ID15- iObserver: Species recognition via computer vision

Fernando Martín-Rodríguez[41], Mónica Barral-Martínez[42], Ángel Besteiro-Fernández[20], José Antonio Vilán-Vilán[21]

Abstract- This paper is about the design of an automated computer vision system that is able to recognize the species of fish individuals that are classified into a fishing vessel and produces a report file with that information. This system is called iObserver and it is a part of project Life-iSEAS (Life program).A very first version of the system has been tested at the oceanographic vessel "Miguel Oliver". At the time of writing a more advanced prototype is being tested onboard other oceanographic vessel: "Vizconde de Eza". We will describe the hardware design and the algorithms used by the computer vision software.

Keywords: fishing boat, oceanographic vessel, computer vision, object recognition, C++/OpenCV.

I. INTRODUCTION

As stated in the abstract, iObserver is part of a larger project called iSEAS (http://lifeiseas.eu/el-proyecto/). iSEAS project is about appropriate management and reduction of discarded fish. iSEAS is funded by Life program. iObserver is part of iSEAS and tries to develop and test various technologies that can be installed on-board fishing (or oceanographic) vessels in order to collect useful data from fishing campaigns.
Particularly, this paper is about the part of iObserver that obtains a report on the number and species of captured fish, both useful captures and discards. For us, iObserver will be a computer vision system, installed over a conveyor belt inside a fishing or oceanographic vessel. This space is called the "processing room" and it is used to manually classify fish that is afterwards put into stowage facilities or discarded and thrown again to sea.
Nowadays, European authorities want to reduce (and even eliminate) discards as they can cause problems to ecosystems. iObserver is seen as a help in this process because it will be able to measure the amount of each species that enters the vessel (useful or not).
iObserver consists of a industrial (computer vision) camera and a processing unit (industrial PC) equipped with our self-developed computer vision software. Normally, it will be placed over the conveyor belt at its same beginning. Software is designed to need almost no user interaction. User only starts the capturing process and stops it when fish classification work has ended. At end of each capturing burst, system generates a report file (ascii-csv format) containing recognition results.

II. HARDWARE DESIGN

The first prototype installed in the "Miguel Oliver" ship in March 2015, was designed trying to hold camera and lighting in the same box. This idea yielded an enormous, heavy and not very practical design. A second, more practical, prototype was designed for the second campaign (travel of "Vizconde de Eza" vessel to NAFO fish area). This new prototype is described below.

A. Main Box

Main system box is a steel waterproof box that contains the key elements of the system: processing unit (industrial PC), industrial camera, touchscreen and an auxiliary system to avoid water condensation (based on a peltier cell).
We have tested two different cameras: Basler ACE acA2040-25gc (GigEthernet interface) [1] and JAI GO- 5000C (USB3.0) [2]. Both of them have a big image sensor: 1" and C-mount for optics, resolution is above 5 Mpixels in both cases. Both are possible elections. Quality of optics and lighting is more important than camera for this system. Although we have performed tests with fixed focal length lenses, varifocal lenses are more practical here because belt width can be very variable from one ship to another (and also are environmental conditions like roof height that can make us to install camera nearer or further).
Industrial PC is a fanless model with solid state disk. The first prototypes are also equipped with an external disk to save captured images (to save disk space the system automatically discards void and/or repeated images, repeated images are frequent because of moments when belt is stopped). These images will be used later in refining the computer vision software.
Nowadays it is still pending the design of an auxiliary system that can measure belt speed generating a synchronization signal for capturing images at constant belt advances (just now, captures are launched at constant time intervals).

B. Lighting

Lighting is designed to provide a constant soft light rectangle on the belt. For that issue, we use a pair of LED strip lights at both sides of the belt (figure 1). Height and angle of these lights can be modified to minimize shadows and light variations.
Lights are embedded in steel waterproof boxes and are electrically fed from a power source inside the main box.

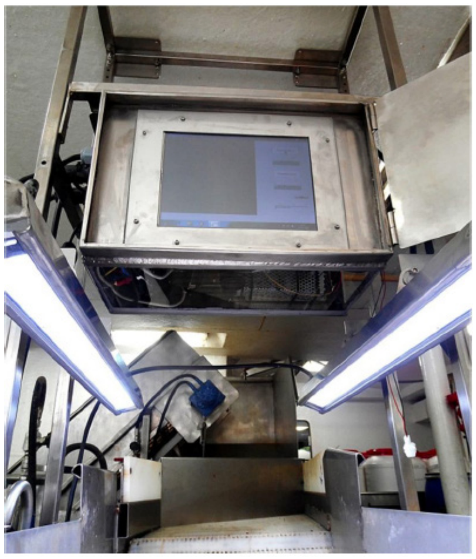

Fig. 1. Main Box and external lighting as they were mounted in the oceanographic vessel: Vizconde de Eza.

