TLIB: a real-time computer vision library for HCI

Sébastien Grange, Terrence Fong, and Charles Baur

Virtual Reality and Active Interfaces (VRAI) Group
Institut de production et robotique, Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
{sebastien.grange, terrence.fong, charles.baur}@epfl.ch

Abstract. A computer vision software library is a key component of vision-based applications. While there are several existing libraries, most are large and comple or limited to a particular hardware/platform combination. These factors tend to impede the development of research applications, especially for non-computer vision experts. To address this issue, we have developed TLIB, an easy-to-learn, easy-to-use software library that provides a complete set of real-time computer vision functions, including image acquisition, 2D/3D image processing, and visualization. In this paper, we present the motivation for TLIB and its design. We then summarize some of the applications that have been developed with TLIB, and discuss directions for future work.

1. Introduction

Since 1999, the EPFL Virtual Reality and Active Interfaces (VRAI) Group has been developing non-traditional human-computer interfaces in a variety of fields, including computer assisted surgery (CAS) and mobile robotics. These interfaces exploit numerous interaction techniques based on Computer Vision (CV), such as activity monitoring, human detection and tracking, and gesture recognition. The VRAI group, for example, is part of the CO-ME network [1], which focuses on the application of information technology to medical treatment. One area of interest is increasing the use of computer equipment in the operating room (OR). Because OR's are crowded environments and have stringent sterility requirements, traditional computer input devices (i.e., keyboard and mouse) are problematic. Thus, there is a significant need to develop non-contact (i.e., vision-based) interaction methods.

Another area we have been investigating is teleoperation interfaces for mobile robots. Traditional remote driving systems are based on hand-controllers and video displays. Such systems are error-prone, time consuming to deploy, and difficult to use when joint (i.e., human-robot) task performance is required. An alternative is to enable the human to interact directly with the robot, e.g., communicating commands through hand gestures. Autonomous robots however, typically have limited processing power. Hence, the vision system must be efficient and lightweight.

Finally, many of our research projects involve designing and demonstrating application prototypes. Very often, undergraduate students with limited programming and computer vision experience contribute to such developments as part of their degree program. For these students to be productive, it is important to minimize the overhead associated with learning and implementing fundamental computer vision methods, such as camera calibration and pixel operators.

To address these needs, we have developed the EPFL Tracking LIBrary (TLIB), a software library for computer vision. TLIB provides a structured, object-oriented framework for rapid prototyping and development of real-time, computer vision applications. TLIB incorporates image acquisition, 2D/3D image processing functions, multiple color spaces, and 2D display routines. TLIB is written in C/C++ and is largely hardware and operating system independence.

In the following sections, we describe the structure of TLIB and its capabilities. We then discuss how TLIB differs from other computer vision libraries. Finally, we summarize some of the applications that have been developed using TLIB, and present on-going and planned improvements.

2. TLIB Overview

2.1 History

In 1999, we developed a collection of real-time, 2D image processing functions called VisLib. This open-source C library provided color-based methods for object detection and tracking and was designed primarily for the mobile robot research community. VisLib is currently distributed and maintained by ActivMedia Robotics, LLC[1].

The success of Vislib and the need for a richer set of vision-processing routines motivated us to develop a new library. Our aim was to create a flexible, easy-to-use library that would facilitate development of Vision-Based Interfaces (VBI). TLIB is the outcome of this development.

2.2 Design principles

Ease-of-use/learning. TLIB's architecture and interface provides an easy, consistent and well-documented programming interface. This is made possible by a simple, high-level API that encapsulates highly-optimized, low-level code.

Efficiency. While real-time computer vision usually requires highly optimized code, optimization is difficult and time consuming for non-experts. Our objective was to provide highly tuned functions so that the application programmer need not be concerned with low-level optimization.

Portability. Significant emphasis was placed on platform-independent design. TLIB's API and internal code was explicitly designed to facilitate porting. All non-portable code is isolated and accessed through portable data structures.

VBI-focused. Library capabilities were chosen to facilitate the development of VBI applications. As a result, TLIB includes functions for stereo vision, skin-color modeling and human feature detection and tracking.

2.3 Features

TLIB is intended to enable vision-based human-computer interaction in a variety of settings. Thus, it supports monocular and stereo cameras, fixed and moving viewing angles, and a wide range of processing hardware (both embedded and desktop).

