Raspberry PI Based Stereo Vision For Small Size ASVs

Ricardo Neves
Faculty of Engineering of the University of Porto
Portugal
Email: ricardojmsneves@gmail.com

Anibal C. Matos
INESC TEC
Faculty of Engineering of the University of Porto
Portugal
Email: anibal@fe.up.pt

***Abstract*—This paper presents an approach to stereovision applied to small water vehicles. By using a small low-cost computer and inexpensive off-the-shelf components, we were able to develop an autonomous driving system capable of following other vehicle and moving along paths delimited by coloured buoys. A pair of webcams was used and, with an ultrasound sensor, we were also able to implement a basic frontal obstacle avoidance system. With the help of the stereoscopic system, we inferred the position of specific objects that serve as references to the ASV guidance. The final system is capable of identifying and following targets in a distance of over 5 meters.**

This system was integrated with the framework already existent and shared by all the vehicles used in the OceanSys research group at INESC - DEEC/FEUP.

I. Introduction

Computer vision is one of the most demanding areas in the robotics field. The need for autonomy in water vehicles demands for onboard computational power. Typically, vehicles using image sensors as an aid in their manoeuvring capabilities are either equipped with powerful processing units to deal with the online image processing or they use more capable equipment in remote stations that receive and process the online data, thus limiting their area of action.

As vehicles tend to accomplish larger and more complex missions, energetic autonomy poses a problem to the use of powerful computational systems; on the other hand, the cost of special-purpose hardware, though having dropped over the years, is still a limitation to the dissemination of robotic applications. Recent years have brought us a range of ARM architecture computational devices such as the Raspberry PI or the even more powerful Quad-Core ODROID-U2 , devices under USD 90, allowing the off-the-shelf robotics era to begin. Systems like the one described in [4] use computer vision to detect the horizon line and specific objects in the scene as an aid to a small sailboat guidance. Others [5] [6] use visual information to classify terrain and distinguish the water areas. Some applications have also been developed being capable of avoiding obstacles in specific water and scenery conditions using stereovision [7].

II. System Description

Our system was installed on the Gama catamaran (figure 1). This ASV is 1.5m long and is equiped with a PC-104 stack, WiFi link and multiple sensors, being propelled by two thrusters. A second computational unit, a Raspberry PI, was used to deal with image processing tasks.

The Model B Raspberry PI is a 3.5W, USD 35 computer with a 700 MHz ARM1176JZF-S processor and multiple I/O interfaces. The two webcams mounted in this assembly use the two available USB ports and the ultrasonic sensor is connected to the GPIO pins (figure 2).
The Raspberry PI is powered by the ASV power supply and the two computational units keep a continuous communication thru an ethernet connection.
Both computational units run Linux distributions and the algorithms running on the PI use OpenCV libraries, mainly cvBlobsLib functions, and rely on the Video4Linux2 API to deal with the parallel image acquisition.

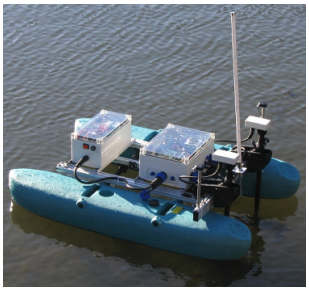

Fig. 1. OceanSys Gama catamaran

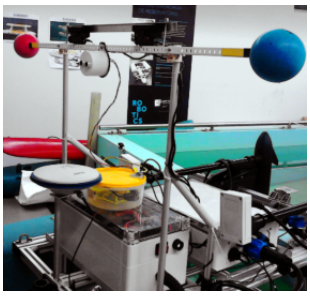

Fig. 2. Detail of the camera pair and ultrasonic sensor

III. Stereo Vision

Color and shape are great cues for identifying objects of interest. By using monocular vision and very simple algorithms, one can easily infer the orientation of a certain point with relation to the camera reference system. On the other hand, a single camera doesn't supply us with the depth information of that same point. This happens because all the points on the same depth ray will be represented in a single point on the camera image plane.
That problem is solved by using more than one camera. A second camera, seeing the same scene from another perspective will have the same point represented in a different location of its image plane (figure 3).