III. COMPUTER VISION SOFTWARE

Computer vision software was designed starting from test images captured at IEO (Instituto Español de Oceanografía: http://www.vi.ieo.es/) where we installed a conveyor belt and a camera. Software development is performed in two parallel stages: algorithms are tested and simulated (prototyped) using atlab [3], definite methods are then implemented using C++ [4] and OpenCV [5].

A. Initial Calibration

Calibration is mainly about finding out color characteristics of belt (we can find blue belts, but with different hues and also white ones). We also want to determine the expectable maximum and minimum brightness values to perform color enhancing (basically white balance). Other important issue in calibration is stablishing a relationship between pixel coordinates and real distances (millimeter per pixel in our images).
See that lighting conditions and camera position will be always constant, so that this process may be performed only one time (offline, with an auxiliary software

application) and we can rely on user's manual help.
To calibrate, basically, user must put some object on the belt and run the calibration application. We have designed a calibration card (figure 2) that can be easily printed any time (the higher printing quality the better calibration).

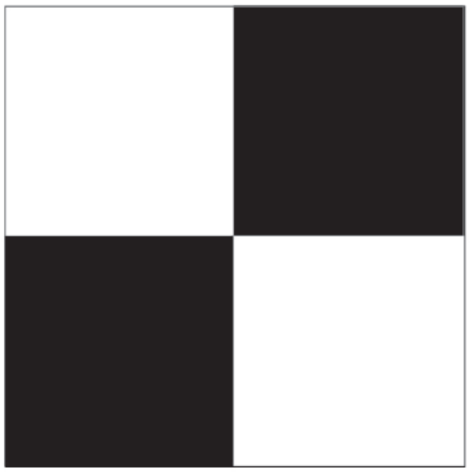

Fig. 2. Calibration card.

Calibration card has absolute white (brightness=1) and absolute black (brightness=0) inside so that calibration program can find out them. In fact, our application searches for the value that only has 1% of image pixels above (in brightness) ➘ white and the value that has 1% pixels below ➘ black.
After we know these two values, we can stretch color histograms [6] to lie always between both values, this is the color enhancement process that we will apply to all captured images.
After that, user is required to manually select background areas so that the application can compute the interesting parameters: mean values of R, G and B and mean value of Hue (the H of the HSV color space [7]). User selects a rectangle on captured image. As background portion can be small if photographing the calibration card, user can remove it and capture an additional "background only" image. This process can be repeated so many times as convenient.
Size calibration (millimeter per pixel computation) is also done using the calibration card. Resolution of the TIFF file (CalibrationCard.tif) is fitted to get exactly a 10 inches card (254 mm), obviously each of the four squares has 5 inches per side (127 m).

B. Conveyor Belt Removing
To remove background we use the mean values computed by the calibration application. Now we are in the main application and calibration values have been saved in a configuration file.
Other configuration file value that we must establish first is ROI. ROI in computer vision means "Region of Interest" and it is the part of the image where we are going to work. In this case ROI is constant because camera position also is. In figure 3 (from tests at IEO), we see that we capture more than the desired belt and its contents. We define a central ROI to avoid dealing with irrelevant and possibly problematic objects.

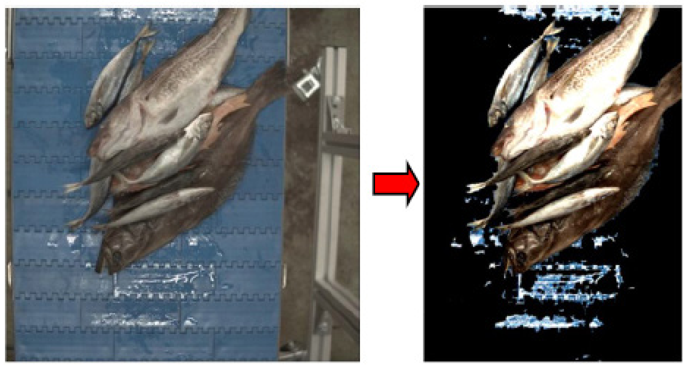

Fig. 3. Image background (conveyor belt) removing.

