongs to the color class "O" we need to evaluate the expression `Yarray[1] AND Uarray[8] AND Varray[9]` resulting in 1 indicating that the pixel belongs to the color class.

The bitwise AND operation used is significantly more efficient than the naïve approach, beside, parallelism in the integer AND operation can be exploited to evaluate the membership of the pixel to several color classes at the same time. Lets say that another color "B" can be represented assigning the next values to the arrays:

```
Yarray[] = {0,1,1,1,1,1,1,1,1,1}
Uarray[] = {1,1,1,0,0,0,0,0,0,0}
Varray[] = {0,0,0,1,1,1,0,0,0,0}
```

Instead of using another set of arrays we can combine them using each bit position in the elements of the array to represent a color. Combining the colors "O" and "B" we get:

```
Yarray[] = {00,11,11,11,11,11,11,11,11,11}
Uarray[] = {01,01,01,00,00,00,00,10,10,10}
Varray[] = {00,00,00,01,01,01,00,10,10,10}
```

The high order bit represents color “O” while the second bit represents de color “B”. We can check whether the pixel with values (1,8,9) belongs to one of the two classes by evaluating the same expression `Yarray[1] AND Uarray[8] AND Varray[9]`. The result is 10 indicating that it belongs to color “O” but not to “B” color. With a 32 bit integer a maximum of 32 colors classes can be segmented in the same operation.

The proposed method relies strongly in selecting appropriate thresholds values for each color of interest in order to obtain a good segmentation. This is probably the most critical step for any vision system based in colors. Our calibration module allows to change dynamically every threshold value while watching the effect of the change in real time. This approach has shown to be a fast method to select values, but is not very reliable when similar colors need to be calibrated , for example with bright blue and cyan, or dark pink and orange. An approach not implemented in our vision system that can be employed to avoid this problem is to compute an histogram and select the corresponding thresholds more accurately.

B. BLOB GENERATION ALGORITHM

After segmentation, the next step is to connect the segmented pixels into regions or blobs. This process can be expensive and can impact in real time performance. The selected method was proposed in [7] and is divided in two steps.

The first step is to compute a RLE of the segmented image. A RLE (Run Length Encoding) is a well known compression algorithm without loss [6]. The basic idea is to take sequences of repeated data and replace them with the data and their quantity.
In computer vision applications changes in adjacent pixels are uncommon and thus a RLE can reduce significantly the amount of data processed in the next step; beside there is also the practical benefit that the blob generation algorithm will only need to look for vertical connectivity, because the horizontal components are merged in the computation of the RLE. Every run generates a data structure that contains all the information of the pixels and a pointer used to connect them with other runs.

The objective of the merging procedure is to connect all the runs that belongs to a region. The idea is that every run of a region points to the region’s run parent. Initially we have a disjoint forest of runs, every run points to itself. The merging procedure looks into adjacent rows and merge runs that are of the same color and overlap under four connectedness. Every run points towards a region’s global parent, if overlapping occurs then a second pass is needed to merge the overlapped runs. The process is shown in Figure 3.

