Chapter 2
Bio-Inspired, Real-Time Passive Vision for Mobile Robots

Piotr Skrzypczyński, Marta Rostkowska, and Marek Wąsik

Acronyms

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FPS	Frames Per Second
GPGPU	General Purpose Graphics Processing Unit
HSV	Hue-Saturation-Value (color model)
MIPI	Mobile Industry Processor Interface
ORB	Oriented FAST and Rotated BRIEF
QR-code	Quick Response matrix code
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
TFLOPS	Tera Floating Point Operations Per Second
USB	Universal Serial Bus
VFH	Vector Field Histogram

2.1 Introduction

We are witnessing how robots proliferate to everyday life, and the number of commercially available mobile robots increases gradually. Mobile robots perform tasks like surveillance, cleaning or they assist handicapped people. However, to have

P. Skrzypczyński (✉) · M. Rostkowska · M. Wąsik
Institute of Control, Robotics, and Information Engineering, Poznań University of Technology, Poznań, Poland
e-mail: piotr.skrzypczynski@put.poznan.pl; marta.a.rostkowska@doctorate.put.poznan.pl; marek.s.wasik@doctorate.put.poznan.pl

O. Sergiyenko et al. (eds.), *Machine Vision and Navigation*,
https://doi.org/10.1007/978-3-030-22587-2_2

a satisfying level of autonomy these robots need to reliably perceive objects and events in their environment. At the same time robots have to be affordable and easy to maintain. Hence, the design of sensors adequate for the navigation-related tasks becomes important.

Nowadays cameras are considered the most compact and affordable exteroceptive sensors in robotics. Passive vision captures a large amount of data, reflecting both the photometric and geometric properties of the observed scene, but requires considerable computing power, and has a number of limitations related to the used sensors. A monocular camera has a limited field of view, and gives only an angle to the observed feature/landmark, but no range information. Cameras on a stereo rig can measure depth in unknown scenes, but their field of view is also limited.

However, natural evolution has developed visual perception systems that perfectly fit to the needs and environment niches of particular species of animals. Some of them are incredible, like the visual sense of flying insects [37]. These insects have a wide-field view and complex eyes, which allow them to navigate efficiently. Similarly, some mobile robots use omnidirectional cameras, which perceive whole surroundings from a single view [33]. Such cameras ensure that the robot gathers necessary knowledge about the environment in reasonable time. Regrettably, it is not easy to calculate robot's or objects' position employing only data from omnidirectional camera. Visual perception that has developed in more complex animals consists of peripheral and foveal vision. The brain of an animal can provide a correct interpretation of the environment employing cues from both systems. In general, it is possible, because peripheral vision cues are pursued by the eye fixation. However, accurate perception of distances requires foveal analysis, involving central vision. Eventually, two or more views of the scene are required to produce 3D location of unknown objects, which in animals is possible owing to binocular vision.

Following the most efficient biological vision examples, we decided to combine omnidirectional and peripheral/foveal vision mechanisms in our construction. In this way, we delivered a system which combines advantages of both camera types: 360° field of view and accurate environment's data (robot's and objects' position). We created a vision sensor having a hybrid field of view through combination of a camera looking upward into a curved mirror, and a typical perspective camera mounted on top of this mirror (Fig. 2.1). This sensor was presented for the first time in [21], while the obstacle detection algorithm was developed separately, using only the omnidirectional camera [36]. In this chapter we present in a unified way the peripheral vision part, and the algorithms for distance measurement and obstacle detection, that are related to foveal vision. Moreover, the new version of the sensor is presented, which has the perspective camera mounted on a servo. Owing to this design, the perspective camera can be rotated horizontally, which allowed us to create new functions of the sensor. Thus, we describe object tracking that in turn makes it possible to actively select the field of view for the perspective camera, resembling the natural eye fixation mechanism. In our sensor real-time image processing is ensured by a Nvidia Jetson embedded computer. The first prototype was based on

Fig. 2.1 Hybrid field of view passive camera with an actuated central vision sensor

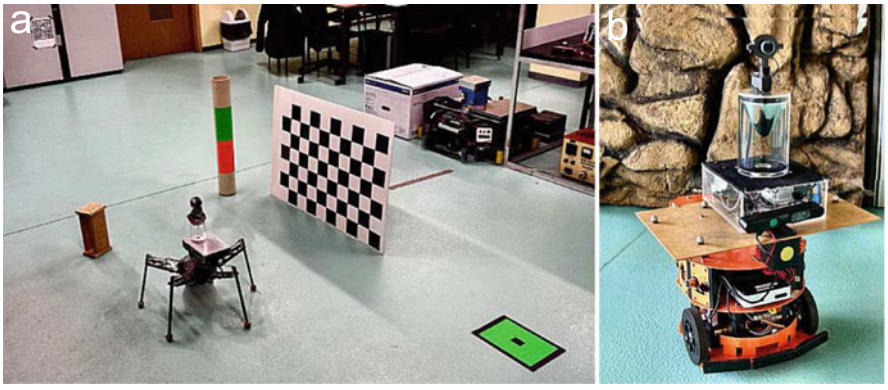

Fig. 2.2 Mobile robots with the self-contained vision sensor: the first prototype on a small walking robot (**a**), and the improved version on a wheeled robot (**b**)

the TK-1 board [12, 21], while the improved version uses the more recent Jetson TX-1. Nvidia Jetson computers are energy-efficient, compact, and powerful, making our sensor a self-contained perception unit. Such a design suits small, low-cost, and resource-constrained mobile robots (Fig. 2.2).

2.2 Related Work

Literature brings different examples of visual perception systems inspired by Nature. The system presented in [30] creates a global description of the scene, and calculates a coarse localization of the mobile robot. Then the results are improved based on the extraction of local features from images. This approach takes inspirations from the peripheral/foveal vision scheme, but still relies on a single camera for perception, and does not control the field of view. A different foveation implementation is presented in [34]. This work introduces an active vision system for an anthropomorphic robot with a pair of cameras located in each of its eyes. This design ensures a wide field of view, and the observed objects are always maintained in the foveal vision area. Santos-Victor et al. [24] describe a robot navigation system based on a wide field of view and simple processing of the optical information, which was directly inspired by insect vision.

Omnidirectional cameras are popular in various applications for two decades [38]. They are used for relatively simple robot navigation [39], and more recently, are combined into multi-camera systems, for example to obtain omnidirectional stereo images [23]. The concept of combining an omnidirectional and perspective camera is widely used in the soccer robots. However, in the soccer domain well-defined visual markers are available in the playing field [15], which are not present in other application areas.

Also the problem of stereo-based distance measurements using a hybrid system of cameras was analyzed in the context of mid-size league robotic soccer [11]. It was demonstrated that stereo-based computation of objects' positions often results in highly uncertain measurements if the cameras are poorly calibrated. Hence, a simple object localization scheme was proposed in [11], combining the bearing information from the catadioptric, and the distance to a known object (e.g. a ball used in soccer) from the perspective camera. In [20], we proposed a similar system, which solves the real-time localization task on a small mobile robot. Our system uses QR-code landmarks located on objects to simplify the localization task. Those examples demonstrate that cooperation between cameras of totally different field of view may be beneficial to various robotic tasks.

In the robotic literature only a few works tackle the problem of integrating the omnidirectional and perspective cameras in a more tight and direct manner than it is done in typical soccer robots. Cagnoni et al. [6] present a hybrid omnidirectional pin-hole sensor, but they focus only on the sensor description and calibration procedure. A system which supports obstacle detection for mobile robots is shown in [1]. Our hybrid solution presented in this chapter is conceptually similar to the designs discussed in [6] and [13], but in contrary to our system, the solutions from [6] and [13] require external processing of the images. Such an approach renders real-time processing of the omnidirectional images almost impossible. Hence, these sensors hardly can be applied in mobile robot navigation, which requires real-time response to various visual stimuli. We have applied the first prototype of our sensor on a small legged robot, which does not have enough on-board computing power to build

an environment representation in real time [36]. Although omnidirectional cameras have been already used on few walking robots [19], our application demonstrates gains due to the use of a vision sensor with on-board processing. Although the first version of our peripheral vision software has been already described in our recent conference paper [22], this chapter not only provides a more detailed description, but introduces also an object tracking module, based on particle filtering. This module is a good example of the on-board processing power in our sensor.