To remove background, application computes four masks. A mask is a selection image that has zero value for nonselected points (background) and value one for selected "active" points. Masks are computed by mean Hue, mean R, mean G and mean B. A tolerance of 0.1 is used (in a scale from 0 to 1, a point is considered to be background if its value is between "mean - tolerance" and "mean+tolerance"). Masks are combined with this equation:

$$M = [M_H]^{c_H}[M_R]^{c_R}[M_G]^{c_G}[M_B]^{c_B}$$

So basically masks are multiplied but using some coefficients (ci's) that may be zero or one (activating or deactivating that criterion). This is so because we did not find a unique formula that could fit all the conveyor belt color/fish combinations. User must see the masks and decide which the best segmentations are, assigning to the corresponding (ci's) the active value. This is the last "manual calibration" that the system requires. Software can be optimized so that masks with a zero coefficient don't need to be computed.
Resulting mask M is refined using mathematical morphology operations [8] that are useful to remove salt and pepper noise (black points in white background or vice versa).
Background removing is also used to compute the percentage of active (non-background pixels) permitting to remove empty images. Previously, capture function has removed images that are equal to the previous one (previous image, empty or not is stored in memory for this task). These two simple operations allow avoiding many unnecessary computations.

C. Object Segmentation
In this problem, object segmentation (Id EST: distinguishing between different overlapping fishes) is very difficult. Traditional methods based on derivatives or other kind of gradient operators do not work well. We have the help of having detected the background but still any derivative is likely fooled by the local variations in fish skin (squama, dots…).
We have implemented an approximate method based on gray regions and in the watershed transform.
Method (explanation of stages):
- Compute the corresponding grayscale image (we tested also with the "value" or V component of HSV space).
- Apply two morphological operations [8]: first, open by reconstruction; second closing by reconstruction. This, basically, will erase strong local variations.
- Search for local peaks in gray level histogram. A peak is defined as a value higher than three times the mean histogram value. This will detect possible different objects.
Applying region growing [8] to the selected objects: this will make greater these objects and will possibly merge similar and near ones.
- Compute a gradient operator for the original graylevelimage. Add to the detected gradient points, those belonging to the boundaries of background detection (frontiers between black and object in figure 3, right).
- Using the detected objects as seeds and the complete gradient of above, apply the watershed algorithm [6,8].
In figure 4, we can see the segmentation result for image of figure 3. We can see that it is not exact but it is mainly correct giving us big portions of fish to be classified.

D. Species Classification
As we have just seen in the previous section, objects (fish specimens) will overlap and will occlude each other, making almost impossible using contour or morphology descriptors. For this reason we have centered in using "skin" descriptors: color and texture.
For a proper classification, user has to manage a "Fish Alphabet". To add a species to the alphabet, user has to put specimens on the conveyor belt and use an "Add Fish Sample" application. The more samples we have, the better this species will be represented in the alphabet.
The "add-sample" application only removes background; it does not execute the object segmentation, assuming all active points come from true skin of the declared species. User is responsible not to fool the application.
Once background pixels are removed, some characteristics about color and texture must be computed. At the time of writing, we only compute a 2D (10x10 intervals) color histogram of the "a" & "b" color components (from Lab color space [7]).

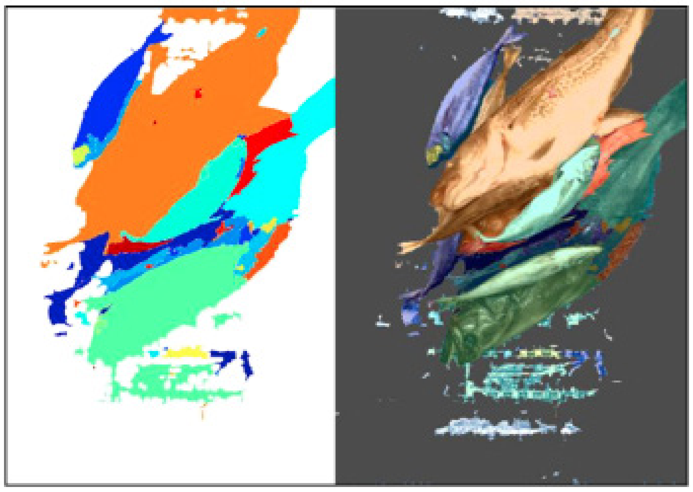

Fig. 4. Object segmentation: label image (left), combined representation (right).

At recognition time, the same characteristics (again only color histogram) are computed from the segmented objects. Then these values (in fact, matrixes) are compared through Euclidean distance with the alphabet yielding a recognized species.