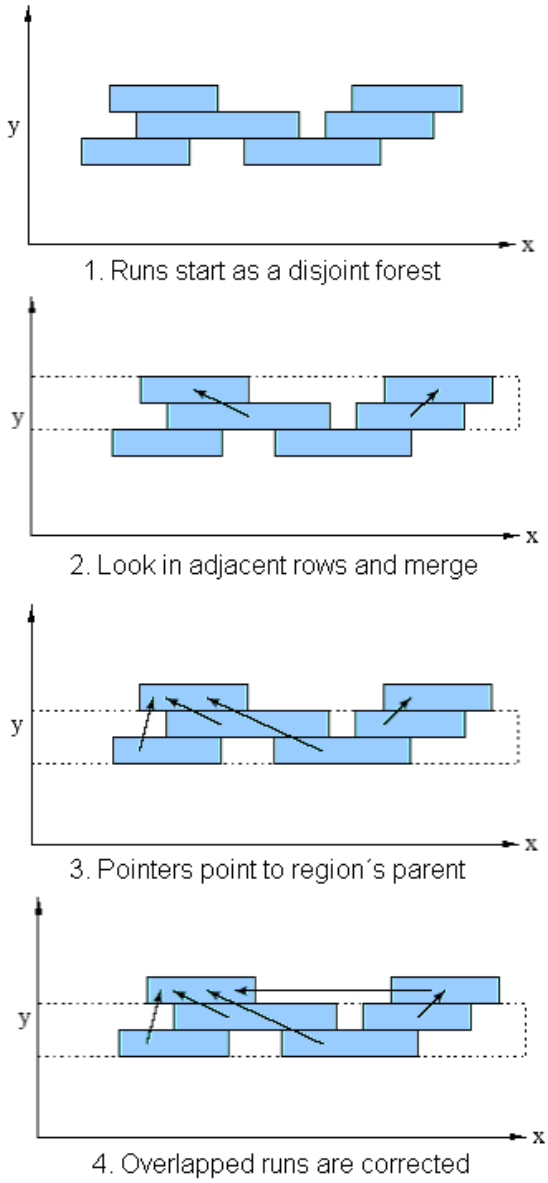

Figure 3. Merging process example

Relevant information such as the area, centroid, bounding box and perimeter are computed in the last step. This information is stored in a region data structure. Lists of regions for each color are generated at the end of the method.

C. IDENTIFICATION ALGORITHM

Once the lists of regions are generated the recognition process begins. As mentioned before, to recognize the ball we simply take the region of orange color whose area is nearest to 85 pixels (ball area with a 640x480 resolution). To do this we

order the list with Quicksort [8] using not the area of the region but instead the absolute value of the difference between the region's area and 85. The first element of the ordered list is the ball.

To recognize robots of the opposite team we follow a similar procedure, but now the order criteria is the difference between the region's area and 115 pixels. We take the first *n* elements of the ordered list, where *n* is set by the user in the activation/desactivation module.

To recognize our robots we first select candidates following the same procedure as the robots of the opposite team. Then every candidate is further examined searching for additional patches used in the robots for individual identification. Every team decides which pattern of patches to use; we decided to use a butterfly pattern [9]. We search the additional patches in a window around the central patch (candidate). If we found less than four additional patches the candidate is discarded, if we find more we select the four regions whose area is nearest to 115 pixel. Once we have four patches in the window area around the central patch the next step is to order them in a counter clockwise direction to obtain the color encoded ID of the robot. Every robot has a unique ID encoded in the additional patches, one color (green) is interpreted as a cero and the other color (pink) as a one. So if we have a patch pattern of green, green, pink and green we interpret it as 0010 or a decimal 2.

To order the additional patches in counter clockwise direction we first compute a simple close path [10]. A simple close path is found when a set of points are connected without crossing the paths between the points. Figure 4 shows an example of a simple closed path.

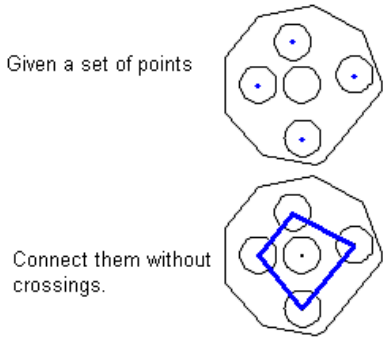

Figure 4. Simple closed path

The first step to produce the path is to find the lowest point and take it as anchor, then the angles of the line segments between all the points and the anchor are computed. The points are ordered form smaller to greater angle. Figure 5 shows the anchor point A, points P_i, line segments and angles computed.

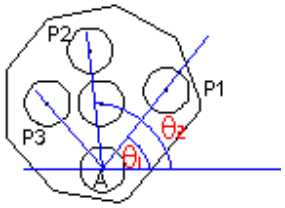

Figure 5. Simple close path computation

Once the patches are ordered the distance between the points are computed, the pair of points whose distance is greater correspond to the frontal patches and the ID can be decoded.

D. LOCALIZATION AND ORIENTATION ALGORITHM

When the ball or the robots have been recognize their position must be calculated. The first step is to compensate for the distortion introduced by the camera lenses. The geometric calibration module calculates the camera internal and external parameters. To undistort the centroid of the region we use the Tsai [3] equations:

$$\breve{x} = x + x\left[k_1 r^2 + k_2 r^4\right] + \left[2p_1 xy + p_2\left(r^2 + 2x^2\right)\right]$$

$$\breve{y} = y + y\left[k_1 r^2 + k_2 r^4\right] + \left[2p_2 xy + p_2\left(r^2 + 2y^2\right)\right]$$

where:

k_1, k_2 are the radial distortion coefficients,

p_1, p_2 are the tangential distortion coefficients,

(x, y) are the coordinate without distortion,

$(\breve{x}, \breve{y})$ are the coordinate with distortion and

$r^2 = x^2 + y^2$.