For one to be able to infer the 3D coordinates of a given point, the geometry of this stereo pair must be known and remain unchanged for the duration of the image pairs acquisition. Also, since image sensors aren't precisely placed

978-0-933957-40-4 ©2013 MTS

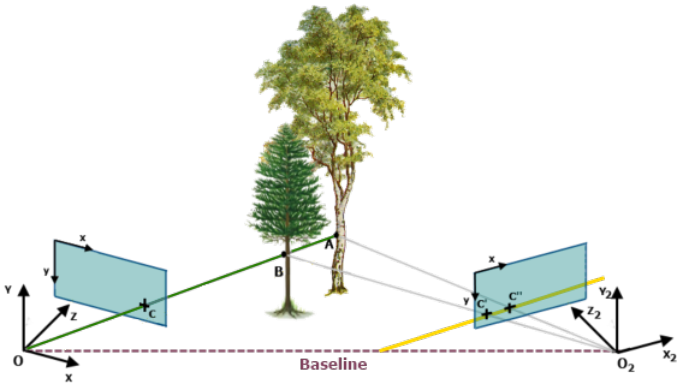

Fig. 3. Stereo Rig

in the camera and lenses induce distortions in the image, there is the need for camera calibration. This is even more important when we deal with low-price components, as we did here.

The calibration procedure aims to discover the intrinsic parameters of each camera and the extrinsic parameters of the stereo rig. The intrinsic parameters are the ones that condense the geometrical and optical specificities of the camera (lens distortion coeficients, focal distance, principal point) while the extrinsic parameters are the ones that relate the unknown reference frame of the camera to a known one. The most frequent way of running this procedure is using each camera to collect images of a chessboard of known dimensions seen from different perspectives. In stereo calibration, the chessboard must, in each image pair acquisition, be seen by both cameras so that the extrinsic parameters can be found; alternatively, if the used rig has a wider baseline (i.e. distance between optical centres) a good calibration can only be achieved by first calibrating each camera and then, calibrating the camera pair.

Finding the cameras parameters will permit that the image of each camera is first undistorted and then the pair rectification. The OpenCV library has specific functions to deal with each of these operations. The end result should be a distortion free image pair that is row-aligned, making the search for matching objects in both frames an unidimensional search (figure 4). This avoids both the higher computational cost of searching entire images and the probability of erroneously matching objects in left and right image. The whole calibration procedure is described in detail in [1]

IV. Parallel Image Capture

For a given pair of images, stereoscopy demands that the left and right image acquisition on a moving scene is made at the same time. Though some methods like in [2] are able to function without fulfilling the previous rule, ignoring this restriction will induce errors in the measuring process and invalidate the epipolar constraint (i.e. unidimensional search).

A. Hardware Synchronization

This assembly used two webcams. To achieve hardware synchronization, the chosen webcam model had to have an image sensor that allowed synchronization.

