

Image Processing Framework for Traffic Monitoring

Peter Major, *Student*,
Department of Mechatronics, Optics and Information Engineering
Budapest University of Technology and Economics

Abstract—This paper describes an image processing framework capable of tracking moving objects in two or three dimensions. The objects are observed by multiple cameras and their path can be recorded without markers, special cameras or background. This paper shows how position coordinates can be acquired from the image of objects. The moving regions of a frame are specified by the difference between the frame and an adaptive background. The adaptive background is calculated from a sequence of frames, meaning that the program can also work in changing light conditions and can handle slow-moving objects. Afterwards, the identified moving regions are tracked inter-frame. The positions of objects received from cameras are then converted to a common world space coordinate system. At this point, the position coordinates are ready for further processing. The program uses a wide array of technological tools, such as hardware acceleration, multi-threaded processing, three dimensional display of objects and network data transmission among the cameras. An integrated framework has also been developed, which is easily customizable, and can be adapted to specific tasks.

I. INTRODUCTION

NOWADAYS most of desktop computers are so powerful that they are capable of running image processing programs. With the spread of multi-processor (or multicore) computers and GPUs - which are optimal for executing SIMD style algorithms - more and more consumer and industrial solutions have the opportunity to use artificial vision for an array of tasks.

As traffic congestion has become a serious problem a lot of traffic measurement techniques were implemented. Some of them use local sensors, such as inductive loops [3], microwave radars, infrared [2] or acoustic [1] sensors etc. These solutions could be robust; however their construction needs and costs are usually higher. The use of video surveillance can solve this, as the deployment of these devices is more cost-effective, and a small number of cameras can observe multiple roads, lanes or a square. Moreover we can also track real 3D movements, such as flying birds, and living creatures, such as running animals, which could have a great use in ethology and other areas. Although the development started as strictly car traffic monitoring, later we saw that the software could solve more generalized problems by splitting the processing steps to exchangeable modules, and the development shifted to

observe smaller amount of objects with greater precision.

The two key points of tracking objects in 3D are that first we need to recognize objects from standalone video sources, and then to pair the objects among these sources. The recognition of moving objects can be done a lot of different ways. Some of them use high resolution images with smaller number of colors, such as [4] and [5], which use gray scale or even black-and-white frames for reducing calculation needs, others (such as [6]) choose to keep colors for better recognition. In this system I choose to use a small and a high resolution 24bit color image. This allows better recognition with the bigger copy, and less precision demanding parts can use the smaller one. The system also makes use of multi-processor computers through parallelization, and GPGPU through HLSL image filters. The recognition of moving objects can be also different, some use difference between the current frame and an adaptive background [4], others employ edge [7] and/or blob detection [6]. The modules created so far use an adaptive background subtraction method, however the system is ready to accept edge detection based modules if needed.

The objects seen by cameras are coupled using their possible positions in three dimensional space and their images if needed. This step however is not solved within the same process, as an independent data fusion application has been developed. Therefore I use one instance of the image processing application for each camera and one fusion application for tracking objects in 3D. These communicate over network.

Instead of expensive industrial cameras and other equipment I used low-cost consumer webcams and a midrange computer for the research.

The contents of this paper are the following: Section II describes the internal structure of the image processing framework, Section III details the procedures of image processing, Section IV points out the applications of this software and a conclusion summarizes the results of this research.

II. THE FRAMEWORK

As the development started, it became clear that there are a lot of tasks that almost every image processing application does the same way. If we create a framework we can reuse the code that solves these common tasks for a lot of different problems. The framework handles the cameras, removes distortion, manages the processing threads, writes a log of the programs events, provides a pipeline builder etc. The system

Manuscript received February 28, 2011. This work was supported in part by the Department of Mechatronics, Optics and Information Engineering, Budapest University of Technology and Economics.

P. Major is a student at Budapest University of Technology and Economics, 1111 Budapest, Hungary (e-mail: axodox@hotmail.com).

is built on the .Net framework, and uses XNA for DirectX based graphical output [11]. The project was developed mostly in C#, the performance sensitive parts and interoperation regions in C++\CLI. The image filtering and data visualization is processed by HLSL shaders.

A. The Design Goals

The main objectives of designing this machine vision software and the framework were the following:

- 1) A group of stationary cameras observe moving objects. Any of the moving entities must be seen by at least two cameras at the same time.
- 2) The background of the objects is mostly invariable, although the slow change of light conditions, and stopped entities must be well tolerated.
- 3) The objects are distinct from the background, and most of the time they are moving and don't touch the edge of the image.