Easy-to-use interface. TLIB provides a "simple", high-level programming interface. All of TLIB's high-level methods have been designed with default parameters optimized for vision-based HCI. In practice, this means that these methods can be invoked in most situations using few (or no) parameters. In addition, TLIB is designed to automatically perform format conversion and run-time sanity checking to reduce coding errors. For example, the code below shows a complete TLIB program to continuously capture, edge detect and display a camera image.

Simple image processing with TLIB:capture, Sobel edge detect, and display.

```
int main (int argc, char **argv)
{
  tlDigitizerDX *digitizer = new tlDigitizerDX ();
  tlDisplay *window = new tlDisplay ();
  tlImage *image = new tlImage ();
  tlImage *imageEdge = new tlImage ();
  while (tlDisplay::eventQuery() != TL_EXIT) {
    digitizer->grab (image);
    image->edges (imageEdge, TL_EDGES_SOBEL);
    window->display (imageEdge);
  }
}
```

We have found that a "simple" programming interface provides substantial benefits: it facilitates code development; it improves code readability; and it aids code understanding. In our experience, a novice programmer without background in computer vision can be proficient with TLIB after only 1-2 hours of use.

Real-time. For portability reasons, TLIB relies only on hardware-independent code optimization to achieve real-time performance. The following is a partial list of the code optimization techniques used in TLIB, based primarily on guidelines from [16]:

- Loop unrolling
- Invariants in loop processing
- Few, small function arguments
- Function inlining
- Minimizing memory allocation (including heap use)
- Math function approximations

Portable. TLIB is written in ANSI C and C++, and is largely independent of OS-specific libraries. Thus, TLIB can be ported to other platforms with little effort. The current version supports both Windows and Linux. The high-level hardware classes can be rewritten easily for different platforms or for hardware components that do not support generic drivers. TLIB also comes with a set of example programs that compile transparently on both Windows and Linux.

Use of multiple sources. TLIB works with a variety of image acquisition hardware and image formats. The TLIB image formats support several standards and provide a wrapper for the SRI SVS stereo engine [13]. It is also supports custom image formats from multiple sources including frame-grabbers, image files, and video files.

2.4 Structure

TLIB library is implemented as a set of optimized ANSI C functions wrapped by higher-level C++ objects. Figure 1 shows the class structure and inheritance diagram between TLIB classes.

2.4.1 C++ wrappers

Table 1 lists some of the high-level classes available and their functionality. Methods in each class incorporate data format and geometry consistency checks. This allows the optimized, low-level functions to execute without safeguards (i.e. under the assumption that all parameters are valid).

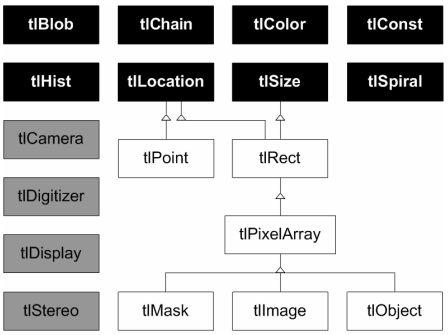


Fig. 1. TLIB image class inheritance diagram (black = base classes, gray = wrapper classes)

Table 1. High-level C++ functions categories

Base classes	
tlBlobs	Blob extraction and filtering
tlCamera	Base class for camera and image sensors
tlDigitizer	Image acquisition
tlLocation	Location object
tlPixelArray	Pixel array data and processing
tlSource	Image source base class for TLIB
Derived classes	
tlCameraSVS	SVS [14] stereo camera class
tlCameraTSAI	TSAI [19] camera calibration class
tlColor	Color format description class
tlDisplay	Display and event management class
tlHist	Histogram implementation and tools
tlImage	Image data and processing
tlMask	Binary mask for image processing routines
tlObject	Object description and tracking methods

2.4.2 C functions

C functions are the core of TLIB. They implement optimized, low-level operators on specific data types. These functions are grouped into the following categories:

Table 2. Low-level C functions categories

Timer functions	Platform-independent, high-resolution timing routines
Channel operators	Add, remove and switch image channels
Convolution operators	Arbitrary kernels (separable and non-separable)
Edges operators	Separable extraction methods
Object extraction operators	Eegion growing and color filter based extraction
File read/write	Various image formats (JPEG, PPM, BMP, etc.)
Filter operators	Color and depth filtering
Template matching operators	SSD operators
Format conversion operators	Various color spaces (RGB, HSI, rgb, etc.)
Memory operators	Memory allocation and copy
Morphological operators	Dilation, erosion, etc.
Pixel operators	Addition, subtraction and scaling

3. Related Work

The Computer Vision Homepage [3] cites more than 100 image processing libraries currently available from academic and commercial sources. Clearly, not all of these software packages are designed for real-time applications. In fact, fewer than a dozen can be characterized as "real-time". Moreover, many libraries are highly operating system dependent (generally Windows or UNIX only).

