Transactions of the VŠB – Technical University of Ostrava, Mechanical Series
No. 1, 2009, vol. LV
article No. 1661

Tomáš KOT*, Vladimír MOSTÝN, Petr NOVÁK*****

INTEGRATION OF SENSOR INFORMATIONS INTO CAMERA IMAGE PRESENTED TO MOBILE ROBOT OPERATOR

INTEGRACE INFORMACÍ SENZORŮ DO OBRAZU KAMERY ZOBRAZOVANÉHO OPERÁTOROVI MOBILNÍHO ROBOTU

Abstract

The paper covers a possible way of displaying the image of mobile robot camera subsystem on a personal computer independently on the used hardware solution of camera connection; together with integration of additional text and 2D graphical information about the state of the robot and data from the sensory subsystem. Also utilization of augmented reality is described, for composition of the camera image and 3D graphical objects generated based on laser scanner data. The created application has been programmed in Visual C++; graphics is rendered using Microsoft Direct3D.

Abstrakt

Článek popisuje možný způsob zobrazování obrazu kamerového subsystému mobilního robotu na osobním počítači nezávislý na použitém hardwarovém řešení připojení kamery, včetně integrace dodatečných textových a 2D grafických informací o stavu robotu a údajích senzorického subsystému. Rovněž je popsáno využití tzv. rozšířené reality pro kompozici obrazu kamery a 3D grafických objektů generovaných na základě údajů laserového scanneru. Vytvořená aplikace je naprogramována ve Visual C++, grafika je renderována s využitím Microsoft Direct3D.

1 INTRODUCTION

In spite of progress in autonomous mobile robotics, mobile robots controlled remotely by a human operator are still widely used. The operator navigates the robot in almost all cases primarily using feedback in the form of images from the camera subsystem, consisting of one ore more fixed or pivoted cameras. Because of the limitations of cameras, robots are usually equipped also with some sensors, which help the operator to avoid obstacles and to better understand the current state of the robot.

A single software application running on a personal computer (a notebook) can serve at the same time as a control application (processing inputs from the operator and sending them to the robot) and as a system displaying images from cameras and data from sensors. When using only a single monitor, it is better to integrate the sensor information directly into the camera image. This way the camera image can occupy the whole screen and moreover – the operator can see all the information without looking away from the image.

Even augmented reality can be applied to render some virtual object directly into the real 3D world, for example to highlight obstacles detected by distance sensors.

* Ing., Department of Robotics, Faculty of Mechanical Engineering, VŠB - Technical University of Ostrava, 17. listopadu 15, Ostrava, tel. (+420) 597 329 363, e-mail tomas.kot@vsb.cz

** prof. Dr. Ing., Department of Robotics, Faculty of Mechanical Engineering, VŠB - Technical University of Ostrava, 17. listopadu 15, Ostrava, tel. (+420) 597 324 257, e-mail vladimir.mostyn@vsb.cz

*** doc. Dr. Ing., Department of Robotics, Faculty of Mechanical Engineering, VŠB - Technical University of Ostrava, 17. listopadu 15, Ostrava, tel. (+420) 597 323 595, e-mail petr.novak@vsb.cz

2 VIDEO CAPTURE

Video signal from the camera subsystem can be transmitted in analog or digitalized form. If using the analog transmission method, additional hardware must be provided to digitalize the signal for the computer. It can be a variety of different devices; for example a video capture PCI card, an external USB tuner with a composite input etc. The video capturing software should be preferably able to capture video from any active device; this can be fulfilled almost perfectly using the *DirectShow* media-streaming architecture [1, 2]. However, as *DirectShow* is a very complex and complicated application programming interface, it is favorable to choose an already made library based on *DirectShow*, for example the library called *VideoInput*.

VideoInput [3] is a very fast and free windows video capture library, distributed as a source code, which can be added directly to the final application and compiled with it; thus no additional software nor dynamic-link library files are needed. It also supports almost every standard video input device, including USB webcams.