2.3 Hardware of the Sensor

Our proposal of the new passive vision sensor consists of three subsystems. The first subsystem is a single-board computer hosting other components and providing the on-board computation resources. The second part is an omnidirectional subsystem consisting of an upward-looking camera with a properly curved mirror. The camera and its mirror are combined by a transparent tube. The last subsystem is a standard perspective camera with the USB interface that is rotated by a servo. Two prototypes of the sensor have been constructed, which share the general design, but differ in the components being used. In both sensors a hyperbolic mirror machined from aluminum alloy and then polished manually is used. The shape of the mirrors is chosen in a way that ensures the single effective viewpoint imaging geometry [2]. With this geometry every pixel in the acquired image receives the light passing through the common point in one particular direction, which is required to produce geometrically correct images.

The omnidirectional subsystems in both variants of the sensor use cameras dedicated to the Jetson single-board computer, and equipped with the CSI-2 MIPI interface. In the first prototype it was E-Cam130 CUTK1 manufactured by E-Con, which yields 1920×1080 images at the frame rate of 30 FPS. The second prototype has the Leopard Imaging LI-IMX274-MIPI-CS camera, with the resolution of 3864×2196 pixels. In both cameras the resolution and frame rate can be changed by software, but we have chosen the parameters that best suit the application, and are a trade-off between the high resolution, high frame rate, and compatibility with the available software.

The most different component in the first and the second prototype is the perspective camera subsystem. In the older version, a simple webcam is fixed to the top of the mirror with a printed plastic part. In the upgraded design, the camera is attached to the small MX12-W servo from Robotis [4]. Thus, the perspective camera can rotate horizontally. The flexible USB cable makes it possible to cover full 360° field of view, but the camera cannot rotate $n \times 360°$. The first version of the design uses a Logitech 500 webcam with the resolution 1280×1024 pixels, while the improved one is equipped with a more compact Microsoft Lifecam, having the resolution of 1280×720 pixels.

The most important aspect of our sensor design is the use of a modern single-board computer as the base. The first prototype uses Nvidia Jetson TK-1 with the

Tegra K1, for which the main computing power is provided by Kepler architecture graphics cores with compute unified device architecture (CUDA) support. Jetson TK-1 achieves 300 GFLOP/s in the single-precision mode. The second version is based on the more recent Jetson TX-1, which has the GPGPU with 256 cores of the improved Maxwell architecture. The improvements allow the TX-1 version to produce the computing power of about one TFLOP/s. However, a drawback of the standard TX-1 model is its increased footprint, which required to increase the size of the whole sensor unit. Both models of the Jetson board facilitate the CSI-2 MIPI interface for cameras allowing direct data transfer between the camera and the GPGPU. This interface is used, however, only for the omnidirectional cameras. The perspective camera is connected via typical USB interface.

2.4 Basic Software and Calibration

The developed sensor requires proper calibration of its components, as well as parameters calibration for the stereo pair that consists of the perspective camera and the virtual camera created by software from the omnidirectional image. All extrinsic parameters (rotations and translations) are estimated with respect to the coordinate frame of the catadioptric camera, which is considered the reference frame of the whole sensor.

Calibration for a hybrid system is much more complicated than in a standard stereo vision systems, because of the geometric distortions in the omnidirectional images. Before extracting metric information from images acquired by the omnidirectional part, the geometric distortions must be removed. Calibration of a stereo pair, where one of the cameras is perspective, and the other one is omnidirectional, is described in [6]. In this method, both cameras observe calibration patterns on several parallel surfaces of known relative positions. This assumption is a drawback of the method, because it makes implementation complicated and vulnerable to errors due to inaccurate location of the calibration surfaces. Hence, we decided to use a simpler method based on the existing open-source calibration tools, which are well-documented and commonly used in vision research. In our method, the omnidirectional camera and the perspective camera are at first calibrated independently, and a virtual camera view is defined from the omnidirectional image. Then, calibration patterns are shown to both cameras (perspective and virtual) in different positions and angles. When the required number of images is collected, these two cameras are calibrated as a standard stereo pair.

The calibration procedures for both subsystems and the stereo pair are highly automated, and do not involve any external equipment other than a simple 63-field chessboard pattern. We have used the same pattern in all the calibration procedures, and the same size of the calibration database (20 images).

2.4.1 Calibration of the Subsystems

Calibration of the perspective camera can be accomplished by using the popular Matlab toolbox [5] or the OpenCV library procedures [16]. In contrast, the known calibration methods for omnidirectional vision systems are often particular to the camera type [3], or require an accurate specification of the mirror geometry and additional equipment to perform the calibration procedure [10]. Among the omnidirectional camera calibration methods known from literature the one proposed by Scaramuzza et al. [26] appears to be the most universal and practical one, as it uses only a standard chessboard pattern, and does not assume any particular mirror or camera type. This method is implemented in the OCamCalib toolbox, which we have used for the presented sensor.

The calibration process starts with determining the model of the perspective camera. The model of distortions with five coefficients, which describe the radial and tangential distortions of the image is used. The perspective camera matrix is calculated from formula:

$$\mathbf{K}_p = \begin{bmatrix} f_{c_1} & \alpha_c f_{c_1} & c_{c_1} \\ 0 & f_{c_2} & c_{c_2} \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.1}$$

where f_{c_1} is the horizontal and f_{c_2} is the vertical focal length, c_{c_1} and c_{c_2} define the center of the image, and α_c is the pixel skew coefficient.

As already mentioned, the omnidirectional camera is calibrated employing the approach and camera model proposed by Scaramuzza [25]. The geometric model shown in Fig. 2.3 is represented by the formula:

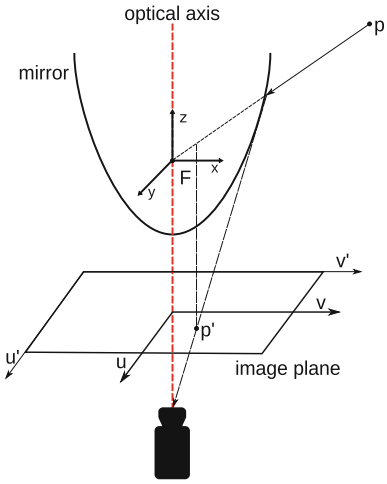

Fig. 2.3 Geometric model of the catadioptric camera

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(u,v) \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(\rho) \end{bmatrix}, \tag{2.2}$$

where u and v represent projection of a 3D point $\mathbf{p}$ into the perfect (i.e. undistorted) image, x, y, and z are image coordinates of this point on the mirror surface, while ρ is the distance between the projected point $\mathbf{p}'$ and the undistorted image's center.

To solve the Scaramuzza equations and define the camera model, it is necessary to calculate $z = f(\rho)$, defined as a fourth order polynomial $z = a_0 + a_1\rho + a_2\rho^2 + a_3\rho^3 + a_4\rho^4$. In order to receive an optimal solution, the coefficients are computed iteratively and the reprojection errors are observed. We use the experimental procedure described in [26]. When preparing the calibration data it is of great importance to cover the whole field of view of the camera by the chessboard patterns, because the calibration data are used also to compensate any misalignments existing between the mirror and the center of the camera. The calibration process has two stages. First, the center of the omnidirectional image $\mathbf{o}_c = [u_c, v_c]^T$ and the affine matrix $\mathbf{A}_{(2\times2)}$ are calculated. The affine matrix $\mathbf{A}_{(2\times2)}$ determines the relation between (u, v) coordinates of the idealized image and the actual image coordinates (u', v'). Eventually, the calibration results are refined applying the iterative Levenberg-Marquardt non-linear optimization technique.

2.4.2 Panoramic Images

Typically, a view of the environment, which is seen in a picture from the catadioptric camera is highly distorted. While objects can be detected and roughly localized or tracked using raw omnidirectional images, it is not possible to calculate accurate positions of these objects or point features. For the calculation of the accurate distances and geometric relations a rectified (i.e. geometrically corrected) 360° panoramic image is necessary. With such images, the sensor can not only detect obstacles, but also measure accurate distances to objects in the wide field of view.

