Practical applications for robotic arms using image processing

Conference Paper · January 2012

CITATIONS
17

READS
9,894

3 authors, including:

R. Solea
Universitatea Dunarea de Jos Galati
70 PUBLICATIONS **482** CITATIONS

SEE PROFILE

Practical Applications for Robotic Arms Using Image Processing

Mihai Dragusu[1], Anca Nicoleta Mihalache[1] and Razvan Solea[2], *member, IEEE*

***Abstract*— The following paper presents various applications for a robotic arm using image processing. These applications have as a basis a video camera that captures the images to be processed for either recognition and sorting or for contour coordinates extraction. Based on these captures, there were developed algorithms that enable the arm to either sort objects recognized to be having different shapes, or to draw shapes based on the coordinates received. Both the object detection and the contour coordinates extraction methods are implemented using a series of image processing techniques like border extraction, contour detection, contour extraction, along with match template and the aid of OpenCV library. Also, a PI controller that ensures the position of the robotic arm was developed.**

I. INTRODUCTION

The influence and impact of digital images on modern society, science, technology and art are huge. Image processing has become such a critical component in contemporary science and technology that many tasks would not be attempted without it. It is a truly interdisciplinary subject and is used in computer vision, robotics, medical imaging, microscopy, astronomy, geology and many other fields [1] - [5].

One of the core technologies of the intelligent robot was, of course, vision, and this vision research area was called "computer vision", which is now regarded as a fundamental and scientific approach to investigate how artificial vision can be best achieved and what principles underlie it [6].

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. OpenCV is free for both academic and commercial use. It has C++, C, Python and soon Java interfaces running on Windows, Linux, Android and Mac. The library has more then 2500 optimized algorithms [7].

OpenCV played a role in the growth of computer vision by enabling thousands of people to do more productive work in vision. With its focus on real-time vision, OpenCV helps students and professionals efficiently implement projects.

The contribution of this paper is to report new practical applications for student laboratory that provide the hands-on experience that is critical for robotics education. These types of exercises supply the student with hands-on experience that complements classroom lectures and software development. Through this experience, the student confronts the hard realities of robot systems; realities such as sensor noise, nonzero calibration residuals, low fidelity of models, and real-time constraints; and learns to deal with them.

[1]M. Dragusu and A.N. Mihalache are students at the Faculty of Automatic Control, Computers, Electric and Electronic Engineering, "Dunarea de Jos" University of Galati, Romania dragusumihai@yahoo.com, mihalache_anca@ymail.com

[2]R. Solea is with the Faculty of Automatic Control, Computers, Electric and Electronic Engineering, "Dunarea de Jos" University of Galati, Romania razvan.solea@ugal.ro

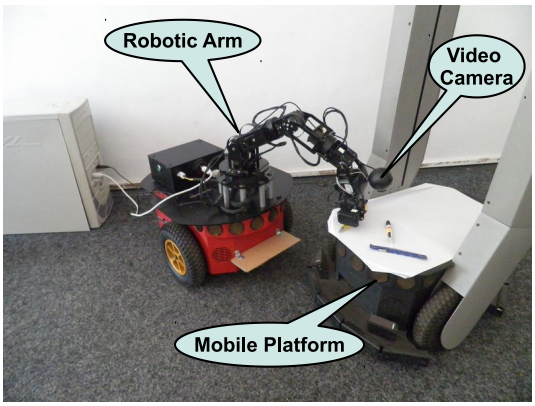

Fig. 1. System hardware components

The goal of this paper is to provide an overview of the practical laboratory applications and the required facilities. As an overview, the paper will provide enough practical information to replicate the applications.

II. PRACTICAL APPLICATIONS - ALGORITHMS

The system is designed for processing images captured by a web cam so important information can be extracted and sent to the robotic arm. The robotic arm has a software implemented, so it can use the information received in order to complete its given task: either recognize shapes for sorting or coordinates extraction from a contour for drawing.

System hardware components are (see Fig. 1): video camera and processing tool, robotic arm and mobile platform.

A. Contour Coordinates Extraction

The web cam captures the image to be processed. In order to obtain information about this capture, edge detection applied to the image is necessary.