If these conditions are met, then the program can measure the positions of moving objects. The reusability and performance of the code was a main concern in development, especially the following:

- 1) The program is fully modular, so every operation on the frames is done in easily exchangeable modules. The queue of modules makes up a processing pipeline.
- 2) By using multi-threaded processing, the algorithm utilizes the power of multiprocessor computers.
- 3) The program makes use of GPU acceleration by offloading a large part of image processing from the CPU.
- 4) For every camera there is an independent process, these communicate with each other and one or more "inter-camera" processes through network.
- 5) The source of the video can be any Microsoft DirectShow compatible device (including most of webcams and video digitalization cards). For testing purposes the program can also read video files if the appropriate codecs are available.
- 6) The framework supports plug-ins, which can be developed completely independently from the main program, and easily loaded with a plug-in manager. This makes customization and extension simple, and for example a mathematician could write algorithms for the application without the knowledge of the full software, the handling of cameras etc. and even the parallel processing is transparent from the view of a module developer.

B. The Processing Stages



Fig. 1. The stages of image processing

The image processing consists of five stages (Fig. 1.). In the pre-processing stage the distortions of the camera are corrected, and a small resolution copy is made for helping some processing methods. The next stage is the intra-frame processing, where the software separates the frame to moving and non-moving regions. Up to this point the order of frames

is unimportant because they are handled individually. The object tracking is done in the transition stage, which is partly intra- and partly inter-frame. First the entities are selected from the image, then they can be tracked between frames. Some other processing is done in the inter-frame stage, such as making a traffic intensity map (Fig. 11.). Any module working in this stage requires that the previous frame must be done before the next can be processed. The post-processing stage includes the recording and/or forwarding of the gathered data. With these stages the modules can easily be ordered to form a processing pipeline.

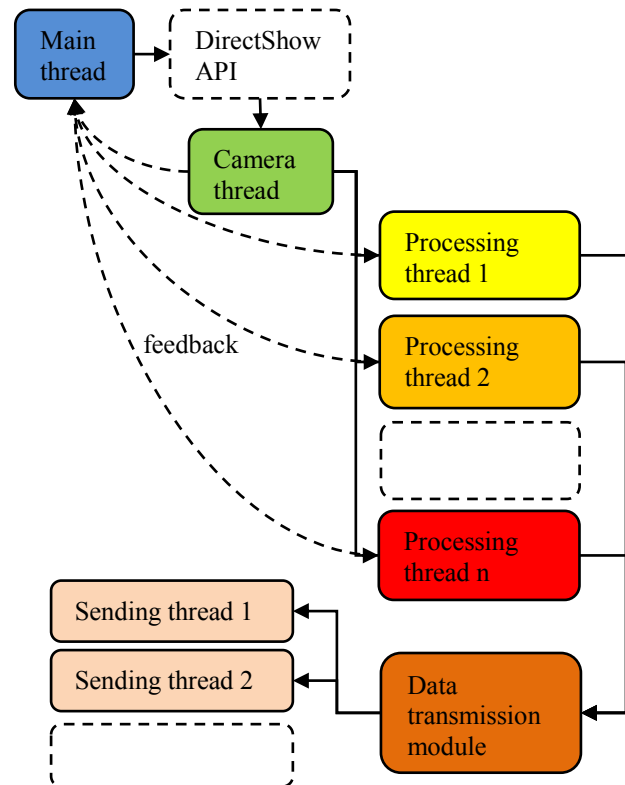


Fig. 2. The threads of the image processing pipeline

The computation on frames is done on multiple threads (Fig. 2.), the number of which is equal to the number of virtual processors. A dispatcher algorithm creates a queue of frames to be processed, and sends a signal to the – otherwise blocking – threads, that to initiate processing of said frames. Then the threads read out a frame from buffer, and start processing it. This procedure uses a common data format in which the frames are passed from module to module. To ensure that inter-frame modules work correctly, the framework checks before every inter-frame processing step (Fig. 3.) if the previous step was carried out by the time (and if not, then the given thread blocks until it gets a signal from another thread that the last frame is done). The display of the output is completely separated from processing with a lockless asynchronous design, which makes a copy of data to be shown. Every module holds information about the state of processing, and can work on multiple threads simultaneously. The internal state of modules is responsible for thread

synchronization. This way the CPU load from thread management is kept low.