Scaramuzza [25] presents a simple method of the image rectification based on geometric inverse projection and the calibrated model of the catadioptric camera. Based on geometry and dimensions of the images (Fig. 2.4), the omnidirectional image pixel coordinates (u, v) are calculated:

$$u = \frac{2\pi v_p R_{\max}}{h} \cos\left(\frac{2\pi u_p}{w}\right), \quad v = \frac{2\pi v_p R_{\max}}{h} \sin\left(\frac{2\pi u_p}{w}\right), \tag{2.3}$$

where h is the height of the panorama, and w is the width of the panorama, $R_{\max}$ denotes the radius of the omnidirectional image's outer circle, while (u_p, v_p) are respective pixels of the reconstructed panoramic image.

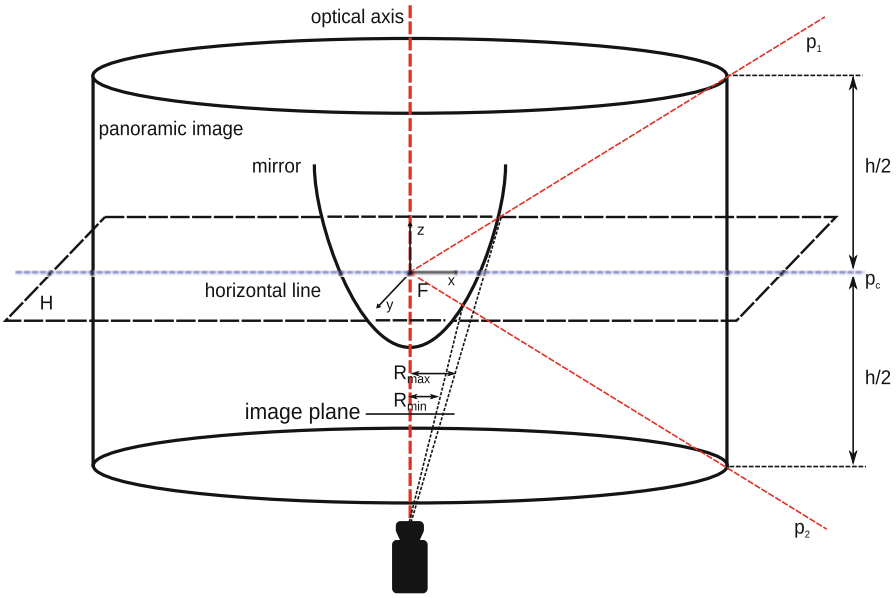

Fig. 2.4 Geometry of the panoramic image surrounding the catadioptric camera

However, the formulas (2.3) do not consider the calibrated parameters of the catadioptric camera. Because of that the panoramic image is not compatible with the field of view of the perspective camera image. To achieve this compatibility, our rectification algorithm has to locate the horizontal line ($\mathbf{p}_c$ point in Fig. 2.4) of the panoramic image. The correctly located horizontal line should be on the same elevation as the optical center of the curved mirror. In practice, it means that pixels taken from the central row of the panoramic image should have zero z coordinates. The half-lines $\mathbf{p}_1$ and $\mathbf{p}_2$ go through the upper and the lower rim of the cylinder, respectively. For further processing, especially for creation of stereo-pair, it is very important that the height h of the cylinder (in pixels) equals the height resolution of the perspective camera image. Next, all pixels (i.e. their coordinates) from the panorama's cylindrical surface are re-projected back into the undistorted omnidirectional image. To achieve this, a method presented in (2.2) is used, which is based on the inverse mapping. The last step in the corrected procedure for panoramic image creation is calculation of the pixel coordinates in the omnidirectional image by formula:

$$u = \rho_v(v_p)\cos\left(\frac{2\pi u_p}{w}\right), \quad v = \rho_v(v_p)\sin\left(\frac{2\pi u_p}{w}\right). \tag{2.4}$$

where $\rho_v = f(v_p)$ denoted the distance between the point's projection and the center of the omnidirectional image. This parameter is computed separately for each row of the panorama. The coordinates u and v are considered in the range between the $R_{\min}$ and $R_{\max}$ radius. The minimal radius $R_{\min}$ is determined by the blind area in the omnidirectional picture.

Fig. 2.5 Rectified panoramic images: constructed using only the inverse projection (**a**), and constructed using our improved method (**b**)

Correct image rectification is a time-consuming process. Hence, to do all calculation in real-time, the OpenCV function `remap()` is used, which is supported by CUDA on the Jetson platform. In this parallelized version the reconstruction of a panoramic image takes only 0.85 ms. Example results of panoramic image reconstruction are shown in Fig. 2.5. The image constructed considering the calibrated parameters of the system and the correct horizontal line location (Fig. 2.5b) looks more natural, and the relations between the height of particular objects seen in the image are preserved, unlike in the simple method (Fig. 2.5a).

2.4.3 Virtual Camera

To calculate the distance between the sensor and an object of unknown size, it is necessary to have two views of this object, which are related by known extrinsic parameters, i.e. the rotation and translation between the cameras that produced these views. To accomplish this task we define a virtual camera, which provides the field of view similar to the actual perspective camera of the hybrid sensor. The coordinate system of the virtual camera has the origin located in the optical center of the curved mirror (Fig. 2.6). The focal length and resolution are chosen purposefully to yield images that are geometrically similar to the perspective ones. While a similar idea was used in [13], our virtual camera image is defined directly from the panoramic image constructed in real-time, which makes the computations much faster. We take advantage from the fact that the panoramic images are reconstructed in real-time in our sensor, and they make a perspective-like view of the scene readily available in the 360° field of view. Thus, we only need to create from the panorama a virtual image that is geometrically compatible with the actual perspective image. The virtual image is created by projecting a ray from the center of the curved mirror towards the cylindrical surface. This ray determines the area on panoramic image defined by the requested resolution of the virtual camera (compatible with the physical one). Pixels from a proper are defined on the panoramic image are taken as a representation of pixels in the virtual camera image.

The calibration procedure for the virtual camera is the same as the one we use for the physical perspective camera. We used the same toolbox [5] to get the camera matrix $\mathbf{K}_v$.

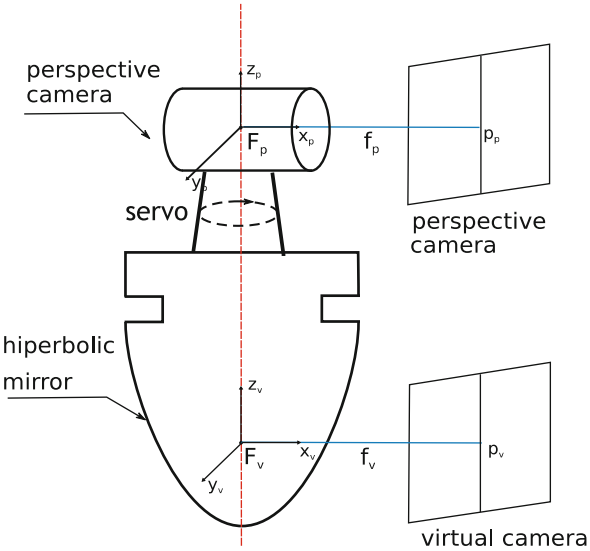

Fig. 2.6 Geometric relations between our sensor and the virtual camera image

2.4.4 Calibration Between the Subsystems

To know the geometric relations between the two subsystems necessary for the stereo distance measurements we perform extrinsic calibration between the perspective and the virtual camera. The results are extrinsic parameters of the stereo pair. We treat the perspective camera and the virtual camera as a stereo pair, unlike [6], where the perspective and the omnidirectional camera are calibrated together. Our approach avoids the use of any special calibration equipment, and allows us to use the standard calibration software. Therefore, having defined the virtual camera, we assume that we have pair of cameras, which are properly calibrated in terms of their intrinsic parameters, as the $\mathbf{K}_p, \mathbf{K}_v$ matrices and distortion models are already computed. The next step of the calibration is calculation of the extrinsic parameters of the stereo pair. To accomplish this, we use again the camera calibration toolbox [5]. We take a new series of images, which contains the chessboard pattern seen in many different positions over the common field of view of both cameras. The transformation of the coordinate system between the perspective and the virtual camera is computed from the corresponding points of the calibration pattern. This relation is described by the rotation matrix $\mathbf{R}_{s(3\times3)}$ and the translation vector $\mathbf{t}_s = [t_x, t_y, t_z]^T$. The last step of the hybrid vision sensor calibration is calculation of the essential matrix [9]. The essential matrix is computed based on rotation and translation between the images from both cameras:

$$\mathbf{E} = \mathbf{R}_s \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}. \tag{2.5}$$

2.5 Peripheral Vision in the Hybrid Sensor

The concept of a stand-alone passive vision sensor comes from the observation, that some mobile robots (e.g. small walking machines) cannot allocate enough computing power to the perception and environment modelling tasks. Therefore, they may benefit from a sensor that provides the robot with pre-processed navigation cues, such as location of obstacles or direction of collision-free motion. The software described so far in this chapter converts raw frames from the omnidirectional camera into geometrically correct panoramic and virtual camera images, but does not support navigation directly. Hence, this section describes how the omnidirectional images can be used to support selected navigation-related functions of the robot, providing the machine with a rough analogy of the peripheral vision in animals. Our peripheral vision enables the robot to detect obstacles and moving objects, track objects, and focus the perspective camera on a selected object. This last function demonstrates benefits from the cooperation between the peripheral and central vision capabilities in our hybrid sensor. Although the algorithms behind these functions are simple in general, we use them to demonstrate that our sensor is an efficient platform to implement various image processing algorithms, also these that benefit a lot from parallel processing on a GPGPU.