2.1 Video images rendering

The *VideoInput* library provides a function that fills a supplied memory buffer with raw image data of the actual camera frame. Format of the data is 24-bit *BGR* (1 byte for blue, 1 byte for green and 1 byte for red color) for every pixel. As *Direct3D* API has been chosen for graphics rendering, and *Direct3D* prefers 32-bit image formats, it was necessary to modify the source code of the *VideoInput* library slightly to return 32-bit *BGRX* format instead. The additional 8 bits are reserved for *alpha channel*, which is not needed in this case, thus it is marked with the *X* letter instead of *A* and filled with 0 for every pixel. Fig. 1 shows the structure of the image data; individual channels (*R*, *G*, *B*, *X*) can contain values from 0 to 255 (1 byte), and the total number of pixels equals to the width of the camera image (*W*) multiplied by the height (*H*).

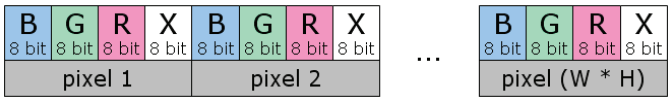

Fig. 1 Video frame data format

The memory buffer filled with these data can be directly copied to a Direct3D *texture resource object*, if the texture has been locked for writing beforehand. Locking is necessary to synchronize the graphic card with CPU to be able to update textures stored in video memory.

The updated texture is then mapped to a *full-screen quad* (a rectangle occupying the whole client area of the application window) and rendered using *pixel shader* effects described later.

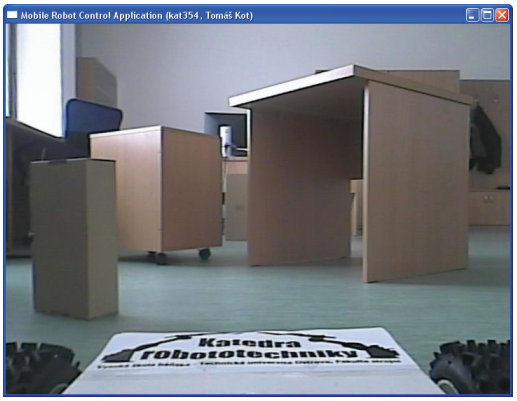

Fig. 2 Window of the Direct3D application with rendered camera image from a mobile robot

2.2 Integration of additional camera images

Mobile robots can have more cameras, for example one pivoted primary camera and one fixed camera located at the back of the robot and serving as a reverse camera. If all the cameras are always equally important, it is better to render them all individually to separate windows, preferably also on separate monitors. When considering the reverse camera, however, the image from this camera is important only when the robot is moving backwards.

The proposed solution combines images from both cameras; the reverse camera image is displayed semi-transparently in a small window at the top-left corner of the main camera window (Fig. 3, left). When the robot starts moving back, the window becomes more opaque and its size increases (Fig. 3, right).

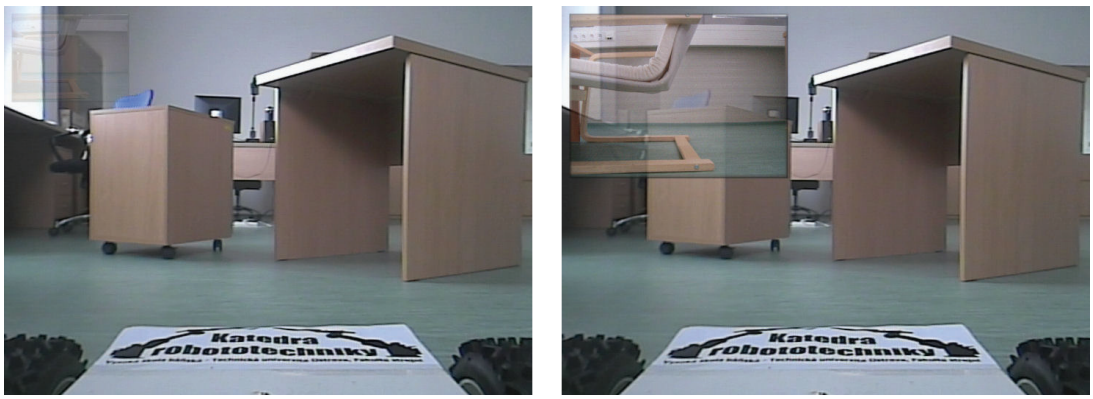

Fig. 3 Reverse camera window – robot moving forwards (left figure) and backwards (right figure)