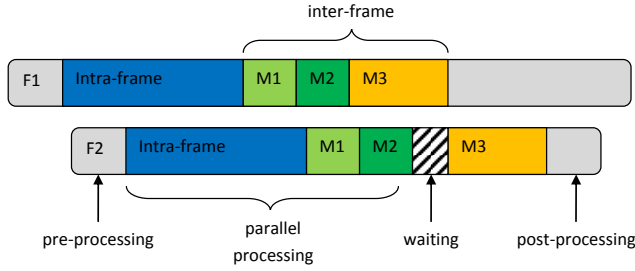


Fig. 3. Multi-threaded image processing

The framework is also responsible for the user interface and data visualization. Since every drawing related to the video is done with DirectX, a rich graphical interface could be created. Every module can draw to the video area a transparent overlay, and can pass intensity arrays which are shown with a collection of built-in shaders. They also have their own page on the header. The framework allows easy debugging, by logging every exception. It also monitors threads, and restarts them if necessary.

III. IMAGE PROCESSING

A. Preprocessing

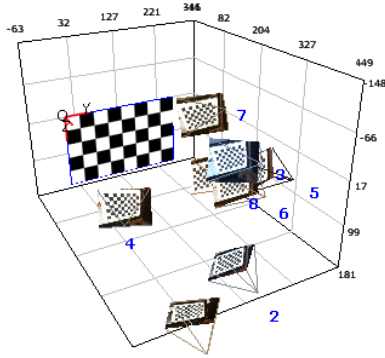


Fig. 4. Camera calibration

To correct the distortion of the cameras the intrinsic parameters are acquired with Zhengyou Zhang's camera calibration method [8]. This uses a known pattern of squares captured from different angles. Because the arrangement and dimensions of squares are known, the cameras' intrinsic and extrinsic matrices can be calculated. The intrinsic matrix is first used by the distortion correction filter. This removes radial and tangential distortions from the images [9]. The program generates a map of two dimension vectors, which contain the distorted position of the given pixel. This map is loaded to the graphics card, where it is used by a HLSL shader. This takes pixels from the distorted input video, and writes them to an output texture undistorted. During this process interpolation is applied, hence the graphics quality of the result image will be high. The small resolution copy of the frame is also made here. The preprocessing step currently

is hardwired to the pipeline, so it cannot be changed within the user interface.

B. Motion Detection

The detection of movement is rather complex, because the algorithm must take the changing background into consideration. The moving parts of the image are given by differences between the current frame and an adaptive background. This background is generated from the consecutive frames and allows adaptation to different light conditions and stopping objects. The process uses the small resolution copy of the frame and three different backgrounds: the main background is used to decide if a given part of the image is changing (moving), the moving background to which the moving regions are compared and a backup background. The main background is set to the first frame in the beginning, while the moving and backup backgrounds start completely black. Each pixel of the incoming frame is compared to the main background, if one's color is found different enough, then it is considered moving. The algorithm assigns counters to every pixel to measure the time which one pixel remains as moving or non-moving. If a pixel doesn't change for a while, then it is merged to the background. This way the slow change of lighting (for example outdoors) can be handled. If a pixel is found to be moving then we compare it to the moving background, so we can see if it is changing continuously. If it's changing then we can safely assume that it is part of a moving object, however if it remains different from the main background for a long time, but it doesn't change, that means it should be merged into the background.

Although the algorithm so far works in theory, it needs some more code for dealing with some side effects. The first of these is flickering: a pixel's color can be on the brink of the interval of background color. The noise of the sensor makes it be sometimes considered moving, sometimes not, and the counters will never fill up. So a different counter, the flickering counter is also assigned to every pixel, which measures the changes back and forth. This way the flickering pixels can be merged. The flickering counters are periodically decreased. The other problem is that while merging steady pixels, the pixels surrounding moving regions will be also merged. That causes semi-transformed pixels to be added to the background, which will significantly reduce the quality of movement detection with false positives. So a map of expanded moving regions is created, and the adjacent pixels are ignored when merging. The last problem to be described is that bigger, slowly moving objects, especially with large solid color surfaces may become victim of merging. The cause is that the movement time counters may fill up, and cause the inner sections of entities to be merged. This is solved by making backup of the merged regions, and restoring the original state if they were inner sections of moving areas.



Fig. 5. Measurements on two pendulums. The overlay shows the time that elapsed after last merging caused by an object staying still. The low resolution overlay can also be compared to the more detailed original frame on this picture.