2.5.1 Detection of Objects

Rapid detection of changes occurring in the environment is crucial to animals, as it is related to their predatory behaviors or the ability to avoid other predators and various natural hazards. Also mobile robots may benefit from the ability to quickly detect changes in the observed scene. Therefore, the main peripheral vision function implemented in our sensor is detection of moving objects in the omnidirectional images. Moving objects are segmented from the background in real-time using the Background Subtraction Library (BGSLibrary) [32]. This function supports human–robot interaction (e.g. detection of people approaching the robot), surveillance applications, and multi-robot systems, where quick detection of obstacles and other robots is required [14, 28]. The BGSLibrary offers an easy-to-use software framework integrated with OpenCV. It makes possible to discriminate moving objects from the background, providing that the camera is static while acquiring a pair of images. The library contains implementation of several algorithms that support different tasks, such as video analysis. From these

algorithms, we chose two techniques of relatively low computational complexity, namely the FrameDifferenceBGS and SigmaDeltaBGS.

FrameDifferenceBGS is very simple, as it only compares the query image to the one acquired in the previous time instance, and then extracts the moving objects by marking areas where the difference of color is larger than a given threshold. SigmaDeltaBGS algorithm is somewhat more complicated, as it attempts to estimate parameters of the observed background, which is assumed to be approximately uniform (e.g. a flat floor). This algorithm produces fewer artifacts than the simple frame difference, but is slightly more computationally demanding. By default the BGSLibrary is running on a CPU, and it is not compatible with CUDA, hence it cannot benefit from a GPGPU. Because in the robotic applications real-time processing is a must, we adapted the used BGSLibrary algorithms to use a version of OpenCV that is supported by CUDA. Eventually, we were able to exploit the GPGPU readily available in the Jetson platform.

2.5.2 Tracking of Objects

Some of the moving objects that could be extracted from background by our change detection functions may be important enough to be tracked for longer time, e.g. to determine their speed and trajectory. To track an object the sensor has to determine some of its perceptual properties, to make this object distinguishable among others. The simplest property that can be easily distinguished is color. An implementation of this concept on the hybrid field of view sensor was presented in [22]. For the sake of speed, detection of objects having a specific, user-defined color was implemented on the raw omnidirectional images applying thresholding in the Hue-Saturation-Value (HSV) color space. The location of an area of the defined color is then converted to the polar coordinate system, with the origin in the center of rotation of the moving camera servo. This gives the perspective camera its reference angle, which is compared to the current angle of rotation from the camera servo. The computed rotation angle is the one that brings the camera to the target heading in the shortest time.

Because the simple tracking procedure can be applied only to bright-colored objects, its practical use on a mobile robot is limited. Therefore, we implemented on the hybrid field of view sensor also a more advanced tracker employing optical flow and particle filter for the tracking procedure. This function makes it possible to track previously unknown objects having arbitrary shapes and colors, as long as they move at a reasonable speed and stand out visually from the background (Fig. 2.7). The new tracker is based on the algorithm presented in [29] with some improvements. The computation of optical flow is implemented as a parallelized version of the Farnebäck algorithm [7], which quickly yields a vector field with the detected velocity vectors of pixels that have moved between two consecutive frames. Then, a filter with 60 particles is initialized around the target object, which has to be designated by the user with a bounding box. As in the original algorithm [29] the particles are described

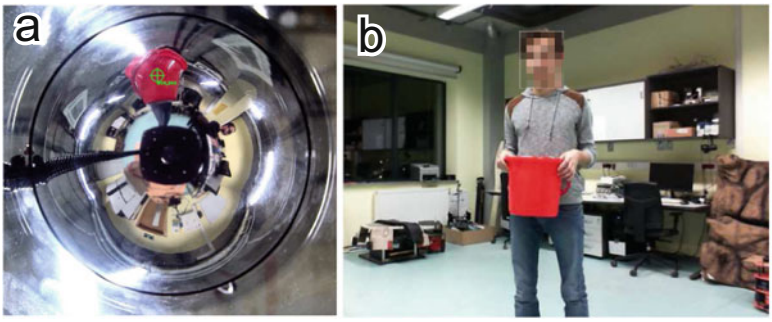

Fig. 2.7 Simple, color-based object tracking: an object (red bucket) detected object in the raw image (**a**), a perspective camera image taken after focusing on this object's location (**b**)

in polar coordinates that are natural for an omnidirectional image. The weights of particles are computed taking into account differences between the velocity vectors of the particles (i.e. image pixels where the particles are located) and the velocity vector of the target, which is known from the previous iteration. Moreover, the similarity of the Hue component of the HSV color model is considered when computing the weights. The resampling step draws a new set of particles from the weighted ones, favoring particles with higher weights, which replace those of lowest weights. As a result, the filter converges in few iterations and the best particles track the moving target.

2.5.3 Avoiding Obstacles

Obstacle avoidance is an essential function in every mobile robot. A robot has to detect any objects that may pose a danger when it is attempting to move towards the given goal. Obstacles may be detected by range or visual sensors, but the avoidance task becomes more efficient if the robot perceive the objects that surround its body without a need to rotate. Therefore, peripheral vision provided by the omnidirectional part of our sensor camera is particularly suitable to indicate the presence of obstacles around the robot in real-time. This concept has been implemented on a compact, low-cost legged robot [36], which got equipped with the first prototype of our vision sensor.

The obstacle detection and avoidance method is inspired by the popular vector field histogram (VFH) algorithm [35]. This algorithm can be directly used for sensors which measure the distances between the robot and the surrounding objects, for example sonars or 2D laser scanners. Based on this data, a local map of the local environment is created. However, our version of the algorithm works using only data from the omnidirectional part of the hybrid sensor. All calculations should be performed in real-time, so the robot can pursue the obstacle avoidance task. Potential

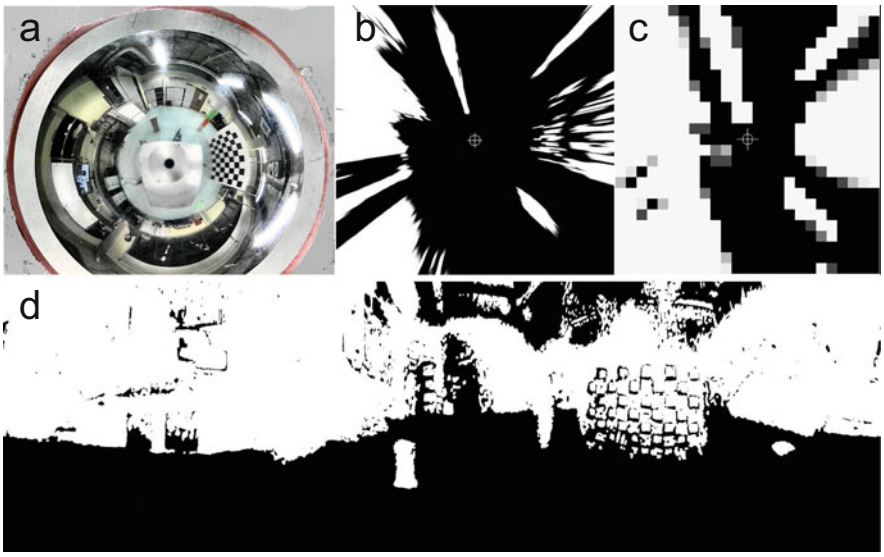

Fig. 2.8 Construction of the occupancy grid from the omnidirectional images: raw image (**a**), data projected to a horizontal 2D image (**b**), final occupancy grid (**c**), and the binary panoramic image after filtration (**d**)

obstacles are recognized on the panoramas rectified in real-time, through background removal. Next, rough distances to them are calculated directly from these images to create an occupancy grid representation of the surroundings. This is possible, because in the panorama objects that are closer to the sensor are located in the lower part of the image, while the upper part depicts more distant objects.

Also because of speed requirements, the panoramic images are constructed in a slightly different way than in the general case (see Sect. 2.4.2). Namely, the background is removed from raw omnidirectional pictures (Fig. 2.8a) using the color information, and only these pixels that represent obstacles are transferred to the panorama. Using a defined color in the HSV color space is a very background removal fast method, and it does not need a good estimate of the robot's ego-motion, which is unavailable in a legged robot. On the drawbacks side, we have to assume that the background is of approximately uniform appearance. Although combining this function with the BGSLibrary functions is possible, we have found that the simple method works well indoors, while it is much faster and easier to parallelize using CUDA. On the legged robot, the areas that can show the body or the swinging legs of the machine are masked by proper shapes directly on the omnidirectional image [36]. The omnidirectional images with masked background and the robot's element that may be treated as close objects are then binarized and converted to panoramic images (Fig. 2.8d). The binary panoramas are treated with erosion and then dilation morphological operators in order to eliminate small, isolated pixel blobs. The next step is to fill the 2D local occupancy grid of the environment with the information

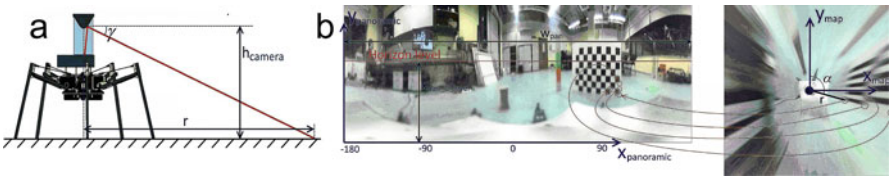

Fig. 2.9 Configuration of our sensor on the walking robot (**a**), panoramic image data projected into the 2D local environment representation (**b**)

from the prepared images. In the pictures obstacles are marked as white areas, and free areas are marked black. The panoramic image coordinates that have to be used to find information about the occupancy of the cells in the grid are defined as:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} \frac{\alpha w_{\text{pan}}}{360^\circ} \\ x_{\text{pan}0} - g_{\text{pan}}\gamma \end{bmatrix}, \tag{2.6}$$

where: w_{pan} is the width of the panoramic image (pixels), g_{pan} defines the vertical resolution of the panoramic image (pixels/1°), and $x_{\text{pan}0}$ defines the elevation of the horizon line (Fig. 2.9a). The angles α and γ are given by formulas:

$$\alpha = \arctan\left(\frac{x_{\text{map}}}{y_{\text{map}}}\right), \quad \gamma = \arctan\left(\frac{h_{\text{camera}}}{\sqrt{x_{\text{map}}^2 + y_{\text{map}}^2}}\right), \tag{2.7}$$

where x_{map} and y_{map} are coordinates of the occupancy grid, and h_{camera} is the elevation of the sensor center measured from the floor. This parameter yielded by the robot's controller, because in a legged machine it depends on the legs configuration (Fig. 2.9b).

The local grid map that is a robot-centric representation of the environment accumulates the occupancy information extracted from the prepared panoramic images. Using this intermediate representation makes it possible to accumulate the information related to obstacles in short time windows, and avoids the necessity to compute the control commands using highly uncertain data. The coordinates of the pixels that represent obstacles are transformed into an instantaneous and local horizontal image—a form of simple map (Fig. 2.8b). Next, the coordinates of obstacles are employed to update the robot-centered occupancy grid. The occupancy grid is attached in the origin of the vision sensor coordinate frame, and its size is 5 m×5 m, with 0.2 m×0.2 m cells (Fig. 2.8c). Following the idea of original VFH we increase the cell occupancy by a fixed value (in the experiments the value of 3 was applied) whenever a pixel representing an occupied area (i.e. obstacle) is transferred into this particular cell. If the transferred pixel represents an empty area, the cell occupancy is decreased by a smaller value (value of 1 was used). However, the occupancy counter is bounded for each cell: it cannot exceed 25 or drop below zero. Then, the one-dimensional polar histogram is built upon this occupancy grid, exactly

as in the original VFH. This histogram is also attached in the origin of the sensor coordinate frame (coincides with the center of the robot), which makes it possible to select the steering direction that avoids all obstacles, but is the one closest to the direction to the goal.

2.6 Central Vision in the Hybrid Sensor

The central vision in animals and human beings serves mostly the needs of accurate interaction with particular objects in the environment, supporting, e.g. grasping of objects. However, our sensor is dedicated to mobile robots, and the central vision serves mostly navigation tasks, such as landmark-based localization [20]. Hence, the main function is accurate measurement of distances to selected objects.

The distances are measured employing an unorthodox stereo vision setup, in which one image in the stereo pair is yielded by the perspective camera directed towards the chosen object, but the other one is synthesized from the omnidirectional image by our virtual camera. Assuming that both cameras are calibrated with respect to intrinsic parameters, we need to relate the perspective camera coordinates to the coordinate system of the virtual camera by extrinsic calibration, as described in Sect. 2.4.4. Once the extrinsic parameters are known, we compute the projection matrices for both cameras in the stereo pair. The projection matrix of the virtual camera reduces to $\mathbf{P}_v = \mathbf{K}_v[\mathbf{I}|\mathbf{0}]$, because we assume that the coordinate system of this camera is attached in the origin of the coordinate frame of the stereo pair. Then, the projection matrix of the perspective camera is calculated. This matrix accounts for the rotation and translation between the two cameras: $\mathbf{P}_p = \mathbf{K}_p[\mathbf{R}_s|\mathbf{t}_s]$. A point in the 3D scene $\mathbf{p}$ is related by the projection matrices to its counterparts $\mathbf{p}'_v$ and $\mathbf{p}'_p$ in the 2D images obtained from the virtual camera, and the perspective camera, respectively:

$$\mathbf{p}'_v = \mathbf{P}_v\mathbf{p}, \quad \mathbf{p}'_p = \mathbf{P}_p\mathbf{p}. \tag{2.8}$$

Therefore, we can reconstruct the position in 3D of the point $\mathbf{p}$ from its projections on the undistorted images from both cameras. The stereo distance computation is accomplished using the optimal triangulation method [8], which runs in real-time on the Jetson board in our sensor. It should be noticed that this method has been chosen mostly due to its computation speed advantage and simple implementation, while more recent and advanced methods exist, e.g. based on neural networks [18]. The GPGPU available in our sensor makes it possible to implement such a method in the future.

Prior to the stereo computations, we have to determine the matching point features. These features are located on the images from both cameras, but they represent the same points in 3D. The computer vision literature lists a number of methods to determine point correspondences in stereo vision [9]. Taking into account that the images produced by the virtual camera are of relatively low resolution,

because they are only up-sampled to the resolution compatible with the perspective camera, we employ the point descriptors to find the corresponding features. The descriptor vectors catch the appearance of the local neighborhood of each feature. They are commonly used to match point features in robot navigation, and are characterized by a good trade-off between the matching efficiency and the computing power requirements [27]. We have implemented three alternative feature matching procedures using SIFT, SURF, or ORB detector/descriptor pairs. The use of sparse point features yields "sparse" stereo information, as the position is computed only for a certain number of features. This is, however, acceptable for most navigation algorithms that natively employ sparse feature maps [31].

