Bio-inspired Active Vision for Obstacle Avoidance

Manuela Chessa, Saverio Murgia, Luca Nardelli, Silvio P. Sabatini and Fabio Solari

Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, Via all'Opera Pia 13, 16145 Genova, Italy

Keywords: Bio-inspired Framework, Active Vision, Non-rectified Disparity Estimation, 3D Reconstruction, Robot Navigation.

Abstract: Reliable distance estimation of objects in a visual scene is essential for any artificial vision system designed to serve as the main sensing unit on robotic platforms. This paper describes a vision-centric framework for a mobile robot which makes use of bio-inspired techniques to solve visual tasks, in particular to estimate disparity. Such framework features robustness to noise, high speed in data processing, good performance in 3D reconstruction, the possibility to orientate the cameras independently and it requires no explicit estimation of the extrinsic parameters of the cameras. These features permit navigation with obstacle avoidance allowing active exploration of the scene. Furthermore, the modular design allows the integration of new modules with more advanced functionalities.

1 INTRODUCTION

Depth estimation from stereoscopic image pairs is a fundamental problem widely discussed in the literature. It is necessary to perform complex tasks such as navigation, scene analysis and interaction with the environment. However, depth estimation is often affected by restrictions such as stereo calibration and rectification, noise, high computational demand and occlusions. Usually, fixed and rectified stereoscopic or RGB-D systems are adopted because of the reduced computational load of the algorithms involved in disparity estimation (e.g. block matching) and 3D reconstruction. For example, RBG-D devices like Microsoft Kinect or ASUS Xtion project a known pattern (Scharstein and Szeliski, 2003) of points on the scene using infrared light. While this allows for faster computation, this strategy is heavily dependent on the scene dimensions and the sensing range is fixed. In (Grigorescu et al., 2011), a robust closed-loop camera pose and scene structure estimation is performed, which, though providing good results, relies on rectification and parallelism of the cameras. All of these solutions are far from being similar to how humans and animals sense of sight works: moreover, they often reduce the degrees of freedom offered by the system.

In (Klarquist and Bovik, 1998), a foveated vision system is proposed, which relies on consecutive fixations of scene features in order to estimate a global reconstruction of the 3D scene: their work, despite having a variable baseline, explains how vergence can greatly help in scene analysis and highlights the limitations imposed by rectified systems (e.g. minimum distance for objects, according to the overlapping region of the view volumes of the cameras).

Thus, despite its complexity, a vergent system can often be a desirable choice: it allows for the adjustment of the sensing range and many common tasks (e.g. object tracking or fixation) can benefit from the use of such a system. Thanks to the growth of the computational power of PCs and the advantages in code parallelisation brought by GPGPU, this approach has been made feasible on current, consumer-grade platforms.

Due to these reasons and since the human vision system copes with the aforementioned limitations, we have based our work on a bio-inspired approach, where:

- The orientation of each camera can be adjusted independently.
- The system structure resembles the modularity of the human vision system, where different areas have different purposes and information is combined at upper levels.
- Disparity is estimated using a cortical model of the primary visual cortex (V1) neurons and their interactions.
- The 3D reconstruction of the scene is computed

in parallel with a Single Instruction Multiple Threads (SIMT) approach.

Following the Early Vision model proposed by Adelson & Bergen (Adelson and Bergen, 1991), many steps are necessary in order to gain a deep understanding of the scene structure, ranging from disparity estimation and 3D reconstruction to image segmentation and blob detection: many solutions have been devised over the years (Chen et al., 2011), each of them based on different assumptions and/or with different strengths, weaknesses and execution times. In any case, their main goal lies in measuring a specific feature of the scene.

When approaching the problem of robot navigation, computer vision is not the only discipline involved: the robot needs to be modelled, all of its sensors have to be analysed in order to characterise its proprioception (position and orientation estimation) and interaction with the external world (e.g. cameras for object detection, tracking and/or avoidance). Finally, every module has to be integrated with the others.

The main contributions this paper provides are:

- Integration of features belonging to the Early Vision model (e.g. disparity and edges) in a modular framework which combines them in order to solve complex visual tasks (e.g. 3D reconstruction in a vergent system, blob extraction, navigation).
- Development of a SIMT approach towards depth estimation, starting from disparity computed through a bio-inspired algorithm.
- Evaluation of the performances (execution times and depth estimation errors) of such platform when equipped with the aforementioned bio-inspired algorithm.

The remainder of this paper describes the proposed system, the flow of data, the experiments and the obtained results.

2 PROPOSED SYSTEM

The proposed system is represented in the block diagram in Fig. 1. It comprises many modules: image acquisition, used to obtain the images from the cameras and to apply undistorting operations, disparity estimation and 3D reconstruction, segmentation and blobs extraction, Navigator module, which combines depth and blob information with the robot state (RobotModel module) to detect near objects and avoid them during navigation. Finally, a GUI module is present which shows a 3D reconstruction of the scene in real-world coordinates, allows orders to be issued to the robot and to set the various parameters which characterise the system.

Great attention has been paid whilst designing this framework in order to keep it as modular as possible, having many pieces of software running at the same time sharing data, and still being easy to update with new modules and features.

Image Acquisition and Segmentation. According to the pinhole camera model (Forsyth and Ponce, 2002), the acquisition module encapsulates all the intrinsic parameters of the cameras and the functions to obtain undistorted stereo images. The system does not rectify the images, and thus it does not need the extrinsic parameters of the stereo rig.

In order to implement an efficient colour segmentation module, colour edge detection was applied as a preliminary step to detect the boundaries between the observed surfaces. Similarly to other approaches (see (Chen and Chen, 2010; Dutta and Chaudhuri, 2009)) our implementation performed the following operations:

- Median filtering, to lessen the effect of noise preserving edges.
- Image gradient computation for every channel, through a derivative kernel (e.g. $[-1,0,1]$) or other operators (e.g Sobel).
- Sum of the norm of the gradients (6 components, X and Y for every channel) for every pixel. To provide a faster execution, we chose $||\cdot||_1$, summing the absolute values of the components.
- Binary thresholding (see Fig. 2, second row) in order to set the strong edges to 0 and the inner regions to 1. The obtained binary map can be then easily labelled with blob detection algorithms.

Blob Extraction. Based on the algorithm proposed by F. Chang in (Chang et al., 2004), component labelling was applied through contour tracing: we developed our own implementation of the algorithm and a library which has been released under the LGPL license (OpenCVBlobsLib[1], originally based on cvblobslib). The original project was enhanced both in terms of performance (implementing a multi-thread algorithm) and functionalities: for example, blob joining capability was added, allowing to link many separate regions to one entity. Regarding the multi thread implementation, the approach can be described as follows:

- Horizontal splitting of the image into *number of Threads* regions.

[1] v1.0 https://code.google.com/p/opencvblobslib/

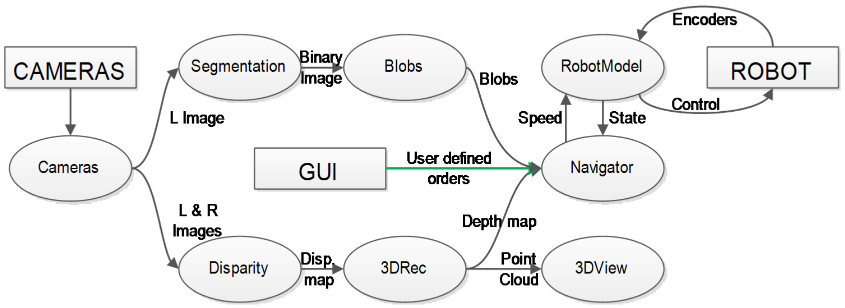


Figure 1: Modules and data flow. Different paths mean parallel tasks.

- Blobs crossing the regions are detected by a single thread detection algorithm, which runs along the separating row.
- Having already labelled the crossing blobs, the other threads can be created and can run in their image regions without worrying about conflicts with other threads (that could happen if two or more would trace the same contour).
- Detected blobs are concatenated in a single array.

Moreover, the library allows to compute geometric properties of the blobs (joined or not): this is fundamental after depth estimation, since it allows to analyse the depth map on a region basis and not per pixel, enabling the robot to perform coarse scene analysis for navigation.

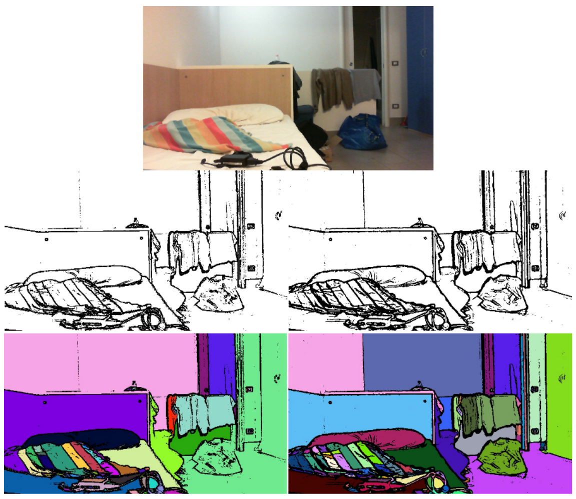


Figure 2: Image segmentation and component labelling. The right pair of images shows our approach (using a 11x11 median filter and threshold value of 21), while the left one shows the result using Sobel operator. Our implementation manages to separate more regions.

Disparity Estimation. Disparity, in a vergent stereo system which purposely loses the horizontal epipolar constraint, is represented by a two-dimensional vector: this further complicates the heavy task of its estimation, yet adds a very important degree of freedom to the vision system, which could perform better in many tasks (e.g. tracking and attentional mechanisms). Bio-inspired algorithms can cope well with such two-dimensional disparity vectors: in particular, we have chosen an algorithm based on the energy model of V1 cortical neurons (Fleet et al., 1996) (its efficient GPU implementation is described in (Chessa et al., 2012)). This architecture is built on a population of binocular simple and complex neurons, the former implemented as a bank of complex-valued Gabor filters, with different orientations in space and phase shifts, the latter as a squaring operation (energy computation) on the output of the simple units. Its fundamental processing steps can be summarised as:

- Linear filtering stage, in which the RFs (Receptive Fields) of the simple S cells are applied to the image through convolution.
- Energy model, where quadrature pairs of S cells are combined, squared and eventually thresholded.
- Divisive normalisation, used to remove noise and to simulate the mechanism of light adaptation.
- Population decoding, in which pools of neurons responses are combined in order to extract disparity information.

Although detected disparities are limited in range depending on the radial peak frequency of the used Gabor filters, and on their spatial support, sub-pixel accuracy in disparity estimation is guaranteed, thus creating very accurate estimates. Moreover, a coarse to fine approach through a Gaussian pyramid is introduced to further extend the range of detectable disparities while maintaining a relatively low computational load for the processing system. Given the intrinsically parallel nature of our visual neural pathways, a GPU implementation of this model is adopted

(Chessa et al., 2012): this dramatically reduces the time needed for a complete disparity estimation ($35\times$ gain in performance compared to the CPU implementation using 1024×1024 images), opening the door for real time systems.

3D Reconstruction. As illustrated in (Chessa et al., 2009), the cameras and robot reference systems can be chosen like in Fig. 3, where the 2 cameras are displaced only along the X axis of the robot coordinate system and their rotations are described with α and β angles: With this choice, the projective equations for the left camera can be written as

$$x^L = f_0 \frac{X_+ \cos\alpha^L + Z \sin\alpha^L}{X_+ \sin\alpha^L \cos\beta^L - Y \sin\beta^L - Z \cos\alpha^L \cos\beta^L}$$
$$y^L = f_0 \frac{X_+ \sin\alpha^L \sin\beta^L + Y \cos\beta^L - Z \cos\alpha^L \sin\beta^L}{X_+ \sin\alpha^L \cos\beta^L - Y \sin\beta^L - Z \cos\alpha^L \cos\beta^L} \tag{1}$$

where f_0 is the focal length of the camera (in cm), α and β respectively represent the pan and tilt angles, $X_+ = X + \frac{b}{2}$ where $b = O^R - O^L$ is the baseline.

With Eq. 1 in mind, assuming that the baseline and the camera pan and tilt angles are known, the projection of a point $F(X,Y,Z)$ onto the left image plane can be computed. Similarly, this process can be applied to the right camera, by substituting the angles and changing X_+ to $X_- = X - b/2$.

Having obtained a good estimate of the disparity vector, it is then possible to relate every pixel with its homologous on the other image. By inverting the projective equations (considering the world coordinates as unknowns) a pair of 2 by 3 linear systems is obtained, one for each camera. The left one can be written as

$$\begin{gathered}(x^L \sin(\alpha^L)\cos(\beta^L) - f_0 \cos(\alpha^L))X \\ -x^L \sin(\beta^L) Y \\ +(-x^L \cos(\alpha^L)\cos(\beta^L) - f_0 \sin(\alpha^L))Z \\ = -1/2 x^L b \sin(\alpha^L)\cos(\beta^L) + 1/2 f_0 b \cos(\alpha^L)\end{gathered}$$

$$\begin{gathered}(y^L \sin(\alpha^L)\cos(\beta^L) - f_0 \sin(\alpha^L)\sin(\beta^L))X \\ +(-f_0 \cos(\beta^L) - y^L \sin(\beta^L))Y \\ +(f_0 \cos(\alpha^L)\sin(\beta^L) - y^L \cos(\alpha^L)\cos(\beta^L))Z \\ = -1/2 y^L \sin(\alpha^L)\cos(\beta^L) b + 1/2 f_0 \sin(\alpha^L)\sin(\beta^L) b\end{gathered} \tag{2}$$

and by itself could not provide a unique solution for the problem. By combining it with the linear system relative to the right camera (very similar in structure to the left one), assuming that F is the solution for both, a 4 by 3 linear system is obtained.

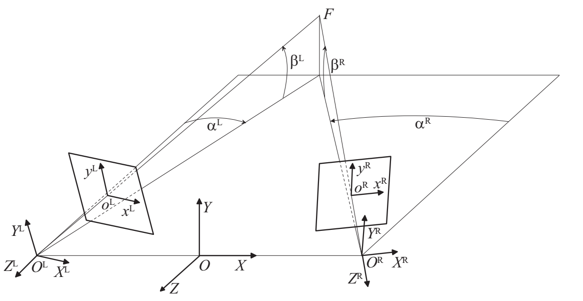


Figure 3: Cameras and robot reference systems.

In an ideal case, this system of equations provides the intersection between the 2 lines starting from the cameras origins, passing through the points on the image planes and reaching F. However, since disparity estimation, angle measurements and pixel quantisation introduce errors, these lines may not intersect at all. Thus, an ordinary least squares solution was adopted, by minimizing the norm $||A\hat{F} - b||_2^2$. The solution is then $\hat{F} = (A^T A)^{-1} A^T b$ where $\hat{F}$ is an estimate of F.

Since each system is independent from the others, an SIMT solution was devised and implemented on the GPU, with different threads handling separate linear systems. In this way, VGA-resolution (i.e. 640x480 pixels) images, which involve around 10^5 independent systems, can be easily processed in real time (10ms in GPU vs 60ms in CPU), transforming disparity information in depth (see Fig. 4).

Robot Modelling and Control. In order to provide effective and easy to use controls, the robot was modelled following the unicycle model (Matveev et al., 2013): thus, two PID controllers were designed to control linear and angular speed, using the linear distance and the heading difference as error functions.

The robot state (position, heading direction and cameras angles) is constantly updated by a dedicated thread which reads the encoders and computes the new state for every iteration. In this way, an update routine runs in the background, supplying the whole system with a constantly up-to-date state of the robot.

Navigation. Navigation is implemented through position goals, whose coordinates are taken with respect to the robot starting position and orientation (i.e. at the start of the program, the robot sets the origin of the world reference system to its position). In this step, a first integration of the two streams of information is done: segmentation/object information is combined with the depth map in order to analyse it based on the different objects in the scene, rather than just by pixel. By checking the central area of the image

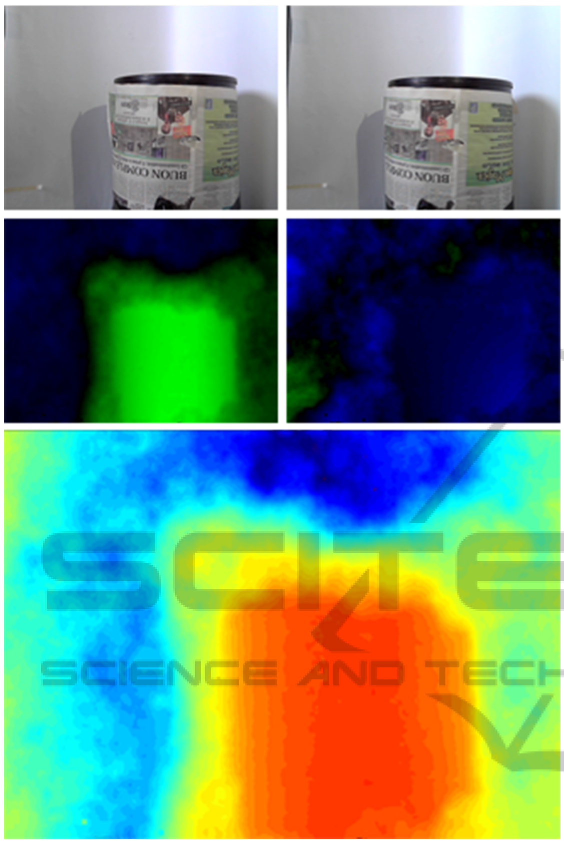


Figure 4: Top row: RGB frames. Middle row: horizontal and vertical disparities, positive values encoded in blue and negative in green. Bottom row: Depth map, close pixels represented by warm colours.

for blobs under a certain depth value, the robot can detect obstacles, and by evaluating the lateral regions (see Fig. 5 on the left for the 3 regions) it can effectively set new intermediate waypoints to reach before its prefixed target. Having detected which of the 2 areas has the furthest objects (i.e. the minimum distance in that region is greater than in the other one), the robot proceeds to set an intermediate goal, whose distance from the robot is proportional to the distance from the issued target and whose angle (with respect to the robot heading) has value of $\pm\pi/10$, with the sign coherent with which of the 2 lateral regions has been selected. Moreover, in case the robot finds other obstacles before reaching its intermediate goal, it will overwrite it with a new one, thus avoiding the situation in which a self-generated goal falls over an obstacle.

Thread Execution. A modular system often requires for many operations to be executed simultaneously: parallelism is a solution adopted in every biological system, and has proved itself very effective in handling multiple tasks. Similarly, our system is characterised by many threads, some of them created to follow the data flow pattern and others to provide constant updates about the state of the robot. We have:

1. RobotModel thread, which communicates with Navigator, reads the sensors, updates the robot position and translates orders (i.e. speed) into robot parameters.
2. Navigator thread, which applies the PID controllers for target-reaching, draws the 2D map, and generally coordinates the movement process (e.g obstacle avoidance).
3. GUI thread, which controls I/O with the user and displays windows.
4. Ogre3D thread, which controls and manages the 3D view window.
5. Main thread, designed to coordinate the whole data processing.

This last thread, after acquiring the images, immediately creates two child threads, one entrusted with disparity estimation and 3D reconstruction (computations that happen almost completely on the GPU), and the other with colour segmentation and blobs extraction (which takes place only on the CPU). Consequently, every resource the PC can supply is exploited, thus maximizing the performance. Focusing on the actual implementation of the system, data is passed between modules through pointers, when possible. In this way, unnecessary memory copying operations are avoided, and the overall performance benefits from this approach.

GUI. With the provided GUI module, the user can effectively visualise all the intermediate results of data processing, issue orders, manually set cameras angles and algorithms parameters and choose one of the implemented disparity estimation algorithms, if more than one is present. Moreover, a 2D map shows the robot position and orientation in real time and allows for order issuing. Finally, a 3D engine renders the 3D model of the robot, updated in real time, with the reprojected pixels from the depth map (see Fig. 5).

3 EXPERIMENTAL SETUP

The hardware used in our tests consists of a consumer-grade PC and a K-Team Koala robot[2] equipped with a pan-tilt module with two *off the shelf* webcams and encoders in every motor. All the computations are performed on the aforementioned PC.