With the undistorted coordinates of the object the next step is to calculate their position in the field. We use five known reference points in the field (the corners of the first or second half of the field depending on the camera and the center of the field). With these reference points we calculate the true position of the object in the field computing the differences between the reference points and the undistorted point.

To calculate the orientation of our robots we undistort the coordinates of the centroid of the frontal patches and compute the midpoint of the line segment between these two points, then the angle formed by the midpoint is calculated and the central patch is computed, this angle is the orientation of the robot.

IV. SYSTEM IMPLEMENTATION

The vision system was implemented in a regular PC: Pentium 4, 2.00 GHz, 512 MB of RAM with two off-the- shelf video

capture cards (based in the popular BT878 chipset) under Windows 2000. We use two low cost handycams and Svideo cables. The programming environment was VC++ 6.0.

We implement DirectShow[10] components that do all video processing. DirectShow is the Microsoft architecture for video and audio. It allows access to all the streaming media capabilities installed in the PC and provides an API for programmers to develop custom filters.
We took the decision to use this software tool for two main reasons, first, DirectShow is a free programming API that allows to isolate the hardware installed in the PC from the vision system, making the system hardware universal (any video capture card should work with the vision system); second, the modular nature of DirectShow components integrate easily with the system design.

Every step of the processing is done in a COM object called filter. A filter perform some operation in the streaming media. The application needs to construct a graph of filters (chain of filters) to achieve the processing.

We use OpenCV[11] to make the camera calibration procedure. The OpenCV camera calibration filter uses a chess pattern to automatically extract the camera's parameters, this filter is integrated in the application graph when necessary.

The complete source code of the vision system can be found in the Robotics Lab web site (http://robotica.itam.mx).

V. RESULTS

The vision system has been used in two international competitions: RoboCup American Open 2003 and USOpen 2004, where our team Eagle Knights had won 3rd and 2nd place respectively.
The system works at 58.96 fps with two video signals at a resolution of 640x480.
Table 1 shows the results in different tests done to the vision system.

Test name	**Error**
Ball not found or misidentified	0.12%
Opposite robot not found or misidentified	0.08%
Team robot not found	0.13%
Team robot misidentified	0.02%
Position	0.01%
Robot orientation	2%

Table 1. Results in tests.

VI. CONCLUSIONS

We have presented the design, algorithms and implementation of our vision system pointing out the areas that we believe are critical for understanding the general structure of the system presenting the architecture of the system and giving a brief description of their functionality. Our mayor emphasis has been in the main algorithms because they constitute the core of the system. The segmentation algorithm is optimized for real time performance and have been probed in our team and many others with great success. The blob generation algorithm is simple but yet strong avoiding the use of recursive functions that can eat memory resources. The identification algorithm is fast and reliable and could be use with other patch patterns with some modifications. We gave a general idea of the software tools used for the implementation giving references for a deeper study and releasing the source code of our system.

VII. REFERENCES

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife, 1995.

[2] Verret, Ball, Kiat Ng. Laws of the F180 League - Release 3.00a. http://www.itee.uq.edu.au/~wyeth/F180%20Rules/index.htm.

[3] Tsai, R.Y. A versatile camera calibration technique for high accuracy 3D machine vision using off-the-shell TV cameras and lenses. IEEE Journal of robotics and Automation, 1987.

[4] González, Woods. Digital Image Processing. Adison-Wesley Publishing Company, 1993.

[5] Bruce, Balch, Veloso. Fast an inexpensive color image segmentation for interactive robots. Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[6] Held, Gilbert. Data Compression: Techniques and Applications, Hardware and Software Considerations. John Wiley & Sons, 1987.

[7] Bruce, J. Realtime Machine Vision Perception and Prediction. Undergraduate thesis. Carnegie Mellon University, 2000. http://www-2.cs.cmu.edu/~jbruce/cmvision/papers/JBThesis00.pdf

[8] Hoare, C.A.R. Communications of the ACM. ACM Press, 1961. http://portal.acm.org/citation.cfm?id=366644&dl=ACM&coll=portal

[9] James Bruce and Manuela Veloso. Fast and Accurate Vision-Based Pattern Detection and Identification. In Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation, Taiwan, May 2003.

[10] MSDN Library. Introduction to DirectShow. DirectShow SDK Documentation, 2004. http://msdn.microsoft.com/library/default.asp?urt=/library/en-us/directx9_c/directX/htm/directshow.asp

[11] Intel Corporation. Open Source Computer Vision Library, Reference Manual. Intel Corporation, 2001. http://www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf