An Implementation on Object Move Detection Using OpenCV

Professor: Dr. Ali Arya
Reported by: Farzin Farhadi-Niaki
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada

I. INTRODUCTION

Computer vision is a rapidly growing field, partly as a result of both cheaper and more capable cameras, partly because of affordable processing power, and partly because vision algorithms are starting to mature. OpenCV itself has 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 and jump-start research by providing them with a computer vision and machine learning infrastructure that was previously available only in a few mature research labs.

II. METHODOLOGY

Computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal. The input data may include some contextual information such as "the camera is mounted in a car" or "laser range finder indicates an object is 1 meter away". The decision might be "there is a person in this scene" or "there are 14 tumor cells on this slide". A new representation might mean turning a color image into a grayscale image or removing camera motion from an image sequence.

For robotics, we need object recognition (what) and object location (where):

a) Object recognition

In OpenCV, there are a wide range of approaches to detect an object; the techniques such as: convolution/filters, thresholds, histogram and matching, contours, efficient nearest neighbour matching to recognize objects using huge learned databases of objects, etc.

b) Object location

Again, in OpenCV, there are various techniques for finding the object location, e.g. background subtraction (to find the moving objects), corner finding, optical flow, mean-shift and camshaft tracking, structure from motion using SIFT descriptor and SURF gradient histogram grids, or just simply finding the edge of the object and momentarily checking the location of object’s centre on pixel by pixel based movement.

A. Convolution

Convolution is the basis of many of the transformations. In the abstract, this term means something we do to every part of an image. What a particular convolution "does" is determined by the form of the Convolution kernel being used. This kernel is essentially just a fixed size array of numerical coefficients along with an anchor point in that array, which is typically located at the center. The size of the array is called the support of the kernel.

We can express this procedure in the form of an equation. If we define the image to be $I(x, y)$, the kernel to be $G(i, j)$ (where $0 < i < M_i - 1$ and $0 < j < M_j - 1$), and the anchor point to be located at $(a_i, a_j)$ in the coordinates of the kernel, then the convolution $H(x, y)$ is defined by the following expression:

$$H(x, y) = \sum_{i=0}^{M_i-1} \sum_{j=0}^{M_j-1} I(x+a_i, y+a_j)G(i, j)$$

B. Canny

The most significant new dimension to the Canny algorithm is that it tries to assemble the individual edge candidate pixels into contours. These contours are formed by applying a hysteresis threshold to the pixels. This means that there are two thresholds, an upper and a lower. If a pixel has a gradient larger than the upper threshold, then it is accepted as an edge pixel; if a pixel is below the lower threshold, it is rejected. If the pixel's gradient is between the thresholds, then it will be accepted only if it is connected to a pixel that is above the high threshold.
void cvCanny(
    const CvArr* img,
    CvArr* edges,
    double lowThresh,
    double highThresh,
    int apertureSize = 3
);

The cvCanny() function expects an input image, which must be grayscale, and an output image, which must also be grayscale.

C. Threshold

double cvThreshold(
    CvArr* src,
    CvArr* dst,
    double threshold,
    double max_value,
    int threshold_type
);

Frequently we have done many layers of processing steps and want either to make a final decision about the pixels in an image or to categorically reject those pixels below or above some value while keeping the others. The OpenCV function cvThreshold() accomplishes these tasks. The basic idea is that an array is given, along with a threshold, and then something happens to every element of the array depending on whether it is below or above the threshold. The cvThreshold() function handles only 8-bit or floating-point grayscale source images.

D. Adaptive Threshold

There is a modified threshold technique in which the threshold level is itself variable. In OpenCV, this method is implemented in the cvAdaptiveThreshold() function:

void cvAdaptiveThreshold(
    CvArr* src,
    CvArr* dst,
    double max_val,
    int adaptive_method = CV_ADAPTIVE_THRESH_MEAN_C,
    int threshold_type = CV_THRESH_BINARY,
    int block_size = 3,
    double param1 = 5
);
E. Contours

Although algorithms like the Canny edge detector can be used to find the edge pixels that separate different segments in an image, they do not tell us anything about those edges as entities in themselves. The next step is to be able to assemble those edge pixels into contours. \texttt{cvFindContours()} is a convenient function in OpenCV that will do exactly this for us. Specifically, with assigning memory storages, OpenCV functions gain access to memory when they need to construct new objects dynamically; then we will need to use sequences (as something similar to the generic container classes), which are the objects used to represent contours generally. With those concepts in hand, we will get into contour finding.