[2] http://www.k-team.com/mobile-robotics-products/koala

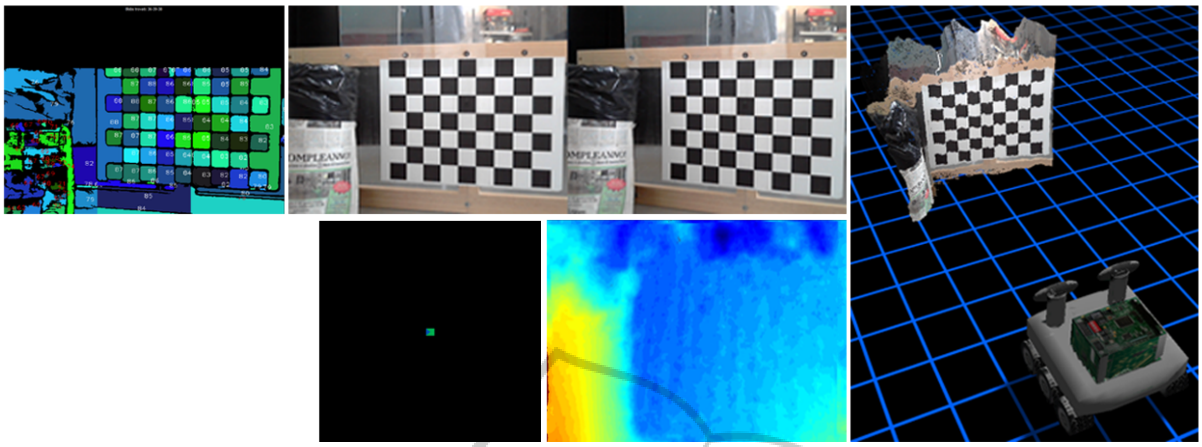


Figure 5: GUI and 3D reconstruction. From top left: Image with blobs and their mean depth, left and right video streams, 3D reconstruction. Bottom center: Navigation map and depth map.

Developing a framework for robot control and navigation requires many sub-systems to be devised, in order to reach quasi-independence of the software from the hardware. To achieve this objective, we based our work on already established software libraries: OpenCV[3], for image processing, basic GUI drawing (highgui module) and user I/O, Pthreads-Win32[4], for cross-platform (Windows and Linux) parallel threads management, Ogre[5], for 3D self-representation of the robot and reconstruction of the depth map, CUDA SDK[6] for GPU coding and OpenCVBlobsLib for labelling and filtering connected regions.

4 TESTING THE SYSTEM ON REAL GROUND

Firstly, 3D reconstruction was tested to check the accuracy of depth estimation: By placing a planar chessboard at a known distance we computed the mean depth over the object area and then compared it with the real one, obtaining the results seen in Fig. 6.

As for navigation, we performed some experiments in order to assess the actual capabilities of the robot in navigating in a room:

1. First experiment: Two waypoints were issued (Fig. 7 shows snapshots taken from a lateral camera). The robot reached the first one avoiding the frontal obstacle (this was achieved through the creation of intermediate waypoints by the robot itself) and then proceeded towards the second one, behaving as expected.

2. Second experiment: A single waypoint was issued, the robot found an obstacle immediately in front of itself and then two others laterally while reaching its target (Fig. 8).

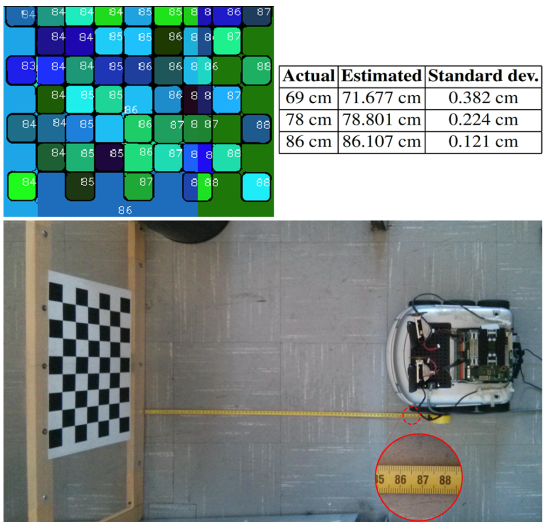
Videos of these tests and others can be found at the following link: http://goo.gl/V1hva8. Finally, in Fig. 9, execution times for a single processing cycle are shown.