Point features are detected in both images, and then described using one of the detector/descriptor pairs. The coordinates of the feature points are undistorted and normalized. Then, we attempt to match points from both images minimizing the distances (Euclidean in the case of SIFT and SURF, and Hamming in the case of ORB) between the descriptor vectors associated with these points. Once the initial associations are established, the symmetrical reprojection error is calculated:

$$e_{\text{rep}} = \max\{\mathrm{d}(e_j, (u_i^v, v_i^v)), \mathrm{d}(e_i, (u_j^p, v_j^p))\}, \tag{2.9}$$

where u_i^v and v_i^v) denote the normalized coordinates of the i-th $\mathbf{p}'_v$ point, u_j^p and v_j^p are coordinates of the j-th $\mathbf{p}'_p$ point, while the Euclidean distance of a point y to the line x is denoted by $\mathrm{d}(x, y)$, and e_i, e_j are epipolar lines computed from the essential matrix (2.5):

$$[e_{i_x}, e_{i_y}, e_{i_z}]^T = \mathbf{E}[(u_i^v, v_i^v, 1)]^T, \tag{2.10}$$

$$[e_{j_x}, e_{j_y}, e_{j_z}] = [(u_j^p, v_j^p, 1)]\mathbf{E}.$$

If the error e_{rep} (2.9) is smaller than a fixed threshold, the match gets accepted. The paired features are then used to calculate the distances in the 3D scene.

2.7 Experimental Results

2.7.1 Peripheral Vision

Peripheral vision functions have been tested in the tasks of obstacle avoidance, detection of dynamic objects, and tracking of both specific color and arbitrary shape/color objects. Some of these experiments are documented on the accompanying video material.[1]

[1] http://lrm.cie.put.poznan.pl/bioinspsens.wmv.

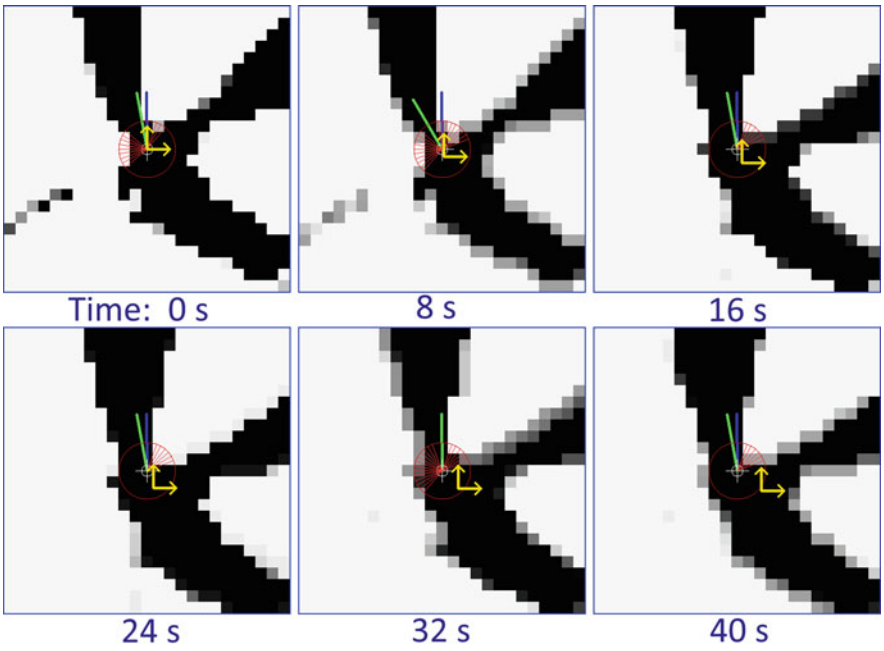

Fig. 2.10 Visualization of the grid maps produced in our sensor while computing a sequence of steering directions to the goal

The more compact version of the sensor, based on the Jetson TK-1 board was tested on a legged robot. The robot walked on a flat floor in a lab, avoiding different types of obstacles, including specially prepared cardboard boxes and tubes, as well as the usual lab equipment. Robot-centered grid maps updated in sequence during this experiment is depicted in Fig. 2.10. The direction to the goal defined by the human operator is shown in Fig. 2.10 by blue arrows, green arrows depict the steering direction obtained from the VFH algorithm, and distances used in the polar histogram to detect obstacles are denoted by red circles. An important improvement in the processing speed has been achieved for this algorithm owing to the use of parallel processing on the Jetson's GPGPU. Namely, if projection of the detected obstacles from the panoramic image to the robot-centered map was implemented on the Jetson's KT-1 CPU, this operation took 3.01 ms for a single image, but the CUDA implementation using the Kepler GPGPU required only 0.45 ms for the same operation.

Also the real-time detection of moving objects through background discrimination was tested on the Jetson TK-1 variant of the sensor. The FrameDifferenceBGS algorithm implemented on the Jetson TK-1 CPU required 19.8 ms per frame. This is enough to detect slowly moving objects, however, the embedded Jetson platform is much slower in this task than a desktop computer (PC with i7 at 2.3 GHz), which took only 8.5 ms per frame. Unfortunately, the execution time decreased only minimally, to 13.7 ms, when the FrameDifferenceBGS algorithm was re-

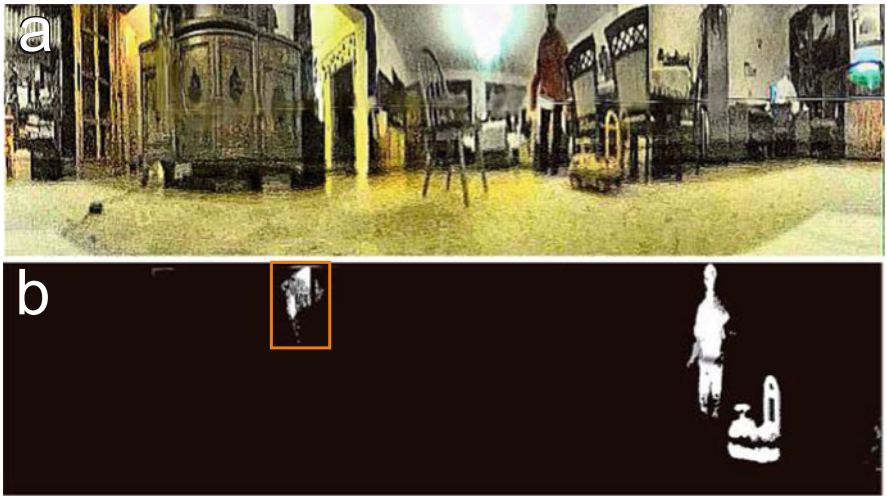

Fig. 2.11 Dynamic objects segmented from the background by the SigmaDeltaBGS method: the panoramic image (**a**), and the detected objects denoted by white pixels (**b**). The orange rectangle surrounds some wrongly identified pixels

implemented using CUDA and ran on the Jetson's GPGPU. The reason for this result is a large number of data transfer operations in the considered algorithm, compared to the relatively simple computations. Such tasks do not benefit much from parallel processing architecture. For the more complex SigmaDeltaBGS the difference in processing speed between the Jetson TK-1 implementation and its desktop PC counterpart was smaller—processing of a single frame took 239.6 and 198.7 ms, respectively. However, in this algorithm data transfer constitutes much smaller fraction of the operations. Figure 2.11 shows example images with a person and a toy cart detected by the SigmaDeltaBGS algorithm. Notice that very few outliers are present (Fig. 2.11b).

Tracking was tested on several objects, including simple balls, toys, and people surrounding the sensor [17]. Figure 2.12a demonstrates the behavior of the particle filter: from the computed optical flow field (left) to the converging particles (shown as pink rectangles). The ability to track an object of complicated shape and color is shown in Fig. 2.12b, where a toy giraffe (pulled on a rope) is tracked by the filter. Real-time performance is achieved due to parallel implementation of both the optical flow and the particle filter on the GPGPU.

2.7.2 Central Vision

For the evaluation of the main central vision function, the stereo-based distance measurements, we performed a series of experiments in a home-like environment.

Fig. 2.12 Particle filter tracking a ball (**a**), and the tracked positions of a toy denoted by bounding boxes (**b**)

Fig. 2.13 Different images for one of the scenes: perspective camera image (**a**), omnidirectional camera image (**b**), and panoramic image (**c**)

At first, the distance measurements accuracy was determined. Then, we tested extensively the descriptor-based feature matching in sparse stereo to determine which detector/descriptor pair suits best the requirements of our sensor. We have set up four simple scenes, gathering furniture pieces and other common objects (boxes, pillows) into sets of two or three items (Fig. 2.13). The ground truth distances between the scene objects and the sensor were measured using a meter tape, assuming that the origin of the coordinates is located in the center of the curved mirror.