Of all the edge detection techniques (Canny, Sobel, Laplace, etc.), Canny method [10], [11] was chosen because of its good performances with noisy images. This algorithm can only be applied on a grayscale blurred image, and it will find all the edges from the image, based on which contours will be formed.

After using Canny algorithm on the blurred image, it has been decided that each contour would have associated a different shade of red for each limited entity, starting from 254 and decrementing this value in successive steps with 1 until reaches the value 10. This means the detection and distinguishing of up to 244 discontinuous contours. But in

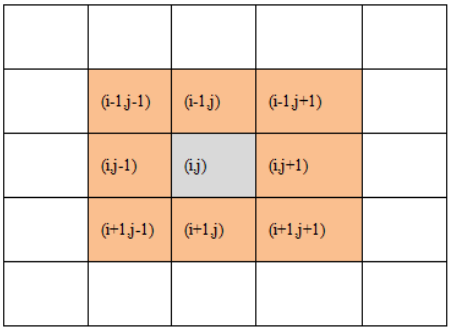

Fig. 2. The Moore neighborhood in coordinates

the captured image there can also exist continuous contours, that can be detected as such, and be assigned another color (blue).

These operations result in an image that contains all boundaries (edges) found in the web cam capture. These contours will have their coordinates extracted in order to be sent to the robotic arm.

It has been chosen the design of an algorithm that is optimal for the issue at hand, the drawing with the robotic arm. The image to be processed is analyzed as a matrix, and the coordinates (x, y) of a given color are saved in a matrix of unsorted coordinates.

For the drawing to be facilitated for the robotic arm, it is of the essence that the drawing of a contour does not start from a random point that first found by the algorithm, but from a start pixel of that contour (a pixel that only has one neighbor in the Moore neighborhood - see Fig. 2).

Once the start pixel is found, based on its coordinates, the sorting of addresses from the unsorted coordinate matrix will follow. This sorting is based on applying an optimization criterion that ensures finding the neighbor of the current pixel. This criterion is represented by the distance between the pixel considered current pixel, and the pixel analyzed from the unsorted addresses matrix. The sorting of the addresses is made based on this criterion, and the update of the current pixel is made by assigning the last pixel to be sorted at any point in the algorithm as the current pixel for the next iteration.

Once the address of a pixel is sorted, its coordinates are to be erased from the initial matrix, and added in the sorted coordinates matrix. This is easily followed in the following code:

```
...

if (dist < min_dist)  //if current distance
          is lower that the minimum distance
  {
    min_dist=dist;   //current distance is
                          minimum distance
    poz_adr=i;
    adr_ord[linie][0]=adr1[i][0];
    adr_ord[linie][1]=adr1[i][1];
  }
else
  {
    if(dist==min_dist)
      {
```

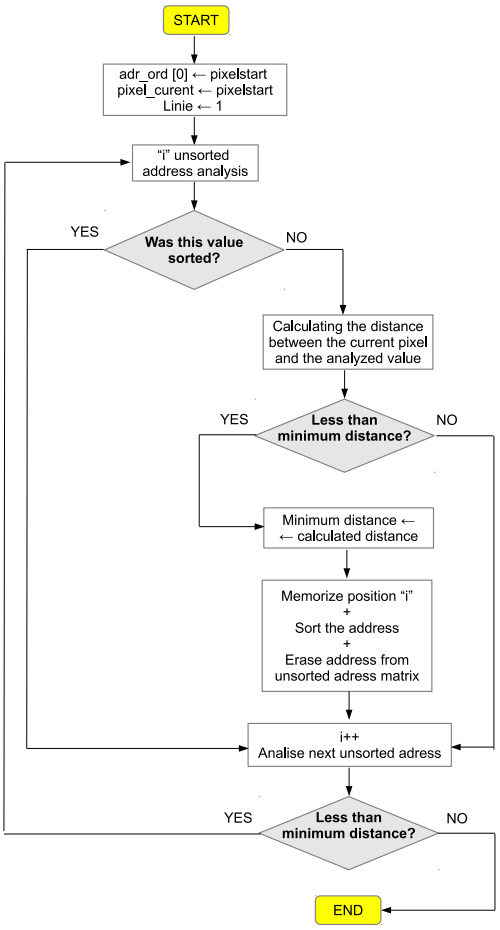

Fig. 3. The sorting algorithm

```
        adr_ord[linie][0]=adr1[i][0];
        adr_ord[linie][1]=adr1[i][1];
      }
  }
adr1[poz_adr][0]=-1;
adr1[poz_adr][1]=-1;
pixel_curent[0]=adr_ord[linie][0];
pixel_curent[1]=adr_ord[linie][1];
linie++;
 ...
```

The sorting of contour coordinates algorithm is presented in Fig. 3.