[3]version 2.4.6, www.opencv.org

[4]ver. 2.9.0, https://sourceware.org/pthreads-win32/

[5]ver. 1.8.1, http://www.ogre3d.org/

[6]ver. 5.0, http://www.nvidia.com/object/cuda_home_new.html



Actual	Estimated	Standard dev.
69 cm	71.677 cm	0.382 cm
78 cm	78.801 cm	0.224 cm
86 cm	86.107 cm	0.121 cm

Figure 6: Experimental setup. On the top left, an extract of the blobs image with their mean depth. On the top right, a table comparing real (Actual) and computed (Estimated) distances, along with the standard deviation of depth over the chessboard area. The error on the estimated depth still allows for precise navigation.

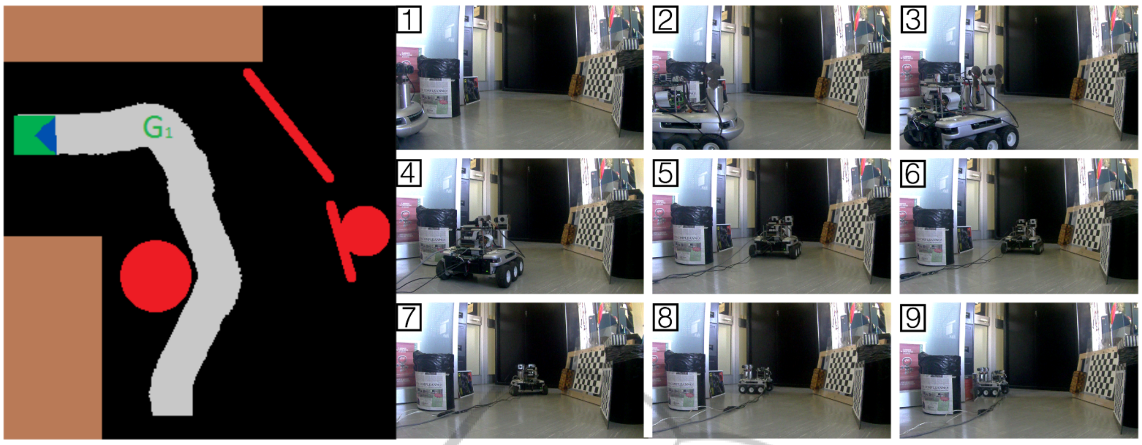


Figure 7: Left figure: Above view of the environment of the first navigation experiment. Three red obstacles and an intermediate goal G_1 are present, the brown objects representing other objects. The grey area represents the trajectory followed by the robot, as drawn by the framework itself. Right figure: Lateral view of the experiment.

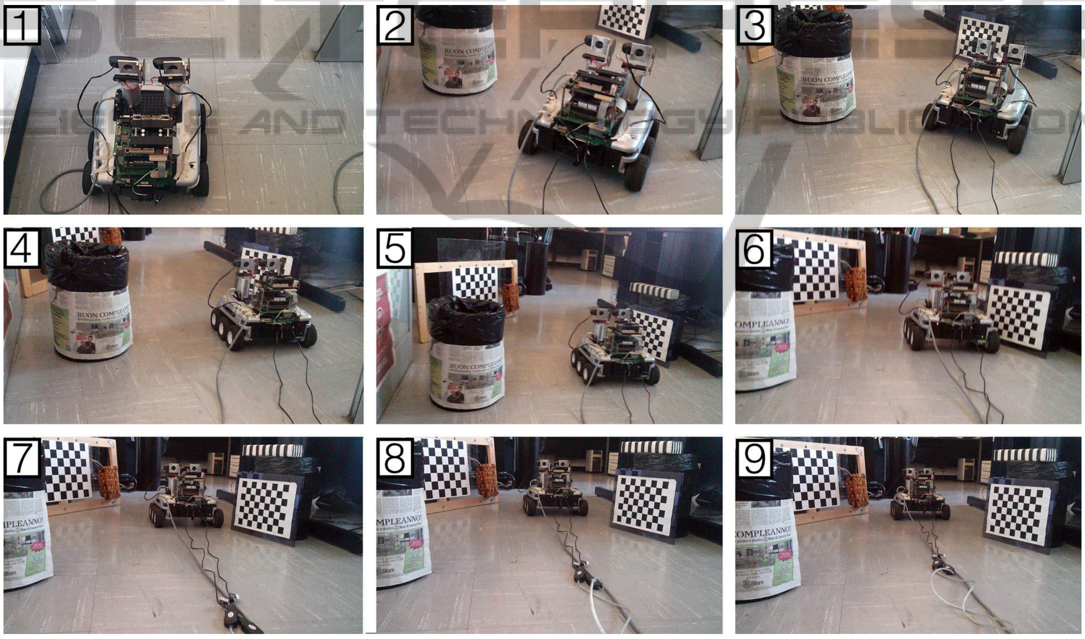


Figure 8: Second navigation experiment: back view. The robot found a frontal obstacle and two lateral ones. This time its goal was set to the position visible in frame 9.