A contour is a list of points that represent a curve in an image. Contours are represented in OpenCV by sequences in which every entry in the sequence encodes information about the location of the next point on the curve.

The function \texttt{cvFindContours()} computes contours from binary images. It can take images created by \texttt{cvCanny()}, which have edge pixels in them, or images created by functions like \texttt{cvThreshold()} or \texttt{cvAdaptiveThreshold()}, in which the edges are implicit as boundaries between positive and negative regions. Drawing a contour on the screen using \texttt{cvDrawContours()} function is the next step.

Here we create a window with an image in it. A trackbar sets a simple threshold, and the contours in the thresholded image are drawn. The image is updated whenever the trackbar is adjusted.

III. RESULTS AND DISCUSSIONS

In this project, using OpenCV, we have tried to implement a simulation of object move detection (Fig. 3). The procedure of requirements for this implementation has the following steps:

1) Capturing the image as a frame from a camera.
2) Converting the image/frame to 8-bits grayscale image to be compatibly usable in some filters like Canny.
3) Creating trackbars for up and down thresholds (Fig. 4).
4) Finding the edge of the object using Canny or Threshold (Adaptive Threshold) functions (Fig. 5).
5) Creating dynamic structure and sequence for the contours, and finding the current location (x, y) of the object’s centre after finding its contours.
6) Finally, drawing and detecting the object’s move, in two ways:
a) Detecting the average move and direction (one of four: left, right, up, down), after the object passes a minimum distance threshold and comes back to the threshold area.

b) Detecting the highest move and direction (one of four: left, right, up, down), after the object passes a maximum distance threshold.
IV. CONCLUSION

Computer vision applications are growing rapidly, from product inspection to image and video indexing on the Web to medical applications and even to local navigation on Mars. OpenCV is also growing to accommodate these developments.

One of the key new development areas for OpenCV is robotic perception. This effort focuses on 3D perception as well as 2D plus 3D object recognition since the combination of data types makes for better features for use in object detection, segmentation and recognition. Robotic perception relies heavily on 3D sensing, so efforts are under way to extend camera calibration, rectification and correspondence to multiple cameras and to camera plus laser rangefinder combinations.

Creating capable robots subsumes most fields of computer vision and artificial intelligence, from accurate 3D reconstruction to tracking, identifying humans, object recognition, and image stitching and on to learning, control, planning, and decision making. Any higher-level task, such as planning, is made much easier by rapid and accurate depth perception and recognition. It is in these areas especially that OpenCV hopes to enable rapid advance by encouraging many groups to contribute and use ever better methods to solve the difficult problems of real-world perception, recognition, and learning.

OpenCV will, of course, support many other areas as well, from image and movie indexing on the web to security systems and medical analysis. The wishes of the general community will heavily influence OpenCV’s direction and growth.

There is a worldwide community of interactive artists who use OpenCV so that viewers can interact with their art in dynamic ways. The most commonly used routines for this application are face detection, optical flow, and tracking. The focused effort on improving object recognition will allow different modes of interacting with art, because objects can then be used as modal controls. With the ability to capture 3D meshes, it may also be possible to “import” the viewer into the art and so allow the artist to gain a better feel for recognizing user action; this, in turn, could be used to enhance dynamic interaction. The needs and desires of the artistic community for using computer vision will receive enhanced priority in OpenCV’s future.

A group of manufacturers are aiming to develop cell-phone projectors—perfect for robots, because most cell phones are lightweight, low-energy devices whose circuits already include an embedded camera. This opens the way for close-range portable structured light and thereby accurate depth maps, which are just what we need for robot manipulation and 3D object scanning.

Computer vision has a rich future ahead, and it seems likely to be one of the key enabling technologies for the 21st century. Likewise, OpenCV seems likely to be (at least in part) one of the key enabling technologies for computer vision. Endless opportunities for creativity and profound contribution lie ahead.

REFERENCES