This implementation prevents the double passing on the same contour/region. So with this solution, no coordinate once sorted would be ever checked again. This is a very important aspect, considering the task given, because drawing the same piece of contour several time is not only useless for the arm, but also it can also emphasis the hardware limitations of the system-drawing the same piece of contour will not overlap on several drawings, even if the given coordinates are the same ones. This event occurs because of the hardware parts involved, like the arm′s encoders.

B. Robotic Arm Software for Sorting Algorithm

In order to obtain information about the presented object, edge detection applied to the image is necessary. For the

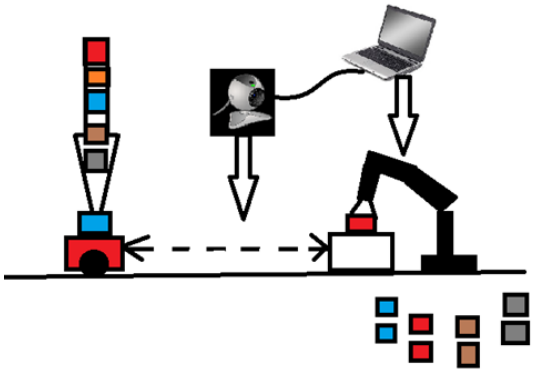

Fig. 4. Object recognition and sorting solution

same reason, Canny method was again chosen. After the edges are extracted from an image a contour (chain of neighbor pixels) finding method can be applied. Some contours may contain the object, since the edge of one object is a chain of pixels.

The actual recognition is realized with match-template method, which consist in placing a frame containing the object on top of the current image, calculating the sum squared error and then moving the object through the image in order to determine where the sum squared error is minimal.

It can be easily observed that the method above can prove inefficient if the number of objects to be found increases, or the size of the image increases. The application will end up looking for a 50×50 image in a 1920×1080 image filled with irrelevant data except for the actual object.

So in order to increase efficiency another method was designed based on match-template. This new method will use edge detection, contour detection, and contour extraction on the current image and only then will match-template be used between the searched object and the object extracted from the current image (resized to the dimension of the searched object). This way the application will use match-template only on relevant data, not blindly on the entire image.

For object sorting algorithm (see Fig. 4), the robotic arm will go through four states:

- A initial position at which the robotic arm will go once the control application is launched. From this position the camera placed on the arm will capture the image containing the object in order to determine its category.
- A state in which the gripper is brought to the object's level in order to grab it.
- Return to the initial position, but this time with the object grabbed (this state is an intermediate state to avoid collision with the mobile platform if the arm were to go directly to the last state)
- The last state is actually a multitude of states generated by the object's category (each category has assigned a different location)

The object recognition and sorting algorithm is presented in Fig. 5, where: $Xdorit$ is the position where the mobile

