Image processing tasks applied to robot vision system and path discovery (March 2016)

Rafael Custodio Cejas, Italo Guedes A. Silva, Advisor: Prof. Sarosh Patel
Computer Science and Engineering Department
University of Bridgeport
Bridgeport, Connecticut 06604, USA
{iguedesa, rcustodio}@my.bridgeport.edu, saroshp@bridgeport.edu

***Abstract*—This work presents the project of a mobile robot with an attached handler, programmed to collect small objects in the near area. The robot takes photos –is able to process a constant stream of images– of the surround area searching for objects, and goes to the nearest one to collect. After collecting, the robot has to put the object in a specified place. The robot will keep searching for objects and moving while on. The image processing job is to do a contrast between the floor color and the colors of the objects, and then the algorithm provides the location to move on. The project is built with a Raspberry Pi/Raspbian platform running a python program that uses the OpenCV library to process images and parameters used to guide the robot, a simple webcam that provides high resolution images, and the motors as final actuators to control the movement of the robot and the handler.**

***Index Terms*— Computer Vision, Gripper, Image Processing, Machine Vision, Mobile Robot, OpenCV, Path Discovery, Python, Raspberry Pi, Robotics.**

I. Introduction

A robot capable of searching for objects, collecting them, and taking them to another place. A small robot that can see what is around it, detect if is a small object, and decide what to do. Our paper describes how works, and which tools were used to build our object-collector robot, explaining some image processing techniques that we applied to our project. The component parts of the robot, how they are connected to each other, and why they were chosen. We will describe how the robot will find the path to deposit an object found, and the solution we found to make it an easy job. Also, some improvements can be taken to optimize the search for objects in the future.

II. Image Processing tasks

One of the most used approaches used in designing machine vision systems is based on recording the external environment –workspace– with a digital camera to obtain images corresponding to the real state of the surrounding area as a primary source of information. These images can be easily processed –despite the hardware restrictions– using image analysis software to match a requirement formulated with the purpose of achieving a previously designed goal. With the constant improvement of computer hardware, digital cameras, software tools and easy-to-learn libraries, more and more applications that deals with pattern recognition in images like the one described can be easily deployed. One example of a modern library that provides a basic infrastructure for image-analysis and largely used in this project is OpenCV. OpenCV is basically a library of C functions that were written to handle infrastructure operations and image processing tasks. Some of the features provided include I/O functions, its own in-memory data organization for an image with structural information about the image data, methods to get and set individual pixels in the image, basic pre-programmed transformations, and a display where it's possible to visualize the output of the current task. For the purpose of meeting the requirements of this project, the version of OpenCV chosen was OpenCV-Python. This version is basically a wrapper that allow us to use the C functions for image processing and other tasks from a python script.

The application used in this project uses the camera features of the OpenCV library to obtain a constant stream of images from the onboard webcam. Every single frame is processed with the purpose of locate objects around the robot and to distinguish between them and the floor. Some transformations are applied to the obtained frame to eliminate any noise that can make harder to distinguish the objects, and other used transformations include an implementation of an edge detection

This paper was first submitted for review in 03/01/2016. This work was supported in part by the Brazilian Scientific Mobility Program, a partnership between Institute of International Education (IIE) and Capes to grant undergrad students with scholarships.

Rafael C. Cejas, was with Federal University of Para, Belem, PA, Brazil. He is now with the Computer Science and Engineering Department, University of Bridgeport, Bridgeport, CT 06604 USA (e-mail: rcustodio@my.bridgeport.edu).

Italo Guedes A. Silva, was with Federal University of Campina Grande, Campina Grande, PB, Brazil. He is now with the Computer Science and Engineering Department, University of Bridgeport, Bridgeport, CT 06604 USA (e-mail: iguedesa@my.bridgeport.edu).

technique and color segmentation, both topics better explained below.

The following functions will take care of making the real world machine-visible, that is, process the captured images in a way that the machine can "understand" what is around it, and then make decisions based on that. Our robot need to see what is around it, and decide either to move ahead, to collect an object, or to turn around. Our robot will convert the color scale, apply filters, change the morphology, and detect edges in the images to keep only the needed information, and define his next path. It may seem difficult, but is as simple as follow the steps.

A. cvtColor

This function is used to convert the original image to other scale of colors. In this case, we will use this function to convert a colorful image to grayscale. This is an important task to do before applying the filters. The function has as parameters the source image, and a code of an enumerator to choose the desired scale, which, in this project, we will use cv2.cv.CV_RGB2GRAY for the objects detection, and cv2.cv.CV_BGR2HSV for distinguish the floor and the walls.

cv2.cvtColor(src, code) → dst

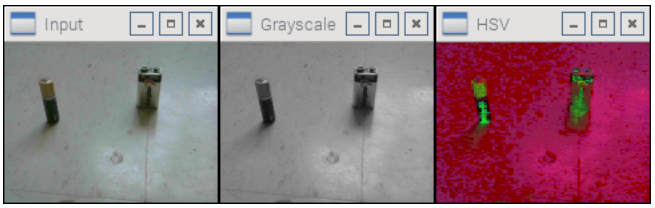

Fig. 1 – Original frame converted to grayscale, and to HSV color system.

B. Gaussian Filter

The first step in processing the image to detect objects is to smooth or blur the image. This effect has the main objective of decrease the small details and the high-frequency noise in an image, often caused by digital cameras. The function that has this responsibility in OpenCV is the *GaussianBlur*. This function applies a Gaussian filter to the image, by doing a convolution with each point (x, y) of the image with a Gaussian kernel, that works like the coefficients of the filter. The result of each operation will be added all together to form the output point. This operation will take point by point of the image and after all points were convolved, we'll have the output image with the same size and same number of channels as the input image.

A kernel is an array of coefficients that are convolved with the pixels of an image. The Gaussian kernel is a low-pass kernel, because it passes through the lower frequencies and decrease the higher frequencies. An example of a Gaussian kernel is shown in the figure 2, which shows a higher weight in the central pixels compared to the others.

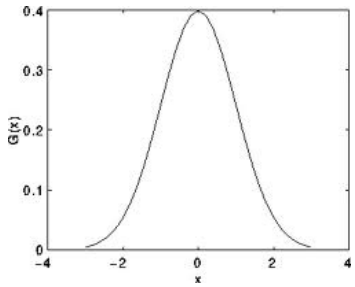

Fig. 2 – Example of a Gaussian Filter.

The function of OpenCV used was the GaussianBlur, which takes the parameters src, ksize, and sigmaX. The src parameter is the input image, that was captured by the camera. The ksize parameter are a composite of ksize.width and ksize.height, which determines the size of the Gaussian kernel to be used in the filtering, both width and height must be odd and positive.

cv2.GaussianBlur(src, ksize, sigmaX) → dst

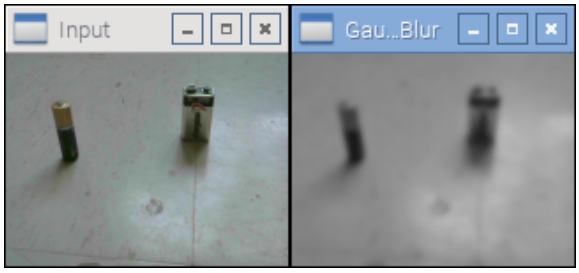

Fig. 3 – Input frame processed using GaussianBlur function.

C. Canny edge detector

The second step to process our image is to using an edge detector. This function identifies the edges of an image by detecting regions with rapid color intensity variation. Inside this function, the Gaussian filter is applied again to make sure all unintended details are without focus, then begins a procedure to find the intensity gradient (strength and direction) of the image. At this time probably all edges were identified, but if there is any pixel left that are not part of an edge still on the image, they will be eliminated by a *non-maximum* suppression. The last step is the hysteresis, that select which pixels should remain in the image and which should be deleted based on value of the pixel gradient (P), using the lower threshold (T_{lower}) and the upper threshold (T_{upper}).