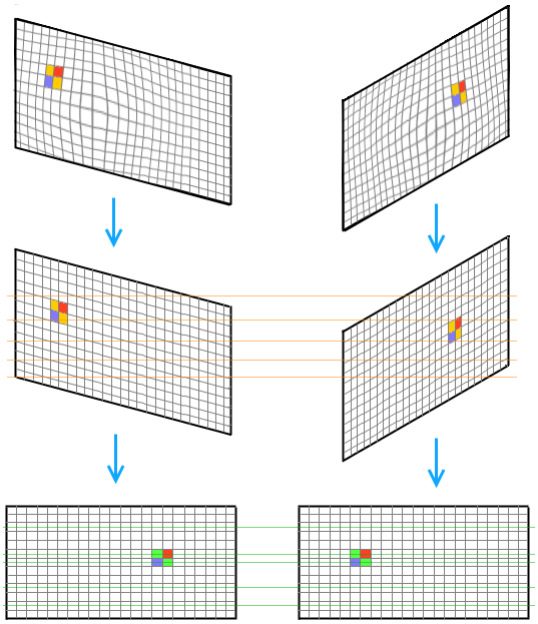

Fig. 4. Calibration sequence

Very often, cameras used for industrial or scientific purposes include a trigger input allowing that same synchronization.Also available on the market, stereo cameras with a fixed baseline can be purchased and already solve that problem. As the original idea was to build a low cost prototype, we've used two PS3 Eye Cam units. This model, very used in image processing by hobbyists, has an Omnivision sensor, OV7720, which has a Frame Sync input. Looking at the diagram in figure 5, one sees that the vertical synchronism signal VSYNC is responsible for the start of the frame acquisition. By making a simple intervention in both cameras, we've used the VSYNC signal of one camera to control the FSIN sensor input of the other [3]. This way, we were able to get a stereo rig able to acquire image at 120 fps for USD 30.

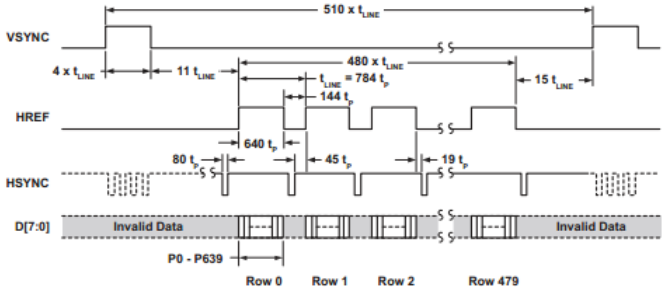

Fig. 5. OV7720 timing diagram for a VGA frame acquisition *(OmniVision)*

B. Image Acquisition

Having the hardware synchronized isn't enough to receive the images at the same time. As the image processing algorithm uses OpenCV libraries, the first attempt for image acquisition was using its image capture functions. Though they perform well for monocular vision applications, they deal uneffectively with the frame pair acquisition. The temporal difference between frames in the frame pair isn't guaranteed

to be limited and is frequently too high to guarantee good results. This forced us to seek for an alternative and we've ended up using Video4Linux2 for the frame acquisition.
Video4Linux2 is an API developed by the Linux community with the goal of unifying the access to video devices. Nowadays, a wide variety of devices including the majority of webcams is compatible with this API. By using it, one can interact with all kinds of camera controls and deal with the buffer content, having a more direct control of the data flow. Using a video capture example available in the API website, we modified it so that processing of the buffer content of the two cameras was made at the same time. The buffer content is in the YUV422 format. This is a format that uses reduced chrominance information and encodes two RGB pixel information - 6 bytes - using just 4 bytes. The buffer content of both cameras is supplied to a conversion function that uses the following known relation to turn the two contents into two RGB frames:

$$\begin{cases} R_1 = 1.164\,(Y_1 - 16) + 1.596\,(V - 128) \\ R_2 = 1.164\,(Y_2 - 16) + 1.596\,(V - 128) \\ G_1 = 1.164\,(Y_1 - 16) - 0.813\,(V - 128) - 0.391\,(U - 128) \\ G_2 = 1.164\,(Y_2 - 16) - 0.813\,(V - 128) - 0.391\,(U - 128) \\ B_1 = 1.164\,(Y_1 - 16) + 2.018\,(U - 128) \\ B_2 = 1.164\,(Y_2 - 16) + 2.018\,(U - 128) \end{cases}$$

V. Algorithm

A. Interconnections and Functionalities

The program developed for the target and buoy identification was written in C++ and a global working structure is presented in figure 6. The main goal of the developed algorithm is to detect specific coloured shapes and use that information to provide references that determine the vehicle's motion. In the case of targets that the vehicle must follow, they are of known color combinations and orientation. The algorithm also provides the vehicle board control software with the position and orientation of left and right buoys that delimit routes the ASV must follow.