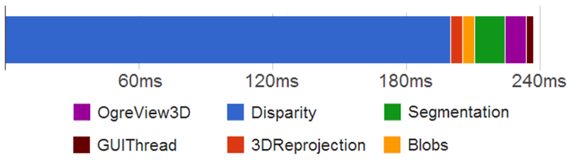


Figure 9: Execution times: note how disparity estimation is the heaviest computation involved. However, being run on the GPU, the CPU is free to process other tasks.

5 CONCLUSIONS AND FUTURE WORK

In this paper a modular stereo-vision based robotic system has been presented. It has been shown that even using only consumer-grade hardware, the system is able to reach real-time performance with a good responsiveness of the robot in the avoidance of obstacles. Also, the biologically inspired approach has provided good results when dealing with unrectified stereo rigs, providing robustness and reliability. Moreover, by integrating many features of the image with the state of the robot, the system is able to provide an estimate of depth of the pixels and of the 3D structure of the scene, along with a coarse segmentation of objects based on their colour. By design, every module of the framework can be enhanced, expanded and more modules can be added: for example, it could be possible to add information from structure from motion to enhance the robustness of the vision module. In the future we plan to add some new features such as object recognition and tracking, execution of tasks when reaching way-points, filtering and clustering of the 3D reconstruction data. Finally, the modularity of this work allows it to be used together with other robotic systems, the only requirement being the development of a new class modelling the robot. Thanks to its modularity, we think that other people could use this framework in their projects and we plan to release the source code in the near future.

REFERENCES

Adelson, E. H. and Bergen, J. R. (1991). The plenoptic function and the elements of early vision. *Computational models of visual processing*, 91(1):3–20.

Chang, F., Chen, C.-J., and Lu, C.-J. (2004). A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220.

Chen, S., Li, Y., and Kwok, N. M. (2011). Active vision in robotic systems: A survey of recent developments. *INT J ROBOT RES*, 30(11):1343–1377.

Chen, X. and Chen, H. (2010). A novel color edge detection algorithm in rgb color space. In *Signal Processing, 2010 IEEE 10th Int'l Conf. on*, pages 793–796. IEEE.

Chessa, M., Bianchi, V., Zampetti, M., Sabatini, S. P., and Solari, F. (2012). Real-time simulation of large-scale neural architectures for visual features computation based on gpu. *Network: Computation in Neural Systems*, 23(4):272–291.

Chessa, M., Solari, F., and Sabatini, S. P. (2009). A virtual reality simulator for active stereo vision systems. In *VISAPP (2)*, pages 444–449.

Dutta, S. and Chaudhuri, B. B. (2009). A color edge detection algorithm in rgb color space. In *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom'09. Int'l Conf. on*, pages 337–340. IEEE.

Fleet, D. J., Wagner, H., and Heeger, D. J. (1996). Neural encoding of binocular disparity: energy models, position shifts and phase shifts. *Vision research*, 36(12):1839–1857.

Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall.

Grigorescu, S. M., Macesanu, G., Cocias, T. T., Puiu, D., and Moldoveanu, F. (2011). Robust camera pose and scene structure analysis for service robotics. *Robotics and Autonomous Systems*, 59(11):899–909.

Klarquist, W. N. and Bovik, A. C. (1998). Fovea: A foveated vergent active stereo vision system for dynamic three-dimensional scene recovery. *Robotics and Automation, IEEE Transactions on*, 14(5):755–770.

Matveev, A. S., Hoy, M. C., and Savkin, A. V. (2013). The problem of boundary following by a unicycle-like robot with rigidly mounted sensors. *Robotics and Autonomous Systems*, 61(3):312–327.

Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proc. 2003 IEEE CS Conf. on*, volume 1, pages I–195. IEEE.