3 ROBOT STATUS DISPLAY

A simple 2D or more sophisticated 3D graphics can be used to display robot status, depending on the nature of the particular information.

State of the mobile robot includes for instance information about speed, battery condition, communication status, detected input device commands, tilt angles, heading direction etc. These kinds of information can be displayed textually, illustratively by simple graphics, or by combination of both. Preferred location for the graphics (for the mobile robot shown on Fig. 3) is at the bottom portion of the image, because the area is occupied by a part of the robot anyway.

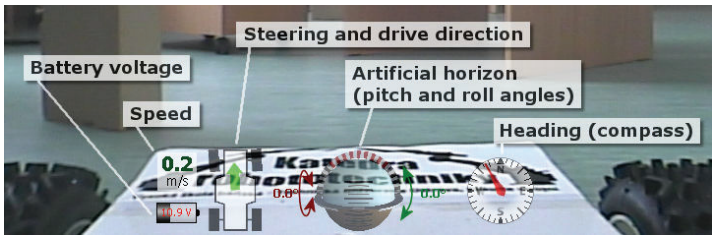

Fig. 4 Robot status display with few basic data

The current version of the application displays (Fig. 4) speed of the robot in [m/s], battery voltage together with a simple illustration of battery condition, robot movement overview (steering angle and drive direction) in a graphical form, a 2D compass based on a solid state compass sensor and a functional 3D model representing artificial horizon supplemented with numeric values of pitch and roll angles based on an accelerometer sensor. When the robot tilts too much (roll angle), a warning about instability is shown.

4 APPLICATION OF AUGMENTED REALITY

A mobile robot can be equipped with a variety of distance sensors, ranging from simple proximity IR sensors to laser scanners. Obstacles detected in the view angle of the camera can be highlighted directly in the camera image, using augmented reality. This representation is very clear, as the operator sees immediately which object in the camera image – and thus in the real world – corresponds to the object detected by the sensor at the specific distance.

4.1 Camera calibration

Direct3D uses a virtual camera described by transformation matrices similarly as the real camera uses lenses. To be able to add some virtual objects into the real world camera image, it is necessary to match the virtual camera with the real one. This includes *extrinsic parameters* describing the camera position and rotation in the world, and *intrinsic parameters* describing the optical properties of the camera. Extrinsic parameters can be ignored now, because we assume the cameras to be in the origin of the world coordinate system; but intrinsic parameters of the virtual camera must be matched to those of the real one.

Projection of a real camera can be described by the following equations:

$$x' = \frac{x}{z}, \tag{1}$$

$$y' = \frac{y}{z}, \tag{2}$$

$$u = f_x \cdot x' + c_x, \tag{3}$$

$$v = f_y \cdot y' + c_y. \tag{4}$$

where x, y, z are coordinates in the 3D space; u and v are 2D coordinates of point projection in pixels; c_x and c_y represents a principal point (ideally at the image center) and f_x, f_y are focal lengths expressed in pixel-related units.

As real camera lenses usually have some *distortion* – primarily *radial*, sometimes also slight *tangential* (omitted in this case) – the equations must be extended:

$$x'' = x'\left(1 + k_1 r^2 + k_2 r^4\right), \tag{5}$$

$$y'' = y'\left(1 + k_1 r^2 + k_2 r^4\right), \tag{6}$$

$$r = \sqrt{x'^2 + y'^2}, \tag{7}$$

$$u = f_x \cdot x'' + c_x, \tag{8}$$

$$v = f_y \cdot y'' + c_y, \tag{9}$$

where k_1 and k_2 are the radial distortion coefficients.