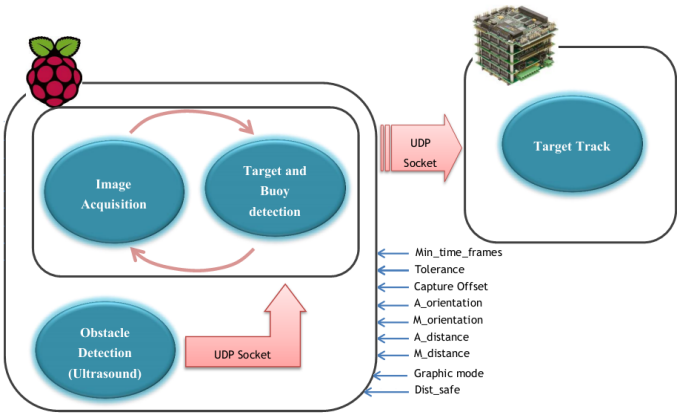

Fig. 6. Software modules

- Min_time_frames- minimum time between frame acquisition (in ms)
- $Tolerance$ - value of the vertical tolerance so that two points are considered to be respecting the epipolar constraint (in pixels)
- $CaptureOffset$ - Frame Height = Capture Resolution Height - Capture Offset. (in pixels)
- $A\_orientation$ and $M\_orientation$ - Additive and multiplicative coefficients to correct possible calibration imprecisions effect on orientation
- $A\_distance$ and $M\_distance$ - Additive and multiplicative coefficients to correct possible calibration imprecisions effect on distance
- $Graphic\_mode$ - Enables graphic mode (bool)
- $Dist\_safe$ - Safety distance information for the ultrasonic sensor (in m)

B. Description

The object identification function uses the cvBlobsLib, an OpenCV library that allows the identification of 8-connected components - blobs - in binary images. It provides functions for filtering undesired objects and extracting blob characteristics such as area, orientation, roughness and axis ratio among others.

On a typical execution of the program, we use a capture offset of 120 for a 320x240 pixel capture resolution; this is done because the usual position of the horizon line is near the middle of the image.

After the image capture process is completed, the algorithm converts the RGB frames to HSV color space. This is a standard procedure because the HSV color space is more robust to light variations. The algorithm is designed to deal with three types of objects: target, right buoy and left buoy. For the data exchange from the Raspberry PI to the ASV PC-104 computer, a structure with three variables for each object type is initialized. Each of the object types is associated with a flag, to determine its existence, an orientation and a distance. When in operation, this structure's content is continuously sent to the *target track* module.

The target identified by the program is bicolor. We used a rectangle of 50x30 cm like the one in figure 7 that must be attached to the back of the vehicle we wish to follow.

Fig. 7. Target

The buoys are 60 cm coloured spheres. The need for big a target and buoy has to do with the low resolution used. First tests using small size buoys greatly limited the working distances.

As we have limited computational power, the idea of using more dense stereoscopy matching methods like Block Matching or Graph Cut algorithms was not feasible. These methods demand for a full remapping of the two images, that is, undistorting and rectifying each of the image pixels and then performing an intensive comparison between image areas.
The problem complexity was reduced and, by using the cvBlobsLib, we basically used the centre of gravity of each blob to decide about its inclusion in the group of the objects of interest and decide about the best matches for each of them.

For a given object in the left image, with center coordinates (x_{left},y_{left}) and its match in the right image, with center coordinates (x_{right},y_{right}), its 3D coordinates are found as follows:

$$\begin{cases} d = x_{right} - x_{left} \\ X = \frac{T_x(x_{left} - c_{x_{left}})}{-d + (c_{x_{left}} - c_{x_{right}})} \\ Y = \frac{T_x(y_{left} - c_{y_{left}})}{-d + (c_{x_{left}} - c_{x_{right}})} \\ Z = \frac{T_x f}{-d + (c_{x_{left}} - c_{x_{right}})} \end{cases}$$

where d is the disparity value, (c_x,c_y) the principal point coordinates for left and right cameras and T_x is the translation along the x-axis of the right image with relation to the left one. Each of these values is obtained after the calibration procedure.