OpenCV [12] is the most widely used real-time library in the computer vision community. OpenCV provides an extensive set of vision algorithms, collected from many different sources. As such, it is a "large" library, with a complex interface. Furthermore, the large number of supported features results in significant memory requirements that are not always appropriate for embedded applications. Thus, while OpenCV is a versatile and powerful library, it requires time to learn and is too complex for short-term student.

The Microsoft Vision SDK [17] is a real-time image manipulation and analysis library. It is a low-level library intended primarily to provide a programming foundation for research applications. The Vision SDK is highly Windows-specific. It relies on Microsoft data types and interface standards such as VFW (Video For Windows) and DirectX. While it offers an extensive image definition, the Vision SDK lacks the basic image processing methods needed for real-time HCI applications

In comparison, to both OpenCV and the Vision SDK, TLIB is designed to support a limited set of vision applications and has a significantly easier-to-use API. For example, Table 3 lists the members of a basic image data structure in all three libraries. In our experience, TLIB's simplified API greatly facilitates rapid-prototyping, without overly restricting application flexibility and capability.

Table 3. Public image data structures in Microsoft VisSDK, Intel OpenCV, and TLIB

```
// Microsoft Vision SDK
CVisShape m_shapeImage;
EVisPixFmt m_evispixfmt;
CVisShape m_shapeMemBlock;
CVisMemBlock m_memblock;
std::string m_strName;
CVisStreamHandler*
    m_pVisStreamHandler;
UINT m_uStreamFrameNum;
int m_cbPixel;
int m_cbCol;
int m_cbRow;
BYTE *m_pbDataOrigin;
BYTE *m_pbFirstPixelInRow0;
HDC m_hdc;
HBITMAP m_hbitmapOld;
FILETIME m_filetime;
bool m_fDirty;
bool m_fDelayRead;
bool m_fReserved;
bool m_fUseColorMap;
CVisMemBlock
     m_memblockColorMap;
int m_imopts;
BYTE *m_pbOriginIliffe;
BYTE **m_ppbIliffe;
CVisMemBlockOf<BYTE *>
    m_memblockIliffe;
CVisPropList m_proplist;
```

```
// TLIB
int           width;
int           height;
int           pixelWidth;
tlPixel*      pixel;
tl_format     format;
unsigned long timeStamp;
_____________________________
// OpenCV
int  nSize;
int  ID;
int  nChannels;
int  alphaChannel;
int  depth;
char colorModel[4];
char channelSeq[4];
int  dataOrder;
int  origin;
int  align;
int  width;
int  height;
struct _IplROI *roi;
struct _IplImage *maskROI;
void  *imageId;
struct _IplTileInfo
     *tileInfo;
int  imageSize;
char *imageData;
int  widthStep;
int  BorderMode[4];
int  BorderConst[4];
char *imageDataOrigin;
```

Many CV researchers use Mathworks' Matlab [15] to develop their algorithms before moving to traditional programming languages such as C or C++. The major benefit of Matlab is that it offers an image processing toolbox with many operators. Matlab functionality is accessible in high-level calls, which makes development rapid and easy. The primary drawback of Matlab is that it is not designed for real-time or stand-alone application development. TLIB is designed to offer a high-level interface similar to Matlab that has real-time performance (though with reduced functionality).

Several researchers have developed libraries specifically for vision-based HCI[6][11]. These libraries, however, tend to be application-specific or designed to investigate a particular software design. As such, these libraries are not suitable for distribution as a CV tool to a large community of developers.

4. VBI Applications

We have used TLIB to develop numerous VBI and as an educational tool in several undergraduate and master's-level projects. The following summarizes some of our recent work.

4.1 Human-Oriented Tracking (HOT)

The objective of this project was to create a reusable software architecture to support vision-based HCI applications. This architecture provides applications with information about a person's location, pose and behavior. Such information can then be used to perform more meaningful interaction. This project was the first to use TLIB and was specifically designed for "intelligent environment" (or "smart space") applications [9].