The camera model used by *Direct3D* to render 3D objects onto a 2D screen uses the principle of an *ideal* pinhole camera, described by a transformation matrix called *projection matrix*. This matrix can be created easily using the *D3DXMatrixPerspectiveFovLH* function, which returns:

$$P = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & \dfrac{z_f}{z_f - z_n} & 1 \\ 0 & 0 & \dfrac{-z_n \cdot z_f}{z_f - z_n} & 0 \end{bmatrix}, \tag{10}$$

where z_f and z_n are far and near clipping planes (these parameters are specific only to 3D computer graphics and don't have an equivalent for the real camera), s_x and s_y are scales dependent on the defined field of view (FOV_y) of the desired camera model and pixel aspect ratio (a_r) of the image:

$$s_y = ctg\left(\frac{FOV_y}{2}\right), \tag{11}$$

$$s_x = \frac{s_y}{a_r}. \tag{12}$$

Technically, the virtual camera can be matched perfectly only to an ideal real camera without radial distortion and with $c_x = 0$; $c_y = 0$, using equations (3), (4). In a general case, however, this is not possible, and the real camera must be calibrated to match the ideal pinhole model.

Calibration of the camera used in this project was done with the help of the image processing and computer vision library called *OpenCV* [4]. This library provides methods to find the intrinsic parameters together with distortion coefficients, by taking few images of a black-and-white chessboard pattern. It is also possible to use an *OpenCV* function to remove the distortion of the image, but it proved to be incomparably faster to use a *Direct3D pixel shader* for this.

This specially programmed pixel shader is executed automatically for every pixel of the camera image during its rendering using the full-screen quad and runs directly in the graphic card GPU, thus massively parallelly. While the *OpenCV* distortion-removing function takes 0.026 s on the testing computer (Pentium D940, GeForce 8600GT) for every camera frame, the pixel shader needs only around 0.0001 s. The code of the pixel shader corresponds to equations (5), (6) and (7); the distortion coefficients were taken from the calibration mentioned above, executed only once for the particular camera.

4.2 Integration of laser scanner distance data

A laser scanner provides a set of distance measurements in a plane. The sensor used for this project, HOKUYO URG-04LX, has 240° angle of view and 4 m range [5]. Laser sensors provide only 2D information about the detected obstacles, in the form of polar coordinates; but in spite of that, the obstacles can be still represented by simplified 3D surfaces and rendered directly to the camera image.

Both the virtual world and the real world base coordinate systems were chosen to be at the same point where the cameras (real and virtual) are, see Fig. 5. The scanner is located on the negative z_b axis, thus the measured points (after conversion from polar to Cartesian coordinates) are already in the world coordinate system, lying on a plane with the following equation:

$$z_b + (h_c - h_s) = 0. \tag{13}$$

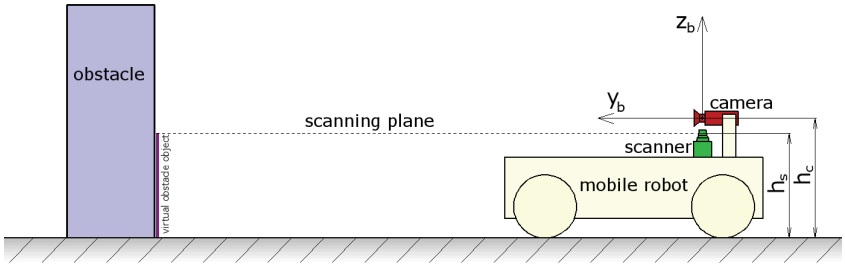

Fig. 5 Side view of the mobile robot

To render the scanned points as surfaces, every two adjoined points are connected with a line and the lines are extruded along the vertical axis to form rectangular surfaces. The top edge of these rectangles has $z_b = -(h_c - h_s)$ and the bottom $z_b = -h_c$ (Fig. 5). When the rectangular surfaces are then rendered over the camera image, they follow the curve where the real obstacles touch the ground. This increases the visual impression of augmented reality, as the virtual objects appear to belong to the real world (Fig. 6).