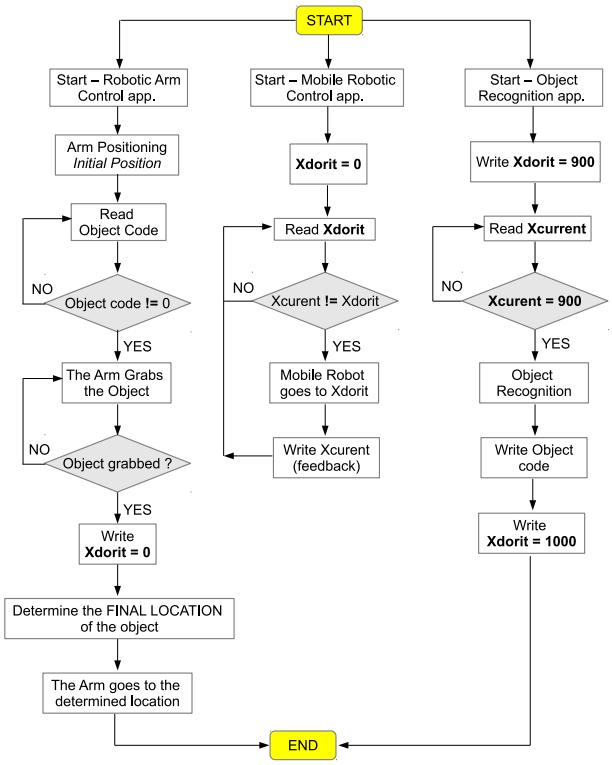

Fig. 5. Object recognition and sorting algorithm

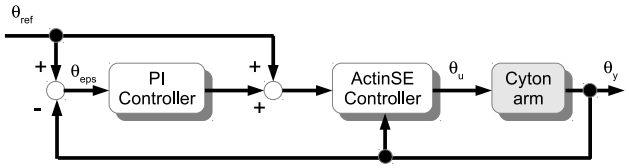

Fig. 6. The control structure of robotic arm

platform should be (0 initial postion, 900 $[mm]$ in front of the camera to determine the object, 1000 $[mm]$ in front of the robotic arm so that it can grab the object); $Xcurent$ is the feedback given by the platform; $ObjectCode$ is an number which represents the detected object (the algorithm uses this number to determine the depositing position).

The control software $(actinSE)$ [9] implemented for this robotic arm has a controller designed, but this controller does not offer the precision needed for this project so a PI controller (see Fig. 6) was designed to correct the command given by the $actinSE$ controller.

III. FIRST PRACTICAL LABORATORY APPLICATION - EXPERIMENTAL RESULTS FOR IMAGE DRAWING

A. Step 1 - Image capture and processing

The web cam included in the system makes an image capture to be processed. Firstly it is needed to set a blur and a Canny threshold according to the image, environment and noise of the camera, with the help of this user-friendly interface, Fig. 7.

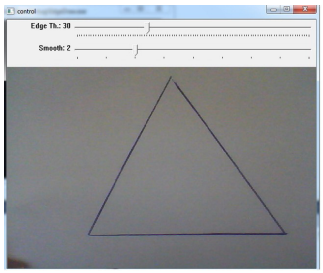

Fig. 7. Capture of the image to be drawn

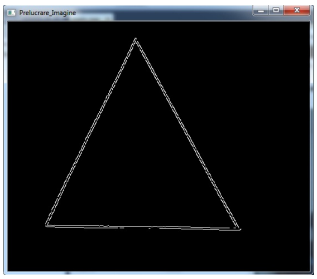

Fig. 8. The result of Canny algorithm on the captured image

The following code sequence is a demonstration on how the capture is prepared for Canny algorithm, by transforming it in a blurred single-channel image:

```
frame = cvQueryFrame( cv_cap);
cvSmooth(frame,frame,CV_GAUSSIAN,smoothD,smoothD);
IplImage* frgray = cvCreateImage( cvGetSize(frame),
           IPL_DEPTH_8U, 1 );///frame in grayscale
cvCvtColor(frame, frgray, CV_RGB2GRAY );
```

The result of applying Canny algorithm is presented in Fig. 7, obtained with the following code sequence:

```
cvNamedWindow("Prelucrare_Imagine",0);
frame = cvQueryFrame( cv_cap);
cvSmooth( frame, frame,CV_GAUSSIAN,smoothD,
                                smoothD);
cont=Canny(frame);
cvShowImage("Prelucrare_Imagine", cont);
```

As mentioned, these contours are to be colored in red for an easier analysis.

B. Step 2 - Robotic arm drawing coordintes

The coordinates that were extracted from this image are the Cartesian coordinates to be rescaled and translated in polar system. But for demonstration purposes, these coordinates were used for a Matlab simulation, where it was created a matrix of addresses to be plotted:

```
for i=1:1:s(2)
    plot(a(1,i),a(2,s(2)-i));
    hold on
    pause(0.001);
    axis([0 640 0 640]);
end
```

And the image that results is drawn fluently point by point for an easier observation on where the contour starts and how the contour drawing is progressing, as if the arm would be drawing. The final result of the Matlab simulation is described in the Fig. 9.

Robot arms offer significant advantages for robotic tasks. With many degrees of freedom, they are able to reach around obstacles, reconfigure for strength, improve accuracy, and manipulate objects with dexterous fluid motion.