At first we evaluated the distance measurements using the SIFT detector/descriptor pair (Fig. 2.14a), because SIFT is considered the "golden standard" of the point feature descriptors if real-time performance is not a concern [20]. We measured distances to a number of features detected on the observed objects (Fig. 2.14b). The objects in the scene had flat front vertical surfaces. Hence, we averaged the distance measurements for all the features appearing on the particular vertical surface to produce the quantitative results shown in Fig. 2.15.

One can easily deduce from these plots that the range measurement errors depend on the distance to the observed object. It is also visible that these errors become minimal in the mid-range of the measured distance. This result coincides with

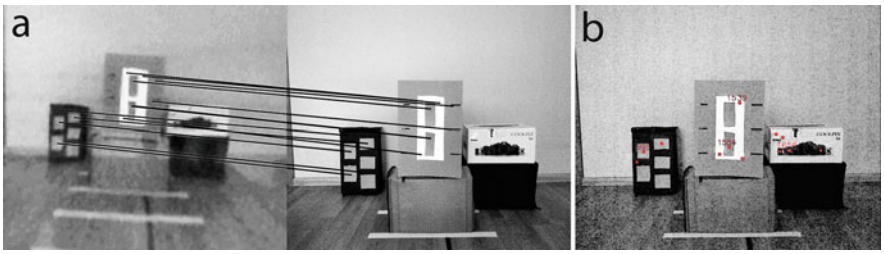

Fig. 2.14 Sparse stereo in the same scene: associated SIFT features (**a**), and 3D feature points located on the scene (**b**)

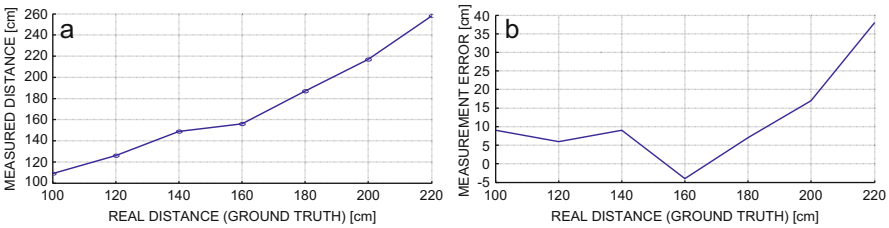

Fig. 2.15 Dependence between the range measurements and the ground truth distance (**a**), and the dependence between the ground truth distance and the distance measurement error (**b**)

the range, where the interpolation errors in the panoramic images are minimal. Apparently, the number of correct matches depends on the distance to object. The number of correctly matched SIFT features varied from 5 to 17 in the scene depicted in Fig. 2.13. The largest number of correct matches was observed at 1.8 m to the middle object. The number of matches is largest for the measured distances from 1.6 to 1.8 m, which coincides with the range of minimal distance errors. However, the number of detected features was typically higher for more distant objects.

Table 2.1 shows the number of correctly matched features for the SIFT, SURF, and ORB detector/descriptor pairs, four different scenes, and three representative measurement distances. Example matches are visualized in Fig. 2.16 for the scene no. 3 at the distance of 1.4 m.

In the same experiments the time needed to process a pair of images (perspective and virtual) and to compute the distance was evaluated. The measured time (in seconds) includes creation of the virtual image, point feature detection, description and matching, and triangulation, but does not cover the reconstruction of the panoramic image. The processing time t_c shown in Table 2.2 was measured on the Cortex-A15 CPU for all the detector/descriptor pairs. For the SURF and ORB features, which have OpenCV implementations supporting CUDA, the time t_g, also included in Table 2.2, was measured on the GPGPU. Only the use of GPGPU allowed the sensor to accomplish the central vision process in real-time. The processing time depended mostly on the detector/descriptor type, however, the matching time depends also on the number of detected features.

Table 2.1 Performance of the detector/descriptor pairs in sparse stereo measured by the number of correct matches

Detector/descriptor	Scene 1	Scene 2	Scene 3	Scene 4
SIFT 1.2 m	5	22	19	33
SURF 1.2 m	7	10	13	9
ORB 1.2 m	3	0	0	3
SIFT 1.6 m	15	36	25	36
SURF 1.6 m	14	21	51	25
ORB 1.6 m	2	3	4	4
SIFT 2.0 m	16	10	24	17
SURF 2.0 m	40	18	8	3
ORB 2.0 m	2	4	1	2

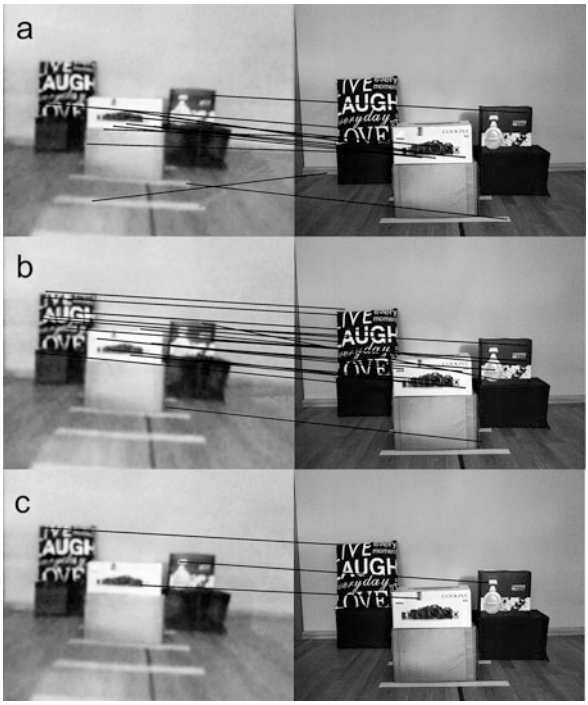

Fig. 2.16 Features associated using SIFT (**a**), SURF (**b**), and ORB (**c**) in the Scene no. 3

2.8 Conclusions

This chapter presents a stand-alone, passive vision sensor of hybrid field of view that draws inspirations from the vision systems of insects and animals. The sensor is hosted by a recent single-board computer that provides enough computing power to implement a wide palette of image processing algorithms supporting robot navigation. Moreover, the high computing power of the sensor and its open-source software architecture, exploiting the common CUDA and OpenCV libraries, make it possible to implement new functions required by the task at hand.

Table 2.2 Total processing time for stereo-based distance measurements on the Jetson TK-1 CPU and GPGPU

Detector/ descriptor	Scene 1		Scene 2		Scene 3		Scene 4	
	t_c	t_g	t_c	t_g	t_c	t_g	t_c	t_g
SIFT 1.2 m	7.94	–	7.90	–	7.92	–	7.65	–
SURF 1.2 m	5.70	0.44	2.51	0.22	2.72	0.21	2.99	0.23
ORB 1.2 m	2.75	0.11	0.59	0.02	1.02	0.04	1.00	0.04
SIFT 1.6 m	7.64	–	7.79	–	7.62	–	7.52	–
SURF 1.6 m	2.22	0.17	2.86	0.22	2.45	0.19	2.46	0.19
ORB 1.6 m	1.75	0.07	1.50	0.06	1.27	0.05	1.23	0.04
SIFT 2.0 m	7.68	–	7.54	–	7.54	–	7.54	–
SURF 2.0 m	2.34	0.18	2.35	0.18	2.61	0.20	2.33	0.18
ORB 2.0 m	1.72	0.07	1.25	0.05	1.05	0.04	1.29	0.05

The concept of a sensor integrating an omnidirectional camera and a perspective camera is not particularly novel, but we contribute new elements:

- software that implements selected functions of the peripheral and central vision concept on top of the wide field of view vision typical to insects;
- the use of GPGPU for real-time image processing in a low-power, embedded vision system;
- simple yet efficient calibration methodology of the hybrid field of view vision system.

Moreover, this chapter contributes also improved algorithms and results of extensive experimental evaluation. For instance, the reconstruction of panoramic images has been improved to ensure better compatibility between these images and the perspective camera images. On the basis of experiments we have selected the SURF detector/descriptor pair for the sparse stereo vision function in the hybrid field of view sensor. The efficient OpenCV implementation with CUDA support ensures that SURF can be used in real-time on the Jetson platform.

Acknowledgements We would like to thank Nvidia for donating the Jetson TX-1 board within the Academic GPU Grants program. Also the involvement of students pursuing their degree in the Mobile Robots Lab, who assisted with the experiments and preparation of the video material is greatly appreciated.