4.1.1 Human Detection and Tracking

In [9], we used TLIB to implement a vision-based people tracker, which creates a geometric and dynamic model of a person. The tracker uses sensor fusion from two input modalities, namely color images and stereovision, to locate particular human features. A model of the human pose is then built from the tracking results, and human movements are segmented and parameterized.

HOT is designed to detect and track the head and hands of a human user. Figure 2 shows the output of simple TLIB filters applied to color and depth images, which led to a pre-processed image that is then used to locate human features. The detection and tracking algorithms use TLIB blob-processing functions combined with geometric constraints from depth information. Figure 4 shows the result of the human feature detection, with vectors indicating segmented hand movements.

Fig. 2. Left to right: normalized color filter, stereo image, and fusion of the two

During our tests, head detection and tracking proved to be fast and reliable, running at more than 20 Hz on a standard office PC (Pentium III, 700 MHz). We found that hand tracking was not as effective, particularly during rapid movements, due to frequent changes

in shape and illumination. Overall performance, however, was adequate for several demonstration applications including a virtual whiteboard and simple robot control [5].

In [14], we performed arm pose recognition. First, we used a combination of color filtering and stereo segmentation to detect head position. Then, a kinematic model of the user's arms was matched to stereo data using multiple depth histograms. An example of arm pose tracking is shown in Figure 3.

Fig. 3. Fitting of two-arm model and hand model with stereo data

In [4], a histogram matching technique was used to build a model of the human hand and fingers in real-time, based on color and stereo segmentation. This method is user-independent and robust, leading to a recognition accuracy of different static finger poses greater than 90%. Figure 3 shows the detection and hand model construction for one of the hand postures.

4.1.2 Activity Monitoring

One way to enhance HCI applications is to monitor the human to ascertain information about the task he is performing, his current locus of attention, his mood, etc. People use this information naturally during peer-to-peer interaction, especially during conversation and joint task performance.

In [9], we developed a method to characterize human activity based on a set of pre-defined, or a priori, parameters. We believe such parametric activity monitoring is more useful for detecting a broad range of activities than task-specific metrics. Using the TLIB human feature detection and tracking process described above, motion "quantifiers" (e.g., relative body displacement) for each tracked feature were computed.

Experiments showed that it was possible to accurately classify general level of physical activity (sitting, walking, "doing something with his right hand", etc.) Figure 4 shows an example of the activity monitor output. Each histogram represents the level of activity for a particular feature (different colors indicate different time scales).

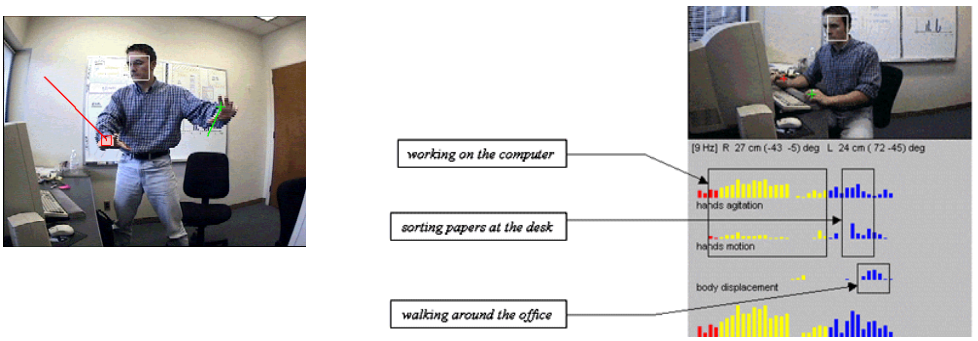


Fig. 4. Human features detection and activity monitoring based on TLIB

4.2 Medical Interfaces

We recently developed a computer vision system to replace standard mouse functions with hand gestures [6]. This application is relevant to the recent introduction of computerized tools in the operating room, since surgeons must have easy control over computers without compromising the sterility of the operating field. Such non-contact, gesture-based user interfaces are a promising candidate that we are developing in collaboration with surgeons at partner hospitals.

The system uses color stereo cameras to detect motion in a 3D workspace and to interpret hand movements as mouse commands (pointer movement and button presses). Because the system requires an unobstructed line-of-sight between the cameras and surgeon, it is most appropriate for minimally invasive surgery (MIS). Figure 5 gives an overview of the setup used.