For the correction an object center coordinates, our algorithm relies on maps that result from a conversion of the ones generated by the OpenCV function *cvInitUndistortRectifyMap()*. This function generates four xml maps, two per camera, for the *cvRemap()* function, whose goal is to generate a new image without distortions. For a given point with coordinates [i,j] in the remapped image, the map mx1.xml tells the *cvRemap()* function the x coordinate of the point of the original image it must copy for the [i,j] location. my1.xml does the same for the y coordinate. This may be useful to remap a whole image but it's not if you wish to have a single point corrected. Our function reorganized these maps in a way that when we need to remap a single point of coordinates [i,j], we just have to access mRx1.xml and mRy1.xml [i,j] cell and to get the corrected x and y coordinates of that same point.
The determination of the 3D coordinates evidently happens after we have corrected the original center coordinates.

In figure 8 is described the sequence of operations for target identification. The buoy detection sequence is a simplified version of this one as in that case we're dealing with only one color. Figure 8 a) shows the aspect of a pair of images acquired by the system. The algorithm starts by binarizing the two frames, separating them by first and second color of the target (figures 8 b) and c)). Each of these binarized images is filtered taking in consideration the area and the axis ratio of each object. If the algorithm doesn't find evidences of the two colors in each of the frames of the pair, it returns no detection; if it finds them, it will register the corrected coordinates of each object's centre and it's area for future matching validation. Figure 8 d) shows the process of finding, individually for left and right frame, the correspondence between first and second color objects. For the object to be a candidate target, it must (a) respect ΔY tolerance, because the ASV oscillates while moving and the center of first and second color might not be vertically alligned, (b) ΔX, or the x coordinate distance between the centres of the first and second color object must be within certain limits and (c) X coordinate of the second color must always be greater than the one of the first color. Every object that passes this test will be included in a candidate list for stereoscopic evaluation. For every candidate identified on the left image, a match is tried with all the candidates found on the right image. This evaluation takes into consideration the respect for the defined Tolerance (epipolar constraint tolerance) and the fact that an object appearing on the left image always has a greater X coordinate than that same object on the right image. After all comparisons of left and right candidates, the system elects the closest target from the set of detected targets and writes it's orientation and distance in the respective structure variables.
The detection function can be called one, two or three times, according with the objects we wish to detect. After each of these executions, the last value supplied by the ultrasonic sensor is evaluated and compared with the safety distance. If an obstacle is detected, the content of the structure is changed so that the *target track* application on the ASV side can react to it.