Combined with Energid's 3D visualization, reasoning, and control software, the Cyton V2 manipulator [8] can perform advanced control by exploiting kinematic redundancy. Cyton have 7 DOF plus gripper, all axes are completely independent yet can be controlled simultaneously (using the included Actin control software). Also, it has high performance, intelligent actuators give feedback on position, speed, load, voltage, and temperature.

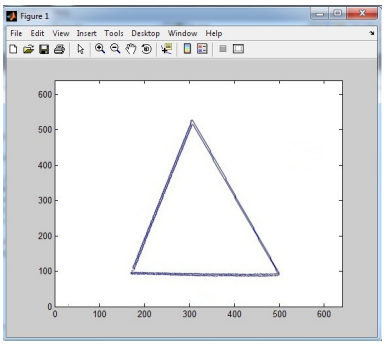

Fig. 9. Matlab simulation of drawing the extracted coordinates

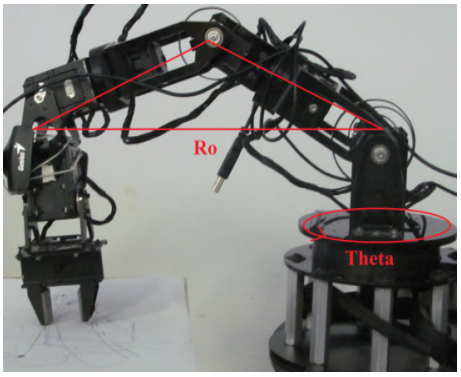

Fig. 10. Polar coordinates defined on the robotic arm

As for the drawing algorithm, as it was mentioned before, the coordinates provided by the contour extraction algorithm are in Cartesian system, so for the actual drawing process, they need to be converted to polar coordinate system.

The real space in which the arm can draw is limited because the distance it can reach (ρ) is not scaled correctly related to its angle (θ) (see Fig. 10. In order to solve this problem, a new scaled space had to be defined like in Fig. 11.

The polar coordinated for this space (P) were experimentally determined, and then transformed back into Cartesian coordinates (C).

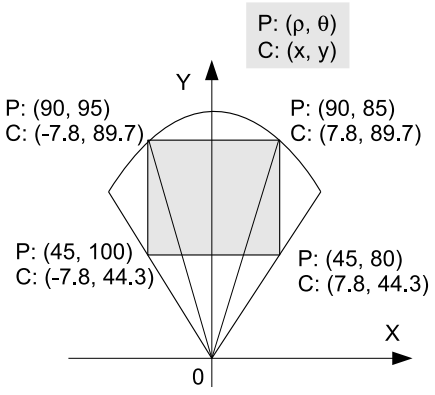

Fig. 11. The scaled space defined in polar coordinates

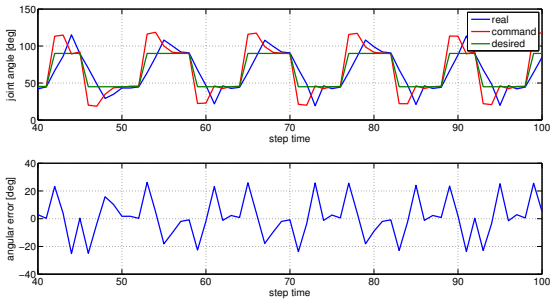

Fig. 12. Joint angle variation

Knowing that the image has a width of 640 pixels and a height of 480 pixels, and based on the C coordinates calculated, two constants, one for each axis, needed to be determined, so that the coordinates extracted from the image can be scaled to this newly defined space:

- x axis rescale $K_1 = 15.6/640 = 0.024$, where $15.6 = 2 * 7.8$ the distance between -7.8 and 7.8;
- y axis rescale: $K_2 = 45.4/480 = 0.094$, where $45.4 = 89.7 - 44.3$, roughly the distance between 89.7 and 44.3.

Once these constants have been defined, the rescale formula for the Cartesian coordinates is:

$$\begin{aligned} x_r &= x * K_1 - 7.8 \\ y_r &= y * K_2 + 44.3 \end{aligned} \tag{1}$$

where x_r and y_r are the rescaled coordinates and x, y are the initial coordinates provided by the extraction algorithm.