The movement detection module is among the intra-frame modules, although it is theoretically inter-frame. This is for performance reasons, and allowed by the fact that even if some frames get swapped during parallel processing, it will still work, because all the buffers (backgrounds, counters) rely on an interval of frames, and not the order of them. We need to take into account the fact that as multiple processing threads run simultaneously, it is possible that they will read and write the same memory block in the same time, which could lead to an unstable application. The longer test runs, however, have never produced such an incident even when the computer was on full load; as such, it is likely that simultaneous reading and writing of these arrays doesn't create exceptions, and effects of invalid data are really small.

C. Object Recognition



Fig. 6. The object recognizer is tracking two cars

After moving areas have been identified the program assigns an ID to each of them. The algorithm starts inspecting the image from the upper-left corner, and if it finds a moving pixel, then it searches for neighboring pixels and flood-fills them with the ID of that region, if they are also labeled as moving. During this process size, bounding rectangle and a lot of other properties are also determined. This is the point when intra-frame processing ends, and the real inter-frame processing begins.

The software must recognize the same object on consecutive frames. If there are a small number of nonintersecting objects it could be assumed, that if an

object's position on one frame is close to another's on other frame, then they must be the same object. This works only if there are no intersections. If intersections happen we can do the following:

- 1) Track the combined object, and then if it splits, the original objects can be identified.
- 2) Use pattern-matching based solutions, and try to recognize the overlaid objects.

The algorithm currently employs the first method, because it is much more efficient than the latter. The program calculates the projection of an object's path, and check if there are any foreseeable intersections. If entities are expected to "collide", then they are marked, and their image is captured for later recognition. If the objects touch each other their size changes suddenly, and their position also jumps, so the program can easily recognize collisions, and even the source objects are identifiable. After splitting the program uses two level comparison. First it compares the color histograms of objects, if they are the same, they are assumed to be the same objects. If there is more than one match, then the level two comparison starts. This consists of transforming the images of objects to a scale invariant polar coordinate system. In this system pixels don't form lines, instead, they build up circles. The algorithm starts the comparing on outer circles, and searches for the angle that corresponds to the minimal difference. In this polar system rotation can be done by shifting, so it makes comparison much simpler. As outer circles contain more pixels than inner ones, the resolutions of circles are decreased with the radius of them. If a match is found then the inner circles are also compared beginning with the last circle's rotation. This method can effectively compare images, if the objects are extracted from the frame and only 2D rotation and scaling happens. Pattern based comparison can also use object's trajectories, and start searching at the most likely spot.

The algorithm classifies objects as their age. If an object has been just discovered it is classified as child. Child objects can be much more easily discarded than adults. This is for reducing the image comparison overhead caused by noise, as images of adult objects are kept in memory for a while after their disappearance for later recognition. The minimal size of objects can also be set. At default settings this module does not forward any child or small sized objects. This means that the object's 3D tracking starts 5-10 frames later than the 2D tracking, but the output is clearer from errors.

Although the object recognition module is not meant to use markers, it can filter colors, so it is possible to use markers with it. The output of this module is a list of the tracked entities, and a map showing their positions.

D. Calculation of Direction Vectors

Using the camera's intrinsic matrix every object is assigned to a direction vector in the camera's coordinate system:

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}. \quad (1)$$

Where: x_c, y_c, z_c are the camera space coordinates, A is the intrinsic matrix, and u, v are the coordinates on the image. And the inverse of this (if $z_c=1$):

$$\hat{A}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}. \quad (2)$$

The relative position of cameras to each other is calculated with an equilateral triangle (Fig. 7.):

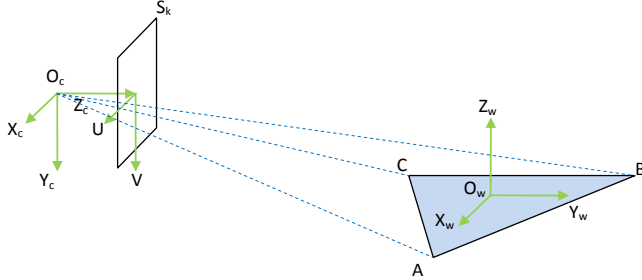


Fig. 7. The determination of the camera's position in world space

The following procedure must be repeated for all cameras. The length of triangle's sides and the vertices' distance from the camera should be given to the program, and the vertices should also be marked on the video. From this data – if the vertices are always specified in the same order – the position of the cameras can be easily determined. Knowing the position of cameras the direction vectors are transformed to world space.