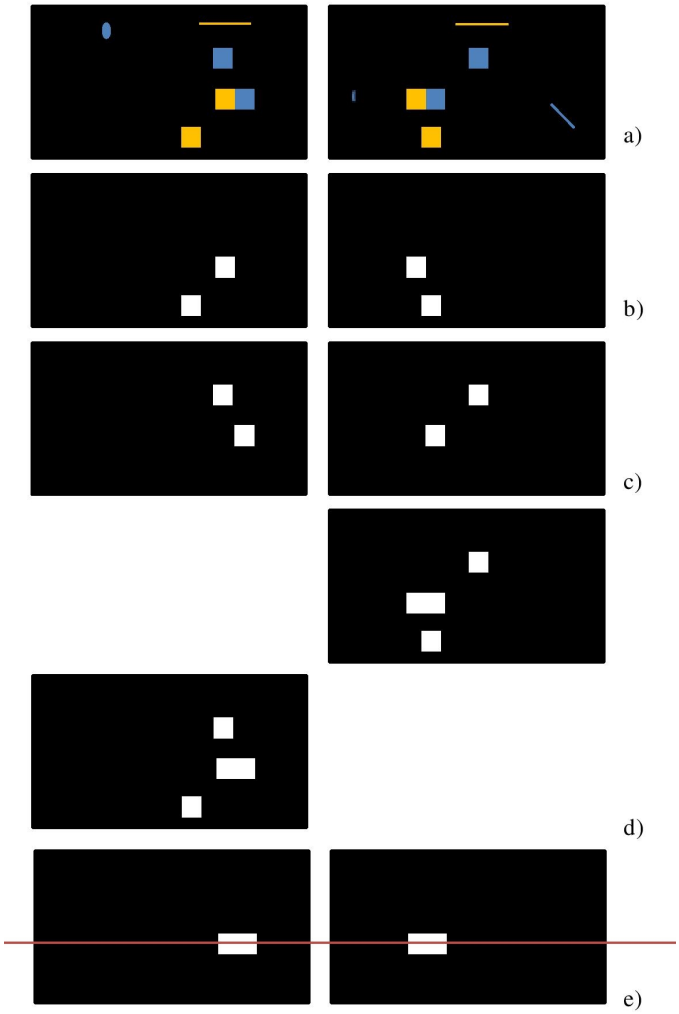

Fig. 8. Target detection sequence

VI. Integration with On Board Control Software

The on-board control software of Gama ASV is composed by a set of modules that communicate with each other by a message passing mechanism [8]. These modules are organized in an hierarchical fashion with the lowest level providing a hardware abstraction layer. The interaction with the control software is performed at a higher abstraction level, by defining one of a set of possible vehicle behaviours and providing in real time, the required inputs for the active control loops. The most relevant high level behaviours are

- independent control of yaw and speed
- line tracking manoeuvre
- station keeping manoeuvres

In order to make the ASV Gama able to track a possibly moving target while keeping itself inside a pre-defined navigation lane, the most adequate behaviour is independent control of yaw and speed. For this behaviour the yaw and speed references are computed in real time from the output of the artificial vision system. This output consists on the definition of the relative orientation and distance to the following objects:

- target
- closest right lane mark
- closest left lane mark

Whenever the target is not detected by the vision system, the speed reference is set to a predefined default value and the yaw reference is set to the middle of the navigation lane (or to the current yaw if any of the lane markers is missing). Whenever a target is detected (inside the navigation lane) the yaw reference is set to the absolute orientation to the target. In this case, the absolute target speed is estimated by a numerical differentiation mechanism, whose output, after passing through a low pass filter is set as the reference input to the ASV speed control loop.

VII. Preliminary Results

Although expecting that by now we'd have outdoor tests, up to this moment they haven't been possible. For that reason, all testing occurred in the laboratory.
An image sequence was acquired in a riverine location, place where the system is planned to work on. We tested the possibility of using Local Variance Filter for terrain classification. In figure 9 it's possible to see the result of the application of a 3x3 and a 5x5 window to a typical working scenario. These filters have proven to be more robust to area classification than typical edge detection filters, normally very sensitive to small details. They are computationally light and seem very promising as a reinforcement to stereoscopic data.

For the determination of the threshold values for the object of interest, a Matlab script was used. A series of images of the object was acquired using the cameras under different light conditions. The script returns the color histograms of each image (figure 10) and establishes the threshold limits, ignoring spurious peaks.