Our system also computes a self-confidence value for each segmented (and recognized) object. Self-confidence tries to determine how sure an automatic recognizer is of its own decision. It is computed using the minimum distance value (the lesser, the more confident), the relation between minimum distance and second best (the greater, the better) and object area (very small objects are typically segmentation errors and are not well classified). Combining all these values empirically we get a number between 0 and 100 (intuitively like a percentage) that measures recognizer self-confidence. Empirically we deduced that confidences less than 10 are not reliable and these recognitions are removed from final report.

In figure 5, we see two recognition examples where recognized objects are converted into color blobs. Color is assigned according to the recognized species. Low confidence blobs are removed. In the upper example, there are no errors: "Atlantic halibut" (Hippoglossus hippoglossus) gets red, species 2 gets green and "mackerel" (Trachurus trachurus) goes yellow. The image below is not so well recognized, the big fish above should get blue and it is misclassified as a mackerel.

Recognition stage still needs improving, adding texture descriptors (like, perhaps, Gabor filters [9]) to the extracted feature vector (nowadays, this vector is only the "ab" matrix histogram). We must also test other classification methods because only Euclidean distance is very basic.

IV. CONCLUSIONS AND FUTURE LINES

We have designed a system for solving the problem of species classification. Development is still incomplete although we are already getting promising results. Important conclusions are: the difficulty of overlapping objects segmentation, solved only approximately and the need to implement a "fish skin" recognizer that works based on color and texture. Perhaps objects get broken or merged in segmentation but the species percentage can still be computed correctly.

Main future lines are:

- Adding a speed sensor to the conveyor belt to fire an image when belt has moved a predefined length.
- Adding new, texture, features to the recognition stage.
- Testing new classification schemes that could be less prone to errors.

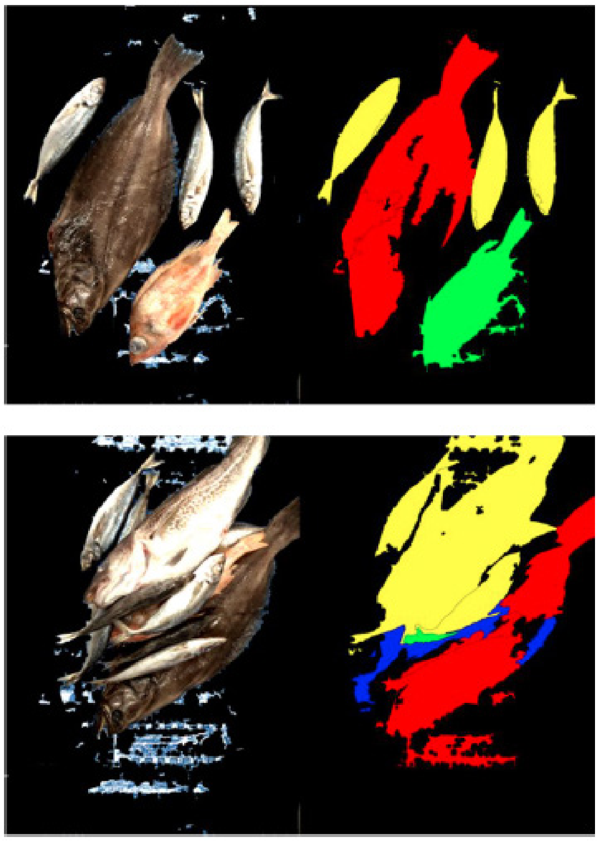

Fig. 5. Classification.

ACKNOWLEDGEMENT

Authors wish to thank European Life program for funding. Also, for their cooperation, we thank all staff in the IEO (Instituto Español de Oceanografía, http://www.vi.ieo.es/), all staff in CSIC-IIM (Instituto de Investigaciones Marinas, http://www.iim.csic.es/) and also the crews of vessels "Miguel Oliver" and "Vizconde de Eza" for their patience.

REFERENCES

[1] http://www.baslerweb.com/

[2] http://www.jai.com/en/

[3] http://es.mathworks.com/products/matlab/

[4] B. Stroustrup, The C++ Programming Language, Reading (Massachusetts, U.S.A.): Addison-Wesley, 1997.

[5] http://opencv.org/

[6] A. K. Jain, Fundamentals of digital image processing, Englewood Cliffs, (New Jersey, U.S.A.): Prentice Hall, 1989.

[7] G. Wyszecki, W.S. Stiles, Color science: concepts and methods, quantitative data, and formulae, New York: John Wiley & Sons, 2000.

[8] R. González, Digital image processing using MATLAB, Upper Saddle River: Gatesmark Publishing, 2009.

[9] J.C. Daugman, "Two-dimensional spectral analysis of cortical receptive field profiles", Vision Research, 20, 847-856, 1980.