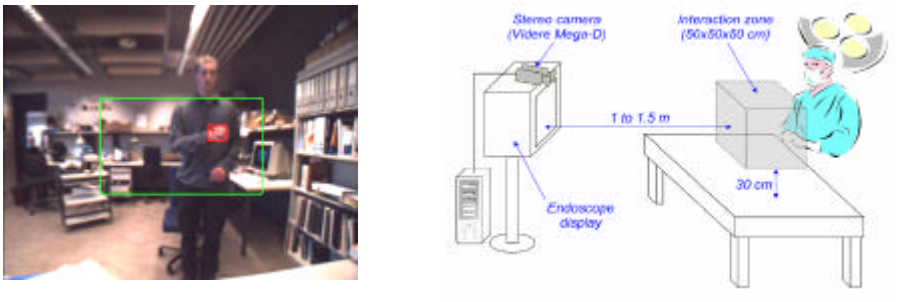


Fig. 5. Hand detection and non-contact mouse for medical applications

Detection is performed using color filtering and range processing. Once a hand has been detected, we apply small-window correlation to track its movement. A Kalman filter is used to estimate hand velocity and to predict future hand position. Because the search is local, tracking is not disturbed by the presence of other hands (or objects similar to hands) inside the workspace. Figure 5 shows the result of the hand detection in a predefined volume of interest. The hand position is then converted into mouse positions on a standard PC display.

To evaluate the non-contact mouse system, we are now packaging the system for clinical testing at the Inselspital hospital in Bern, Switzerland. One approach we are considering is to perform all vision and gesture processing on a laptop computer and to output mouse commands via a serial cable (i.e., using a standard mouse communication protocol). This would allow us to connect our system to a wide range of computers.

4.3 Human-robot interaction

In [18], we developed a system to allow a person to interact with a mobile robot using hand gestures. To achieve this, the robot needed to detect people and recognize gestures. Given that the robot has limited processing power, the main difficulty was to design a fast recognition method that can operate in real-time with minimal resources. Figure 6 shows the robot setup used in our experiments. In particular, we used tilted camera geometry to limit the portion of the image that needed to be processed. To detect people, the stereo algorithm only processed a portion of the image. This proved to be a very robust and fast method for detecting and tracking people.

In order to extract head and hand position in a robust manner, we fused depth and color information using a depth-sorted histogram method. We then used a pre-defined set of static gestures to map head and hand position to robot motions. Figure 7 shows two of the static gestures that the system was capable of recognizing.

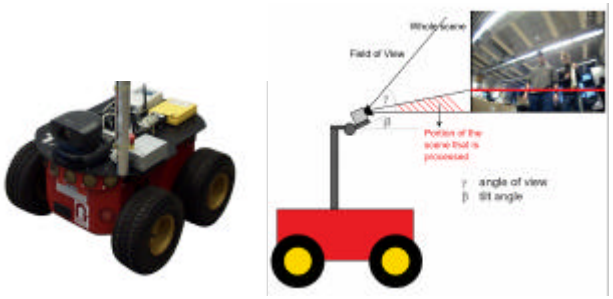
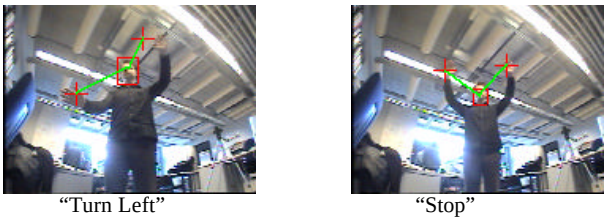


Fig. 6. Mobile robot and its configuration



"Turn Left" "Stop"

Fig. 7. Static gesture recognition for robot motion control

5. Future Work

We are currently evaluating several new VBI techniques for use in TLIB. These include an adaptive background object that uses disparity maps to learn the environment and a statistic color model that can be trained to recognize colors in different image formats.

The development of TLIB is currently being driven by our work in medical UI. A primary focus is to add tools and techniques for dynamic gesture recognition. Thus, we are now integrating a HMM-based method for gesture detection and classification [10].

Another possible addition to TLIB is an implementation of the face detection methods proposed in [20]. This functionality will complement the techniques based on color and disparity that are already contained in TLIB.