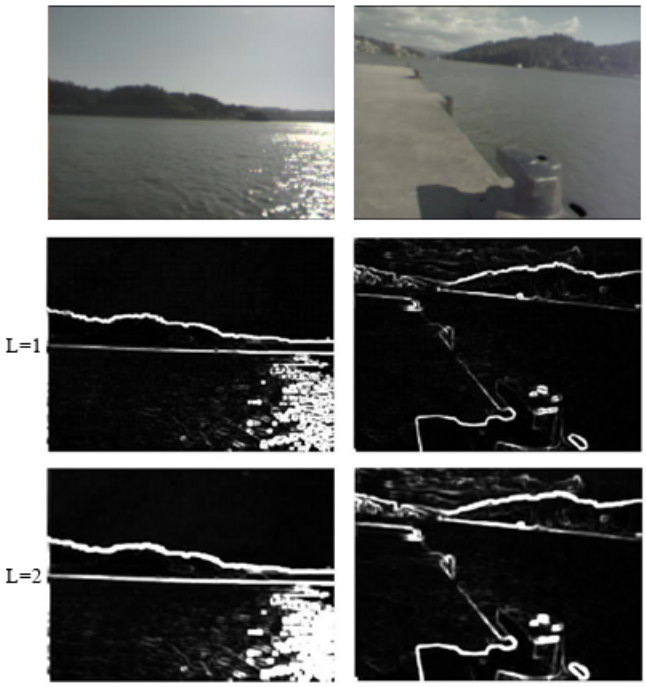

Fig. 9. Local Variance Filter results

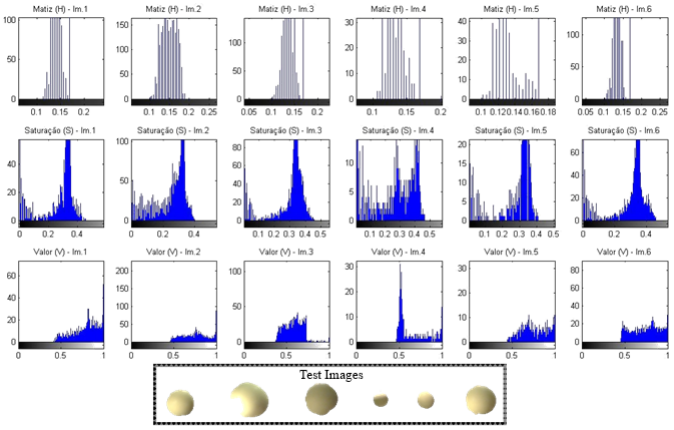

Fig. 10. Histogram script for threshold determination

These values are submitted as a parameter to the target and buoy detection function.
In the tests conducted inside the laboratory, the camera pair used a baseline of 30 cm and its horizontal field of view covered approximately 45.
For the orientation test, a target was placed at around 5.5m of the ASV as the boat rotated. Results are shown in figure 11. *(Xe,Ye)* and *(Xd,Yd)* are, respectively, left and right coordinates, in pixels, of the detected target centre, *D* is the measured Distance, in meters and *Ori* is the orientation, in radians. In this test, orientation is varied from negative to positive values and the distance result is stable around real target distance.
Consistent differences in the vertical coordinates for the left and right image representation of the object justify the need of the Tolerance parameter in the algorithm. These are not only due to calibration errors but also for the differences of the object appearance in each camera sensor, which affect it's

centre coordinates.

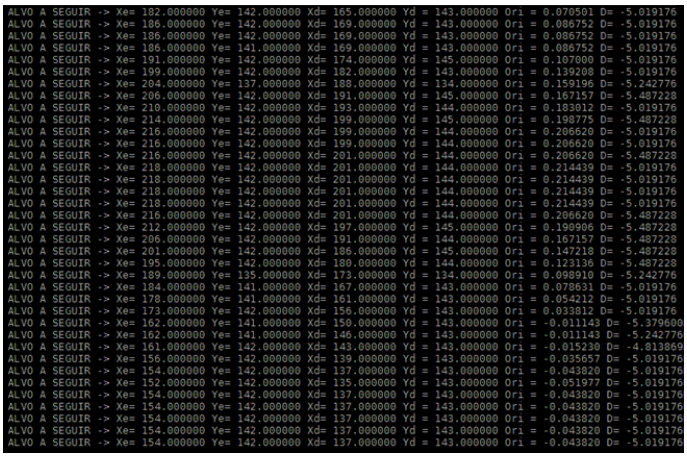

Fig. 11. Orientation test log

Tests with varying distances were also made. Figure 12 shows the results of those tests. Occasionally, as seen in the figure, the target goes undetected but the system rapidly finds it again. By following the log values, the inverse relationship between disparity *(Xd - Xe)* and depth can be confirmed.