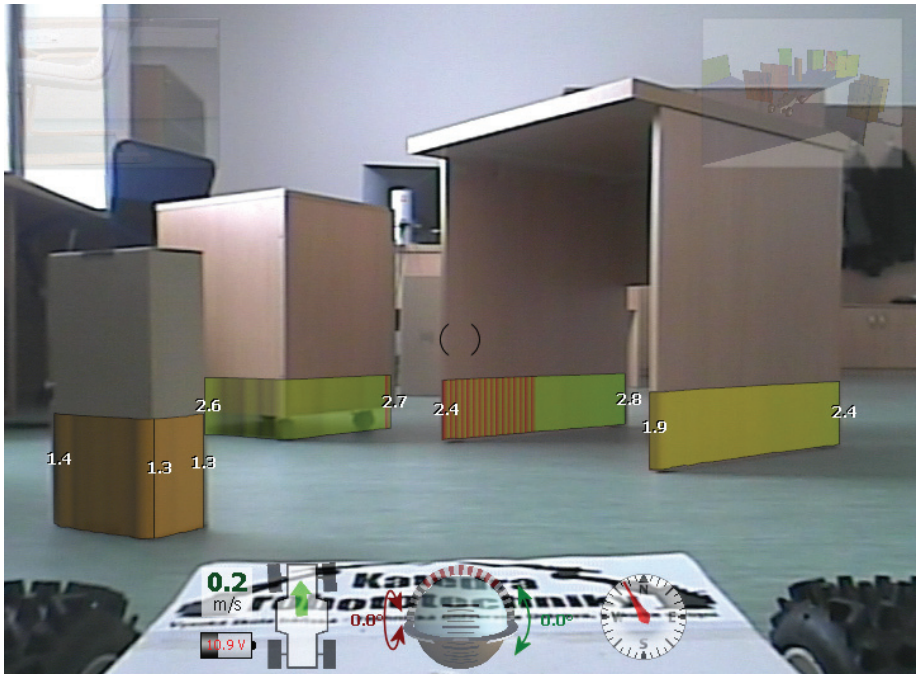

Fig. 6 Semi-transparent virtual objects representing laser scanner distance data

Color of the virtual objects depends on the corresponding distance from the scanner and is coded in gradients: red (10 cm) – yellow (2 m) – green (4 m). Portions of obstacles that are directly in front of the robot (the robot would crash into them if moving still forward) are marked with red horizontal stripes, regardless their distance. Corners of obstacles are traced with black lines, with a numeric value of the distance to the scanner written on each of them (in meters).

As the camera has a very small angle of view compared to the scanner (50° versus 240°), only a very small part of the detected obstacles is visible directly in the camera image. Therefore an additional window has been added to the right corner, displaying a fully interactive 3D reconstruction of the detected obstacles, including a simplified model of the mobile robot at its proper position and orientation. This window can be enlarged similarly to the reverse camera window.

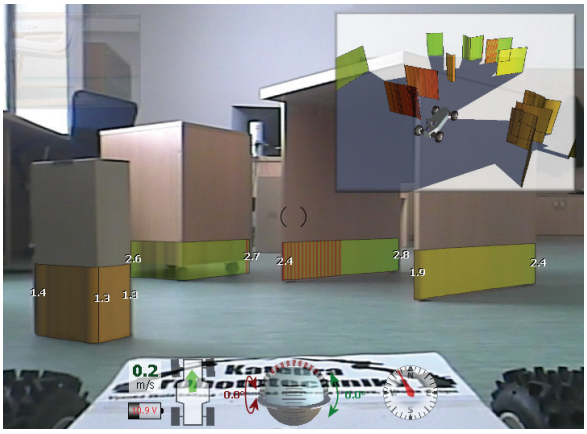

Fig. 7 Enlarged window with 3D representations of all the obstacles in the scanner field of view

If the physical camera can be rotated, the virtual camera rotates equivalently. The scanner has larger field of view, so it can be stationary and even so the obstacles will be correctly highlighted also when the operator turns the camera sideways.