If P > (T_{upper}) => Pixel accepted
If P < (T_{lower}) => Pixel rejected
If (T_{lower}) > P > (T_{upper}) => Pixel accepted only if it has a neighbor pixel above upper threshold

The function of OpenCV used was the Canny, which takes the parameters image, lower_threshold, and upper_threshold. The image parameter is the input image of the function. The lower_threshold and upper_threshold parameters define the threshold values explained in the last step.

cv2.Canny(image, lower_threshold, upper_threshold) → edges

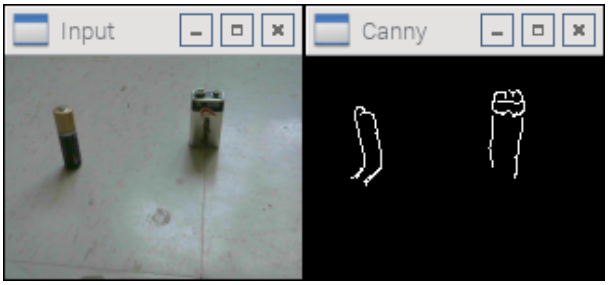

Fig. 4 – Canny edge detection.

D. Dilation and Erosion

The dilation and erosion operations are very similar, the difference between them is that while the dilation computes the maximum pixel value during a convolution, the erosion computes the minimum pixel value. The image is convolved with a kernel, which can be of any size or shape, usually a circle or a square. The kernel has a specific anchor point, usually at its center. As the kernel is being convolved through the image, we replace each pixel as the anchor point position with the pixel value overlapped by the kernel in that window. In a dilated operation, the bright regions of the image will be dilated, and dark reduced. And in a eroded operation, the dark will be increased, and the bright eroded.

E. MorphologyEx

The next function of OpenCV used in our project is morphologyEx. This function performs morphological transformations in the image and, in this case, applies two effects sequentially. First, the image goes through an erosion effect, removing small objects and possible artifacts. Second, the dilation effect, which dilates the remaining pixels to enhance the size of the real objects in the image. This combination of erode and dilate functions is called opening.

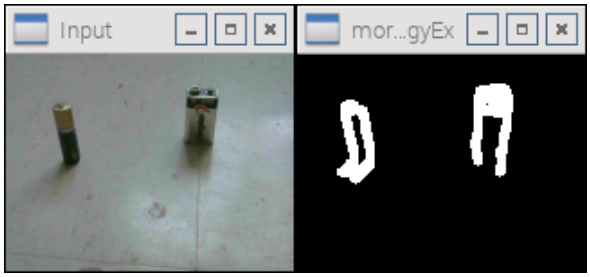

Fig. 5- Morphological transformation applied to input image

F. inRange

This function is used to detect if the robot is in front of a wall. We basically define values of white and gray to the floor, and detect the percentage of non-white elements in the image. If this value is almost 80%, then this means that the robot might be in front of an obstacle and must turn around.

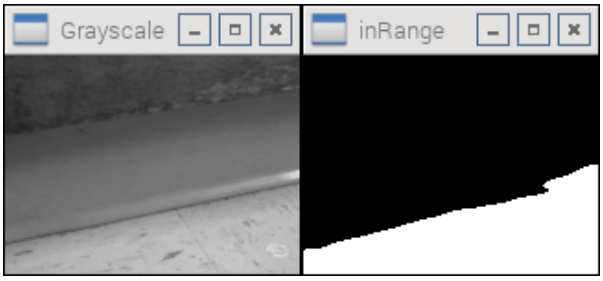

Fig. 6- Obstacle avoidance using inRange function

III. ENVIRONMENT PERCEPTION AND PATH DISCOVERY

The next step after a processed frame is obtained according to the image-analysis techniques described, is to feed the next pipeline stage with parameters that will help our application to take measures and decisions –like obstacle avoidance– about its current set up and to set the appropriate digital signals used to control the actuators –the wheels of the robot. As our project deals with object detection, an OpenCV function called findContours is used to obtain details of the objects in the frame. The cv2.findContours uses a binary image as input –frame obtained after the inRange operation takes place– and returns a set of contours found in the frame. Every contour found is stored as a vector of points and represents only the extreme outer contours of the object, this feature is specified with the parameter cv2.RETR_EXTERNAL –other modes are allowed, but this one fits the requirements of our application very well. Other important parameter is the method that tell us about the internal representation of the contours points in the vectors. We use the cv2.CV_CHAIN_APPROX_SIMPLE option, that compresses segments and leaves only their end points. These points are used later as arguments to a helper function that draws rectangles which will be used later to see the contours found in the output display as a way to obtain feedback of the objects detected.

cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CV_CHAIN_APPROX_SIMPLE)

One more important function is called cv2.contourArea(c). This one is used to calculate the area of the contours specified by a vector of 2d points –like the one that draws rectangles. The value returned is used in the process of decision-making, it can tell us if the area of the objects detected are in the range specified for the targets or if it's an obstacle to avoid –like a wall. Our implementation of obstacle avoidance using this approach consists of verifying if the contour's area is greater than a certain value. Any detection that exceeds the proposed value is a large area in the frame that is considered an obstacle. Using the same principles, areas too small are not considered objects the robot may interact with. One of the decisions to make is decide which path must be followed to achieve the goal of collecting the target objects. In our project we have considered the best approach to collect the nearest objects in the workspace. This approach is implemented maintaining a pair of variables that keep the values of the coordinates of the nearest object found until then. The area of the contours found through

the call to cv2.contourArea(c) are iterated in a way that the nearest object to the robot can be determined.

Once the nearest object has been determined, some calculations are realized in order to centralize the object in the middle of the current frame in a way that it can be collected by the attached gripper. The object's centralization is obtained by dividing the frame in sections, sections with 1/7 of the width of the screen from a line that divides the frame in two halves according to the picture bellow. Some movements –move to left or move to right– are executed by the robot in order to change the camera's position to a frame that contains the object located in the middle of the screen (Figure 7). Once the gripper grabs the target object, the procedure to find the spot of deposit takes place.

The approach used to implement the path discovery of the end point is basically to find the place to discard the objects taking into account a line whose function is to set the workspace of the robot. One line that has been previously drawn to the floor in a specific color is used to determine the robot's workspace and everything outside the delimited area is considered out of range by default. This line can be identified by applying a different image-analysis procedure than the used to identify the objects. This technique basically applies color segmentation to the frame to be processed in order to obtain an output frame which can be used to determine if the line is near the path followed by the robot. Once the line is reached, it will be followed until the end point –collector– is reached too. After dropping the object collected, the robot turns around and returns to the space delimited by the line and executes the object detection procedure once again.

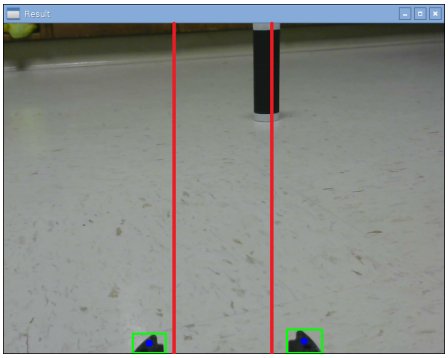

Fig. 7 – Object's centralization in the frame

IV. PLATFORM OVERVIEW

The following list shows the components of the robot, and below a brief explanation about why we choose each one.

- Raspberry Pi™ 2 Model B
- Logitech HD C270 Webcam
- Edimax N150 Wifi USB Adapter
- MicroSD Card 16 GB
- L298N Dual H Bridge
- 2 Wheel Drive Robot Chassis
- Power Bank, JETech® 10,000mAh
- Standard Gripper Kit A
- Servo High Torque Standard Size

We choose the Raspberry Pi as the main board for our project because it has enough processing power to deal with image processing softwares, and deliver in real time the results. It also has the advantage of the GPIO pins, which can control the motors based on the result of the processing. Its compatibility with USB peripherals made possible the use of a common web camera to capture the images, and a simple Wi-Fi adapter to connect to a wireless network. Finally, it is a low power computer that can work for hours with a common battery.

The Logitech Webcam, the Edimax Wifi adapter, and the MicroSD card were picked because of its cost-benefit, compatibility with Raspberry Pi, and ease installation and use. We needed to make a mobile robot, so we searched about small motors and chassis, and we found that a 2 Wheel Drive Robot Chassis, controlled with a L298N are totally compatible with Raspberry Pi, and has the ideal size to fit all elements on it. For the gripper, we found the Standard Gripper Kit A along with a compatible servo that has a compatible size with the chassis, and is also a good catcher to small objects. Finally, we choose a JETech power bank with 10000mAh, that is more than enough to power on the board with its peripherals for hours.