E. Fusion of Camera Data

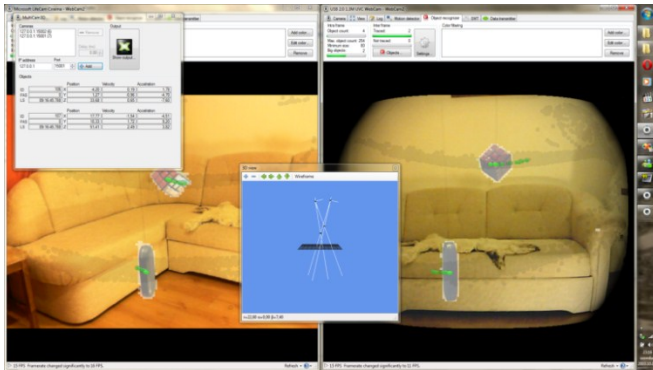


Fig. 8. Two instances of the image processing program and the fusion application

The results from cameras are summarized by a separate program (Fig. 8.), which connects to the cameras through network. The communication channel work on TCP protocol, and allows the cameras' processes to share the gathered information with other programs. The communication module is capable of sending and reconstructing the data that the pipeline processed. Moreover controlling messages can also be sent alongside the video and the list of entities.

When the frames reach the summarizer program they are not synchronous. The algorithm pairs them by using their time code, however this requires that the clocks of the used computers to be synchronized and the delay of the devices also must be known. The first can be solved using NTP protocol. The solution to the latter is still in development.

Although an oscillating light can be used for measuring the delay, the changing movement of objects could also hold enough information to the complete synchronization.

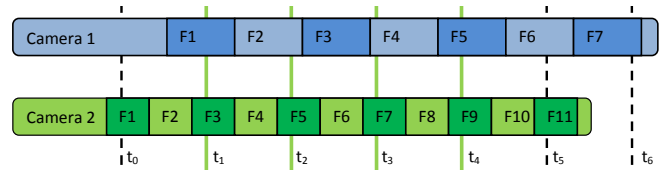


Fig. 9. Determination of the processable time intervals

The data which is captured by cameras and processed by the pipeline is sent to the data fusion application. This data consists of time of capture, list of objects (including their id, position, size, image etc.), and map of objects. As the cameras can connect independently, the summarizer must decide whether enough information is available for processing a time interval (Fig. 9.). Sufficient information consists of the frame data from at least two points of a time interval from at least two cameras. As the capturing is not triggered (consumer webcams were used), the algorithm must interpolate the frames for measurement. Although this has a negative impact on precision, it allows the use of cameras with different frame rates.

The summarizer program pairs the direction vectors received from the cameras. The position of objects is given by the midpoint of distance line segment of the direction vectors starting out from cameras (Fig. 10.). If more than two cameras are used the results are averaged. If no unambiguous solution can be found then the object's image can be also used for coupling (using the methods described in single camera object tracking). If the objects are coupled, the tracking mostly depends on the "intra-camera" processing.

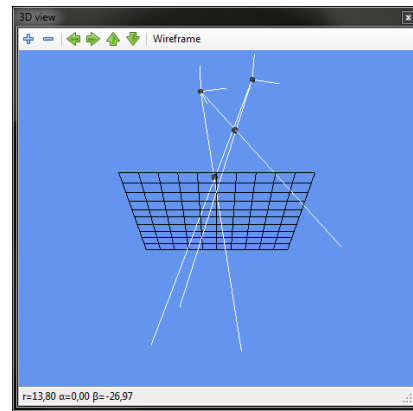


Fig. 10. The three dimensional positions of the two pendulums and the cameras, the grid is anchored to the calibration triangle

The results of measurements can be exported as CSV (comma-separated values) that can easily read by other programs such as Microsoft Excel or LibreOffice Calc (Fig. 11.).

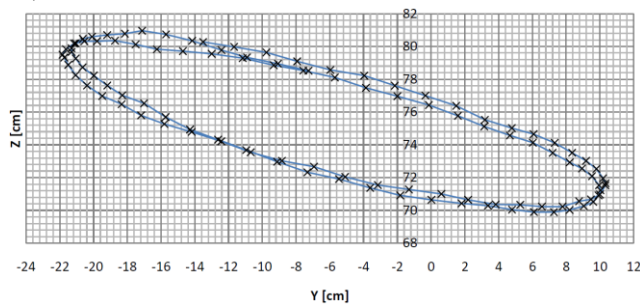


Fig. 11. The trajectories of pendulums as shown in Microsoft Excel

In experiments where two cameras are placed next to each other at 40 centimeters distance and they observe objects 2 meters far, the resolution is around 3 cm in depth, and 0.3 cm in the directions parallel to the captured images (with 720p video source). The precision is highly dependent on the resolution of the used cameras, and their relative position to each other and to the observed objects. A usual desktop computer can process video with this application around 150-400 frame per second depending on the video content and the given hardware configuration.