References

1. Adorni, G., Bolognini, L., Cagnoni, S., & Mordonini, M. (2001). *A non-traditional omnidirectional vision system with stereo capabilities for autonomous robots, AIIA 2001: Advances in artificial intelligence.* LNCS (Vol. 2175, pp. 344–355). Berlin: Springer.
2. Baker, S., & Nayar, S. K. (1999). A theory of single-viewpoint catadioptric image formation. *International Journal of Computer Vision, 35*(2), 175–196.

3. Bakstein, H., & Pajdla, T. (2002). Panoramic mosaicing with a 180 field of view lens. In Proceeding of IEEE Workshop on Omnidirectional Vision (pp. 60–67).
4. Biadala, A., & Czukin, G. (2017). *A hybrid vision system with active field of view selection*. Engineer's degree Thesis, Poznań: Poznań University of Technology (in Polish).
5. Bouguet, J.-Y. *Camera calibration toolbox for Matlab*. http://www.vision.caltech.edu/bouguetj/calib_doc/
6. Cagnoni, S., Mordonini, M., & Mussi, L. (2007). Hybrid stereo sensor with omnidirectional vision capabilities: Overview and calibration procedures. In Proceeding International Conference on Image Analysis and Processing, Modena (pp. 99–104).
7. Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In J. Bigun & T. Gustavsson (Eds.) *Image analysis, SCIA 2003*. LNCS (Vol. 2749, pp. 363–370). Heidelberg: Springer.
8. Hartley, R. I., & Sturm, P. (1995). *Triangulation, computer analysis of images and patterns*. LNCS (vol. 970, pp. 190–197). Heidelberg: Springer.
9. Hartley, R. I., & Zisserman, A. (2004). *Multiple view geometry in computer vision*. Cambridge: Cambridge University Press.
10. Kang, S. B. (2000). Catadioptric self-calibration. In *CVPR* (pp. 201–207).
11. Käppeler, U. -P., Höferlin, M., & Levi, P. (2010). 3D object localization via stereo vision using an omnidirectional and a perspective camera. In Proceeding of 2nd Workshop on Omnidirectional Robot Vision (pp. 7–12).
12. Kozlowski, P., & Drankiewicz, W. (2016). *Hybrid field of view vision for a mobile robot*. Engineer's degree Thesis. Poznań: Poznań University of Technology (in Polish).
13. Lin, H. -Y. & Wang, M. -L. (2014). HOPIS: Hybrid omnidirectional and perspective imaging system for mobile robots. *Sensors, 14*, 16508–16531.
14. Lindner, L., Sergiyenko, O., Rodríguez-Quiñonez, J. C., Rivas-Lopez, M., Hernandez-Balbuena, D., Flores-Fuentes, W., & et al. (2016). Mobile robot vision system using continuous laser scanning for industrial application. *Industrial Robot: An International Journal, 43*(4), 360–369.
15. Menegatti, E., & Pagello, E. (2002). Cooperation between omnidirectional vision agents and perspective vision agents for mobile robots. In M. Gini, et al. (Eds.), *Intelligent autonomous systems* (vol. 7, pp. 231–235). Amsterdam: IOS Press.
16. OpenCV Documentation. http://docs.opencv.org.
17. Plucinska, N. (2018). *Object tracking on omnidirectional images*. Engineer's degree Thesis. Poznań: Poznań University of Technology (in Polish).
18. Rodríguez-Quinonez, J. C., Sergiyenko, O., Flores-Fuentes, W., Rivas-Lopez, M., Hernández-Balbuena, D., Rascon, R., & et al. (2017). Improve a 3D distance measurement accuracy in stereo vision systems using optimization methods' approach. *Opto-Electronics Review, 25*(1), 24–32.
19. Roennau, A., Kerscher, T., Ziegenmeyer, M., Zöllner, J., & Dillmann, R. (2009). Adaptation of a six-legged walking robot to its local environment. In *Robot motion and control*. LNCIS (Vol. 396, pp. 155–164). Heidelberg: Springer.
20. Rostkowska, M., & Skrzypczyński, P. (2015). Improving self-localization efficiency in a small mobile robot by using a hybrid field of view vision system. *Journal of Automation, Mobile Robotics & Intelligent Systems, 9*(4), 28–38.
21. Rostkowska, M., & Skrzypczyński, P. (2016). Hybrid field of view vision: From biological inspirations to integrated sensor design. In *Proceeding IEEE international conference on multisensor fusion and integration for intelligent systems, Baden-Baden* (pp. 653–658).
22. Rostkowska, M., Wąsik, M., & Skrzypczyński, P. (2018) Implementation of peripheral vision in a hybrid field of view sensor. In R. Szewczyk, et al. (Eds.), *Automation 2018, advances in automation, robotics and measurement techniques, AISC 743, Zürich* (pp. 584–594). Heidelberg: Springer.
23. Salinas, C., Montes, H., Fernandez, G., Gonzales de Santos, P., & Armada, M. (2012). Catadioptric panoramic stereovision for humanoid robots. *Robotica, 30*, 799–811.

24. Santos-Victor, J. A., Sandini, G., Curotto, F., & Garibaldi, S. (1995). Divergent stereo in autonomous navigation: From bees to robots. *International Journal of Computer Vision, 14*, 159–177.
25. Scaramuzza, D. (2008). Omnidirectional vision: from calibration to robot motion estimation, PhD Dissertation. Zurich: ETH Zürich.
26. Scaramuzza, D., Martinelli, A., & Siegwart, R. (2006). A toolbox for easy calibrating omnidirectional cameras. In *Proceeding IEEE International Conference on Intelligent Robots & Systems, Beijing* (pp. 5695–5701).
27. Schmidt, A., Kraft, M., Fularz, M., & Domagala, Z. (2013). Comparative assessment of point feature detectors and descriptors in the context of robot navigation. *Journal of Automation, Mobile Robotics & Intelligent Systems, 7*(1), 11–20.
28. Sergiyenko, O. Y., Ivanov, M. V., Tyrsa, V. V., Kartashov, V. M., Rivas-Lopez, M., Herández-Balbuena, D., & et al. (2016). Data transferring model determination in robotic group. *Robotics and Autonomous Systems, 83*, 251–260.
29. Shu-Ying, Y., Wei Min, G., & Cheng, Z. (2009). Tracking unknown moving targets on omnidirectional vision. *Vision Research, 49*, 362–367.
30. Siagian, C., & Itti, L. (2009). Biologically inspired mobile robot vision localization. *IEEE Transaction on Robotics, 25*(4), 861–873.
31. Skrzypczyński, P. (2009). Simultaneous localization and mapping: A feature-based probabilistic approach. *International Journal of Applied Mathematics and Computer Science, 19*(4), 575–588.
32. Sobral, A. (2013). BGSLibrary: An OpenCV C++ background subtraction library. In Proceeding WVC 2013, Rio de Janeiro.
33. Soria, C., Carelli, R., & Sarcinelli-Filhot, M. (2006). Using panoramic images and optical flow to avoid obstacles in mobile robot navigation. Proceeding IEEE ISIE 2006, Montreal (pp. 2902–2907).
34. Ude, A., Gaskett, C., & Cheng, G. (2006). Foveated vision systems with two cameras per eye. In Proceeding IEEE International Conference Robotics & Automation, Orlando (pp. 3457–3462).
35. Ulrich, I., & Borenstein, J. (1998). VFH+: Reliable obstacle avoidance for fast mobile robots. In Proceeding IEEE International Conference on Robotics & Automation, Leuven (pp. 1572–1577).
36. Wąsik, M., Rostkowska, M., & Skrzypczyński, P. (2016). Embedded, GPU-based omnidirectional vision for a walking robot. In M. O. Tokhi & G. S. Virk (Eds.). *Advances in cooperative robotics* (pp. 339–347). Singapore: World-Scientific.
37. Wehner, R., & Wehner, S. (1990). Insect navigation: use of maps or Ariadne's thread? *Ethology, Ecology, Evolution 2*, 27–48.
38. Yagi, Y. (1999). Omnidirectional sensing and its applications. *IEICE Transaction on Information and Systems, 82*(3), 568–579.
39. Yagi, Y., Kawato, S., & Tsuji, S. (1994). Real-time omnidirectional image sensor (COPIS) for vision-guided navigation. *IEEE Transaction on Robotics and Automation, 10*(1), 11–22.