In addition, we plan to develop a variety of techniques to identify and distinguish individual users. In particular, we are considering the use of color histograms and facial features for user identification. We also have begun studying the possibility of porting TLIB to a Windows-CE based PDA, such as the Compaq iPAQ. This would allow us to develop highly portable, personal vision applications.

Finally, we intend to make TLIB publicly available for research in the near future. We believe that TLIB could prove a valuable open-source tool for learning CV, and for easily integrating VBI into existing, or new, applications.

6. Conclusion

In this paper, we have presented TLIB, an efficient, easy-to-use software library for real-time computer vision. TLIB provides a set of optimized routines wrapped in high-level classes and structure that make it both easy-to-learn and easy-to-use. TLIB has been used in

a range of HCI applications, including smart space environments, non-contact medical interfaces, and gesture-based robot control.

TLIB is highly portable, works with a wide range of hardware, and is well suited for researchers with little prior knowledge of computer vision. In this respect, TLIB is an ideal introduction to more complex vision libraries that offer greater functionality, but which are less portable and significantly harder to learn.

7. Acknowledgements

Our thanks to Emilio Casanova, Sébastien Dey, Ryan Garver, Chauncey Graetzel, Mathias Kolsch, Mikael Krummen, and Christian Wengert for contributing to TLIB. This work was partially supported by the Swiss National Science Foundation Computer Aided and Image Guided Medical Interventions (NCCR CO-ME) project.

8. References

1. ActivMedia Robotics, "VisLib, a high-performance vision processing library", http://robots.activmedia.com/vislib
2. Computer Aided and Image Guided Medical Interventions (CO-ME), http://www.co-me.ch/
3. Computer Vision Homepage, http://www-2.cs.cmu.edu/~cil/vision.html
4. Dey, S., Système de reconnaissance de posture de main, VRAI Group Technical Report, Swiss Federal Institute of Technology, Lausanne, Switzerland, February 2002.
5. Fong, T., Conti, F., Grange, S., and Baur, C., Novel Interfaces for Remote Driving, SPIE Telemanipulator and Telepresence Technologies VII, Boston, MA, November 2000.
6. François, A. R. J., Medioni, G. G., A Modular Software Architecture for Real-Time Video Processing, International Conference on Vision Systems, 2001.
7. Graetzel, C., Interface utilisateur basée sur les gestes visuels pour chirurgie, VRAI Group Technical Report, Swiss Federal Institute of Technology, Lausanne, Switzerland, February 2003.
8. Graetzel, C., Grange, S., Fong, T., and Baur, C., A Non-Contact Mouse for Surgeon-Computer Interaction, *NCCR-COME Research Networking Workshop*, Brauwald, Switzerland, August 2003.
9. Grange, S., Vision-based Sensor Fusion for Active Interfaces, Microengineering Diplôme, Swiss Federal Institute of Technology, Lausanne, Switzerland, March 2000.
10. Grange, S., Casanova, E., Fong, T., and Baur, C., Vision-based Sensor Fusion for Human-Computer Interaction, IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, October 2002.
11. Hernández, M., Cabrera, J., Castrillón, M. Domínguez, A., Guerra, C., Isern, J., An Active Vision System for Detection, Tracking and Recognition, International Conference Vision Systems, 1999.
12. Intel. Open Source Computer Vision Library (OpenCV), http://www.intel.com/research/mrl/research/opencv
13. Konolige K., Small Vision Systems: Hardware and Implementation, Eighth International Symposium on Robotics Research, Hayama, Japan, October 1997.
14. Krummen, M., Gesture Recognition based on Stereo Vision, VRAI Group TR, Swiss Federal Institute of Technology, Lausanne, Switzerland, February 2003.
15. Mathworks, Matlab, http://www.mathworks.com
16. McConnell, S., Code Complete, Microsoft Press, 1993.
17. Microsoft VisionSDK Homepage, http://research.microsoft.com/projects/VisSDK
18. Poell, B. and Wengert, C., Human Oriented Tracking and Mobile Robot Gesture Driving, VRAI Group TR, Swiss Federal Institute of Technology, Lausanne, Switzerland, February 2002.
19. Tsai, R., An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision, Computer Vision and Pattern Recognition, Miami Beach, Florida, 1986.
20. Viola, P. and Jones, M., Robust Real-time Object Detection, International Journal of Computer Vision, 2002.