IV. APPLICATIONS AND CONCLUSION

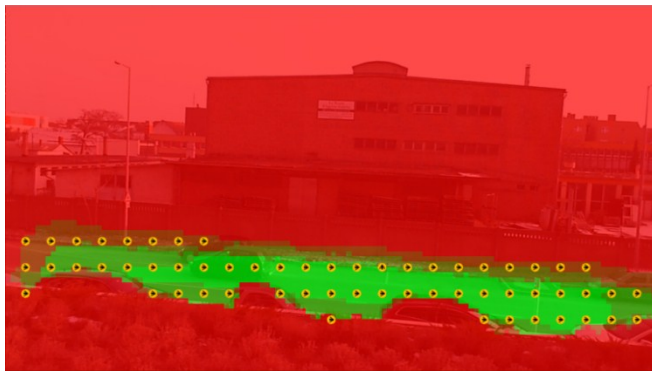


Fig. 12. A plug-in for traffic counting measures the amount of crossings through a given pixel

The framework and the image processing program have a wide range of application possibilities. They can be used for:

- 1) Ballistic experiments with high speed cameras: the program can measure the speed and orbit of projectiles.
- 2) Inspecting the flight of birds, bats and other flying animals. Measuring the speed of running animals.

3) Measuring speed and position in sports, such as golf, javelin throw etc.; measuring the momentary speed of athletes.

4) Traffic monitoring, counting (Fig. 12.)

5) In security as intelligent video surveillance.

The main contribution of this paper is that it describes the development and internal functioning of a low-cost, efficient, hardware accelerated and expandable image processing framework. The description shows the methods of the data processing and the solutions to problems encountered during development. With the finished modules the program is ready to real-world use and can solve simpler measurement tasks. The strongest point of this software is that it can be easily customized and expanded to different needs and it allows the developers to test their ideas quickly, with advanced data visualization and statistics provided by the framework.

REFERENCES

- [1] Barbagli, B. ; Magrini, I. ; Manes, G. ; Manes, A. ; Langer, G. ; Bacchi, M., "A distributed sensor network for real-time acoustic traffic monitoring and early queue detection," 2010 Fourth International Conference on Sensor Technologies and Applications, p. 173, 2010
- [2] Ahmed, S.A. ; Hussain, T.M. ; Saadawi, T.N., "Active and passive infrared sensors for vehicular traffic control" Vehicular Technology Conference, 1994 IEEE 44th, Stockholm, p. 1393
- [3] Gajda, J. ; Sroka, R. ; Stencel, M. ; Wajda, A. ; Zeglen, T., "Data fusion applications in the traffic parameters measurement",
- [4] Fei Zhu, "A video-based traffic congestion monitoring system using adaptive background subtraction," 2009 Second International Symposium on Electronic Commerce and Security, p. 73-77.
- [5] Fei Zhu, Liangyou Li, "An Optimized Video-based Traffic Congestion Monitoring System," 2010 Third International Conference on Knowledge Discovery and Data Mining, p. 150-153.
- [6] Kantip Kiratiranapruk; Supakorn Siddhichai, "Practical application for vision-based traffic monitoring system," 2009. ECTI-CON 2009. 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology.
- [7] Pipatphon Laparmonpinyo; Orachat Chitsobhuk, "A video-based traffic monitoring system based on the novel gradient-edge and detection window techniques," 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), p. 150-153.
- [8] Gergely Vassy; Tamás Perlaki: "Applying and removing lens distortion in post production", The Second Hungarian Conference on Computer Graphics and Geometry 2003, Budapest, 2nd best speech.
- [9] Z. Zhang, "A flexible new technique for camera calibration", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.22, No.11, pages 1330-1334, 2000
- [10] Szirmay-Kalos L., Antal G., Csonka F.: "Háromdimenziós grafika, animáció és játékfejlesztés" ComuterBooks Budapest, 2003
- [11] Riemer Grotians: XNA 3.0 Game Programming Recipes, APRESS, 2009