So after rescaling the drawing space and the coordinates received from the extraction algorithm and transformed into polar coordinate system, the arm can now draw the shapes presented to the camera, with a low degree of difficulty, considering the limitations it has.

Fig. 12 shows the desired, command and real angular position of the first joint of the manipulator and the angular error.

IV. SECOND PRACTICAL LABORATORY APPLICATION - EXPERIMENTAL RESULTS FOR SORTING ALGORITHM

A. Step 1 - Learning Process (object extraction)

During this process the user will help the application learn which objects to detect, using the following steps: i) the image is captured (Fig. 13 and the edge detection is applied ("c" button); ii) the contours are determined ("d" button); iii) the objects captured by the contours are browsed ("n" button) until the user finds the desired object (see Fig. 15) and saves it ("s" button).

B. Step 2 - Recognition principle

During this process (see the diagram: Fig. 17) the camera feedback is constantly checked to see the same object extraction is applied for each frame only now for each object extracted match template is applied for each object saved. When the sum squared error between two objects is less than

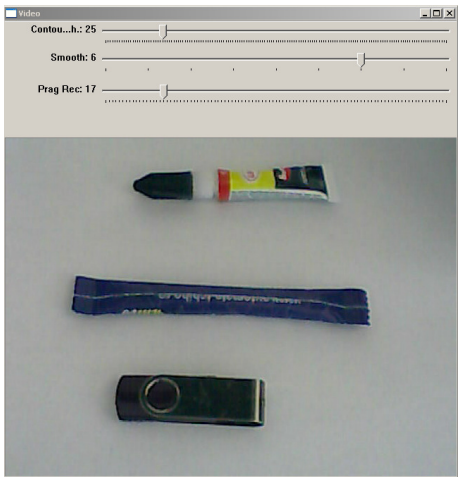

Fig. 13. Captured image

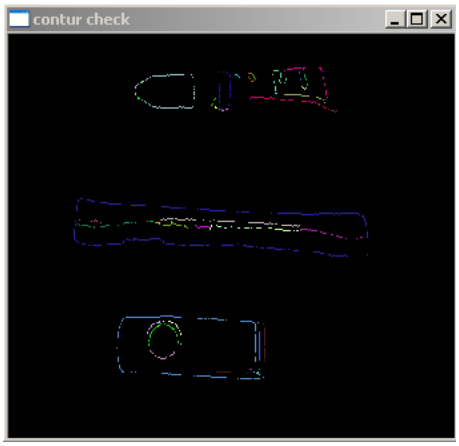

Fig. 14. Canny result

a predetermined threshold then we can say that the object we found (see Fig. 16 is the one were looking for.

The described process can find multiple objects in a single image.

In the following example the searched object is an USB flash stick.

In Fig. 18 the found object is declared ("stick.jpg gasit") and its sum squared error compared to the saved one (0.0736217), as you can see other saved objects have been compared with the found object in the current image but none had the sum squared error $(0.284048$ and $1)$ low enough for a mismatch.

C. Step 3 - Object depositing

After the object has been found it's assigned a code that is communicated to the arm's control software which has specified places for each object type to be deposited.

To show the effectiveness of the proposed algorithm, experiments was made using a Pionner Robot mobile platform (like in Fig. 1). The mobile platform will pass through three states: i) Positioning at the input buffer coordinates; ii) Positioning under the camera in order to determine the

Fig. 15. Saved objects - example

Fig. 16. Showing the found object

object, and to grab the object with the robotic arm; iii) Return to the initial position.

All three states are communicated (in form of Cartesian coordinates) to the mobile platform using ARIA (Advanced Robot Interface for Applications) software. The main application will write the desired position for the platform and the control application of the platform will read the file and command the robot to go to that position. ARIA software can be used to control the mobile robots like Pionner, PatrolBot, PeopleBot, Seekur etc. ARIA it is an object-oriented Applications Programming Interface (API), written in C++ and intended for the creation of intelligent high-level client-side software. ARIA provides tools to integrate I/O and includes comprehensive support for all MobileRobots robot accessories, including the laserrange finders, control of the pan-tilt-zoom camera or pantilt unit, Pioneer Gripper and Arm, and more.