The camera and the Wi-Fi adapter are directly connected on the USB ports of the Raspberry Pi, which is powered by the power bank through its microUSB port. The motors of the chassis are connected to the H bridge, that is responsible for changing the orientation of movement of the motors. 4 AA batteries are connected to the H bridge to power the motors.

V. RESULTS

To test our system, we have set a workspace such a way that the robot could realize its work properly. The environment consists of a square area surrounded by a line that identifies the boundaries of the workspace as can be seen in Fig. 8. The objects in cylindrical form were chosen such a way that the robot's gripper can grab them easily from every position possible. Some parameters were corrected later to ensure that a 180-degree turn could be realized without causing damage to the platform.

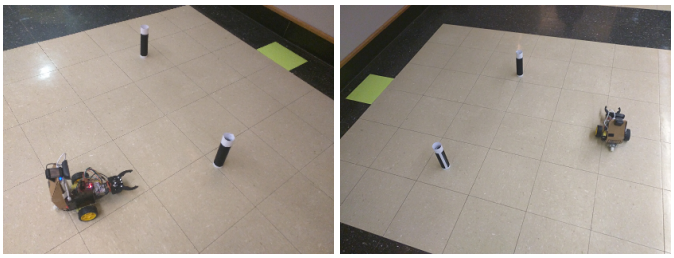

Fig. 8 – Robot moving to first target detected.

The deposit of the object was as expected. Fig. 9 shows a snapshot of the moment the object is dropped.

Fig. 9 – Robot leaving object in marked area.

The moment following the deposit of the object was registered as well. Fig. 10 shows a snapshot of the moment the robot returns to the workspace.

Fig. 10 – Robot returning to workspace.

To verify if the image processing program was pushing the hardware to its limits we've ran a simple test. The results can be seen in Fig. 11; it shows us that the application wasn't using even ¼ of CPU's usage. And the usage decreases even more when we consider not using a graphical interface. This demonstrates the reason why, although of the intense repetitive tasks of processing a streaming image, Raspberry Pi was able to assure the flow of the process. That is why Raspberry Pi is the best choice for the project.

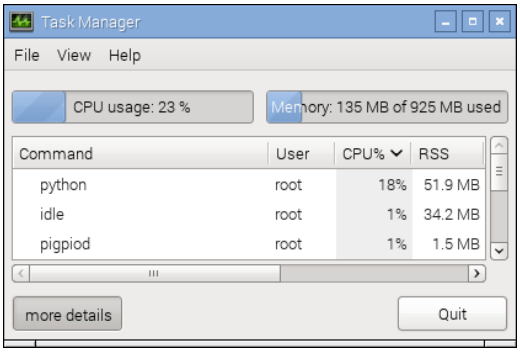

Fig. 11 – Hardware usage by application.

VI. CONCLUSION

In the work presented we gave brief explanations about some of the powerful functions of OpenCV for image processing, and how they can be combined to basic techniques to extract the information we want from the workspace. By doing this project we could begin our journey into the image processing world, becoming more interested in the relationship that it has with robotics, electronics and programming. We faced difficulties, but we were able to overcome them by searching the web, studying the functions, and trying by our own.

ACKNOWLEDGMENT

The Authors thank the help of Prof. Sarosh Patel and Peter Zeno as advisors for this project.

REFERENCES

[1] J. R. Parker, *Algorithms for Image Processing and Computer Vision, 2*th ed. Indianapolis, IN. Wiley Publishing Inc., 2010, pp. *1–177.*

[2] Enric Galceran , Marc Carreras. (2013, Feb.). A survey on coverage path planning for robotics. Robotics and Autonomous Systems 61 (2013) 1258–1276]. Available: http://www.journals.elsevier.com/robotics-and-autonomous-systems

[3] CDR H.R. Everett. (1989, May.). A survey on collision avoidance and range sensors for robotics. Robotics and Autonomous Systems Vol. 5 Issue 1. [Online]. pp. 5–67. Available:http://www.sciencedirect.com/science/article/pii/0921889089900419

[4] OpenCV Documentation [Online]. Available site: http://docs.opencv.org/2.4/doc/tutorials/imgproc/table_of_content_imgproc/table_of_content_imgproc.html