4.3 Integration of IR proximity sensors data

IR proximity sensors measure distance on an obstacle only in one direction. There can be more of them mounted on a mobile robot, and if some of them are looking to the front, the measured distances can be also rendered directly into the camera image.

Fig. 8 shows a mobile robot with 3 IR sensors on the front bumper: one on the left side, second in the middle and third on the right side. Detected obstacles are highlighted with a circular mark rendered exactly at the position where the infrared ray hits the surface of the obstacle. The distance is again coded with color and also a numerical value in meters is added.

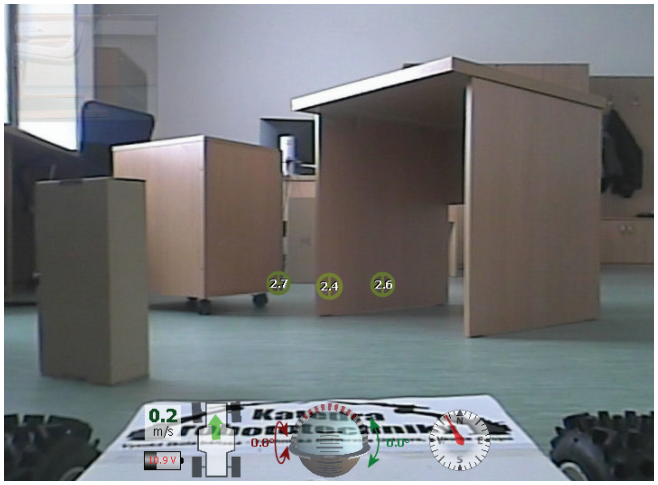

Fig. 8 Distances measured by three IR proximity sensors

5 CONCLUSION

Direct3D proved to be perfect for the desired tasks. It is possible to draw the individual information in almost any way and to freely combine them, even with alpha-blending (transparency). Pixel shaders, usually used in computer games to apply various effects, represent a very powerful and relatively easy way of performing image processing tasks, in this case the camera calibration. Direct3D

graphics is fully hardware accelerated on almost every graphic card (including integrated notebook chips), so the application runs fast and doesn't require a hi-end computer.

The current version of the application works only as a display system, but it will be extended to serve also as a control application. The goal is to make a self-reliant and universal software allowing a full operation of an operator-controlled mobile robot, while being as user-friendly and efficient for the human operator as possible.

The application can be also easily modified to support stereo vision. Both stereo vision cameras can have their virtual equivalent and this way the augmented reality will be consistent even in stereo glasses.

This article was compiled as part of project FT-TA5/071, supported by the Industrial Research and Development Program of the Ministry of Industry and Trade.

REFERENCES

[1] MICROSOFT. *DirectShow* [online]. 2009. Available from WWW: <http://msdn.microsoft.com/en-us/library/ms783323.aspx>.

[2] *Direct Show* [online]. 2009. Available from WWW: <http://en.wikipedia.org/wiki/Directshow>.

[3] WATSON, Theodore. *VideoInput Library* [online]. 2009. Available from WWW: <http://muonics.net/school/spring05/videoInput/>.

[4] *OpenCV* [online]. 2009. Available from WWW: <http://opencv.willowgarage.com/wiki/>.

[5] HOKUYO. *Scanning Range Finder URG-04LX* [online]. 2008. Available from WWW: <http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html>.

[6] MICROSOFT. *Direct3D 9 Graphics* [online]. 2009. Available from WWW: <http://msdn.microsoft.com/cs-cz/library/bb219837(en-us,VS.85).aspx>.

[7] TVARŮŽKA, A. *Senzorický subsystem robotu : dissertation thesis*. Ostrava : VŠB - TU Ostrava, 2008. 139 s.