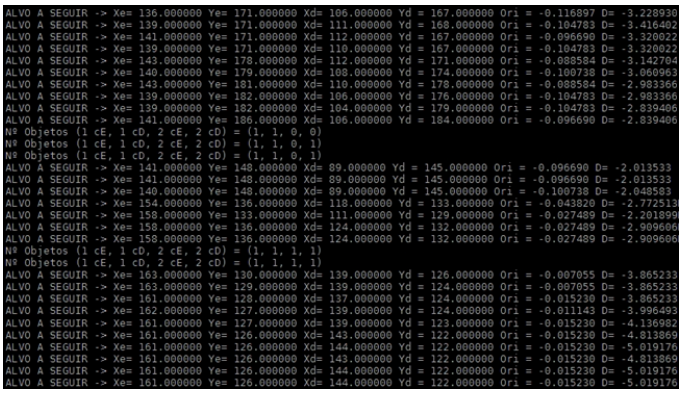

Fig. 12. Distance test log

We hope to be able to test this system soon in it's working environment.

VIII. Conclusion and Future Work

The system we've developed is able to accomplish the function it's designed for for under USD 70.

With this work, it's been proven that it there is the possibility of performing stereoscopic image processing using low cost computational units. Results of 2-3 fps were proven attainable. Although using more dense matching algorithms is still a difficult task to these small units, using simpler techniques envolving binary imaging and criteriously chosen 3D information is a good way of surpassing those limitations.

The possibility of combining stereoscopic data with the local variance filter results seems a promising way of more accurately classifying terrain, in particular water plane classification, and specific objects reducing the possibility of false matches. In the specific case of buoy detection, redundancy achieved by simultaneous application of the Circular Hough Transform to the elected candidate will greatly reinforce the certainty of the detection. The inclusion of ultrasound sensors is a computationally and financially inexpensive way of rapidly detecting obstacles; in the future, surrounding the ASV with several of these will create a cheap near field obstacle avoidance system.

Several of the suggested improvements will only be possible with the replacement of the computational unit by a more powerful one, like the one suggested in the beginning of this text.

Acknowledgment

The authors would like to thank the Faculty of Engineering of the University of Porto, specifically to the Department of Electrical and Computers Engineering and to the OceanSys team, whose help and advise have been of great use to complete this work.

References

[1] G. Bradski and A. Kaehler, *Learning OpenCV*, O'Reilly Media Inc., 2008.

[2] M. Svedman, L. Goncalves, N. Karlsson, M. Munich and P. Pirjanian, *Structure from stereo vision using unsynchronized cameras for simultaneous localization and mapping*, Intelligent Robots and Systems, 2005.

[3] M. Koval,*Vision-based autonomous ground vehicle navigation*,2011

[4] T. Neumann e A. Schlaefer, *Feasibility of Basic Visual Navigation for Small Robotic Sailboats*. Sauz, Colin and Finnis, James, editors , Robotic Sailing 2012, pages 1322. Springer Berlin Heidelberg, 2013.

[5] L. Matthies, P. Bellutta and M. Mchenry, *Detecting Water Hazards for Autonomous Off-Road Navigation*,Proceedings of SPIE Conference 5083: Unmanned Ground Vehicle Technology V, pages 263-352, 2003.

[6] A. Rankin and L. Matthies, *Daytime Water Detection Based on Color Variation*, 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 215-221, 2010.

[7] J. Wang, P. Huang, C. Chen,W. Gu and J. Chu,*Stereovision Aided Navigation of an Autonomous Surface Vehicle*, 2011 3rd International Conference on Advanced Computer Control (ICACC), pages 130-133, 2011.

[8] N. Cruz and A. Matos, *The MARES AUV, a Modular Autonomous Robot for Environment Sampling*, Proceedings of the MTS-IEEE Conference Oceans'2008, Quebec, Canada, September 2008.