V. CONCLUSIONS

In this paper it was presented practical laboratory applications in robotic manpulation, computer vision, and mechatronics. This applications provide the hands-on experience that is essential for robotics education.

The integration of real-world problems with complex imagery can improve image-processing applications. Effective applications-based exercises should be closely aligned with lecture concepts and should be assigned with little time delay. An "OpenCV" environment allows a focus on the imaging concepts and techniques with minimal programming difficulty.

Both applications, the object detection and the contour coordinates extraction methods are implemented using a series of image processing techniques like border extraction, contour detection, contour extraction, along with match template and the aid of OpenCV library.

In future work, motivated by this work, we consider the situation when mobile manipulators are sorting objects and placed the objects in a known location.

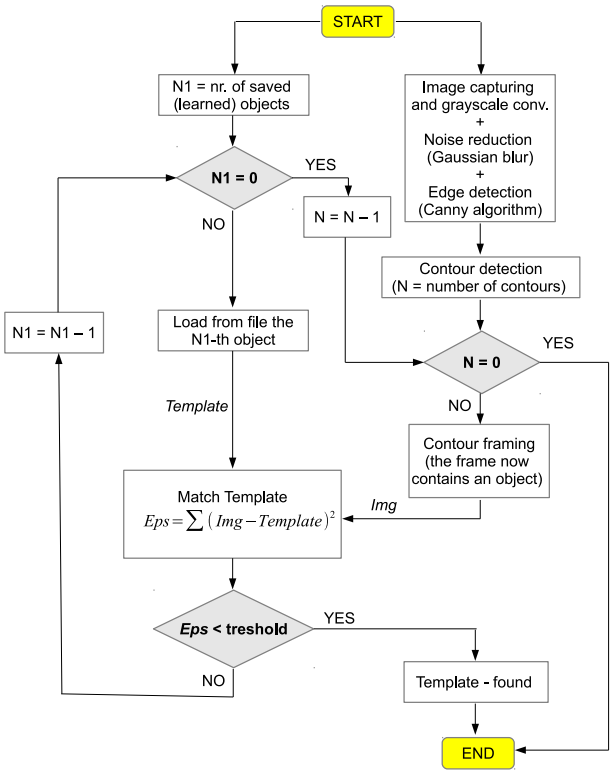

Fig. 17. The object recognition algorithm

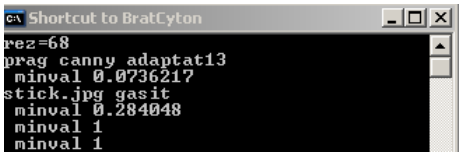

Fig. 18. Result of recognition algorithm

REFERENCES

[1] C. Theis, I. Iossifidis and A. Steinhage, Image Processing Methods for Interactive Robot Control, Proceedings 10th IEEE International Workshop on Robot and Human Interactive Communication, 2001, pp. 424-429.
[2] G. Dougherty, Digital Image Processing for Medical Applications, Cambridge University Press, 2009.
[3] M. Pesenson, W. Roby and B. McCollum, Multiscale Astronomical Image Processing Based on Nonlinear Partial Differential Equations, The Astrophysical Journal, 683 (1), 2008, pp. 566-576.
[4] S. A. Drury, Image Interpretation In Geology, Nelson Thornes, UK, 2001.
[5] Yu-fai Fung, H. Lee, and M. F. Ercan, Image Processing Application in Toll Collection, IAENG International Journal of Computer Science, 32(4), 2006, 6 pages.
[6] M. Ejiri, T. Miyatake, H. Sako, A. Nagasaka and S. Nagaya, Evolution of Real-time Image Processing in Practical Applications, IAPR Workshop on Machine Vision Applications, Japan, 2000, pp. 177-186.
[7] G. Bradski and A. Kaehler Learning OpenCV (Book style), O'Reilly Media, Inc., USA, 2008.
[8] Robai, CYTON V2 7DOF - Operating Manual, USA, 2010, 36 pages.
[9] Energid Technologies, Energid ActinSE Documentation, http: //www.robai.com/content/docs/ActinSE_docs/.
[10] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 3rd Edition, Prentice Hall, 2008.
[11] W. K. Pratt, Digital Image Processing: PIKS Inside, 3rd Edition, John Wiley & Sons, Inc., 2001.