Object tracking with trajectory approximation using polynominal fitting

Heikki Saul, Taavi Adamson
Institute of Technology
University of Tartu
Tartu, Estonia

***Abstract*— In today's world of automation and autonomy, accurate detection, tracking and prediction of objects is extremely important to maintain safety or to allow robots to work independently. The application covered in this paper is the basis of these more advanced applications.**

Keywords—digital image processing, OpenCV, curve fit, contour detection

I. Introduction

Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. [1] Detection of moving objects and motion-based tracking are important components of many computer vision applications, including activity recognition, traffic monitoring, and automotive safety. [2]

The implementation covered in this paper handles two important problems in object tracking - tracking a visible object and approximating its location if it becomes obscured. This implementation can be used a basis for more advanced object tracking and location prediction applications.

This paper describes the theoretical background and methods used for a solution implementation that enables a computer to detect and track a given object on digital video. The application was programmed using Python programming language and using widely-known modules of OpenCV and numpy.

II. Overview

A. Frame processing

Digital video from camera is processed frame by frame. To detect object on a frame, the image is first thresholded within HSV color space. Image thresholding is a simple, yet effective, way of partitioning an image into a foreground and background. This image analysis technique is a type of image segmentation that isolates objects by converting images into binary images. [3].

The resulting binary image is used to detect its contours using OpenCV *findContours* function and the accompanying contour handling methods. This was found to be the most reliable way of detecting a blob in the binary image. OpenCV's *SimpleBlobDetector* was tested, but found to be unsatisfying. From the contour info, the blob's center frame-relative coordinates are calculated and passed on to trajectory calculation.

B. Trajectory calculation and visualization

The trajectory calculation uses given coordinates for trajectory approximation and passes the result onto visualization. The mode of operation depends on whether the object is currently visible or not. Visualization shows the objects current location and the history of its previous locations.

III. Thresholding

A. Color space

The application uses HSV color space for increased robustness. The HSV stands for hue, saturation and value.

- The hue (H) of a color refers to which pure color it resembles. All tints, tones and shades of red have the same hue.

  Hues are described by a number that specifies the position of the corresponding pure color on the color wheel, as a fraction between 0 and 1. Value 0 refers to red; 1/6 is yellow; 1/3 is green; and so forth around the color wheel.

- The saturation (S) of a color describes how white the color is. A pure red is fully saturated, with a saturation of 1; tints of red have saturations less than 1; and white has a saturation of 0.

- The value (V) of a color, also called its lightness, describes how dark the color is. A value of 0 is black, with increasing lightness moving away from black. HSV allows for thresholding that is less dependent on the white balance and uniformity of lighting. That allows the application to be used in a variety of environments with less calibration. [4]

B. Processing

Since OpenCV BGR-to-HSV conversion introduces a considerable amount of noise, the image is first blurred to have a smoother result. The resultant loss of accuracy is insignificant to the desired operation of the application. After blurring, the video frame is converted to HSV color space and then thresholded using the OpenCV *inRange* function.

Thresholding of a camera frame (example in figure 1) results in a binary image that is used as a source to detect the object. One example of a binary image can be seen in figure 2.

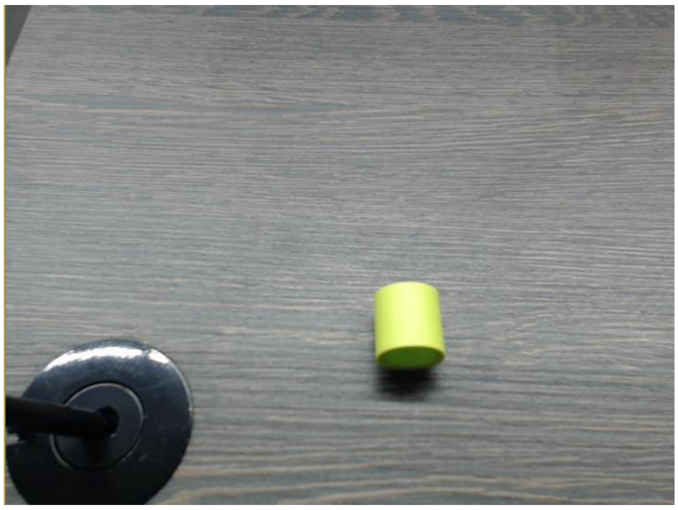
Figure 1: Frame with an example object

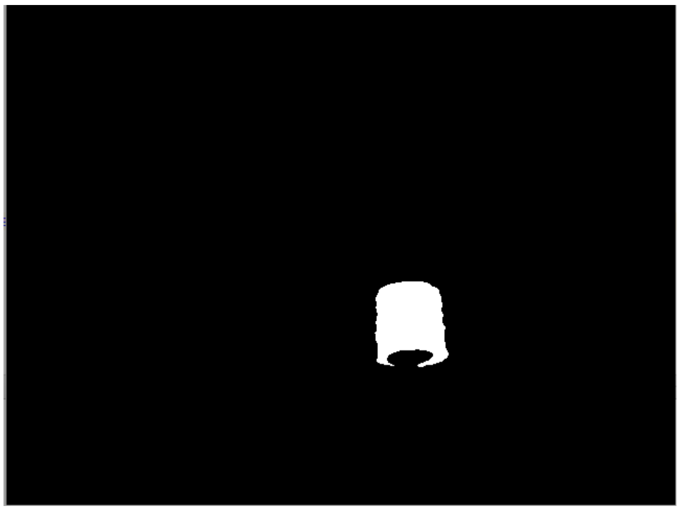
Figure 2: Thresholded binary image

IV. Binary image processing

A. Contour detection

To provide the visualization and curve fitting subprocesses with necessary data about the size and location of the tracked image, OpenCV *findContours* function is used. *findContours* uses an algorithm to retrieve contours from binary images. In the covered application, the retrieval mode RETR_EXTERNAL is used to locate only the outermost contours. To simplify data handling, CHAIN_APPROX_SIMPLE contour approximation is used, as it compresses vertical, horizontal and diagonal segments, outputting only their end points instead of all data retrieved data points. [5] The biggest contour by area is used in the following centroid and area calculations.

B. Centroid detection and area filtering

Only the biggest detected contour is used for centroid and area calculation. Blurring and proper thresholding boundaries guarantee that the biggest detected contour is indeed the object wished to be tracked.

OpenCV moments function is applied on the contour to give a list of necessary moments that are used for centroid detection and area filtering.
The necessary moments are:

- Area of object in binary image: M_{00}
- Centroid {x, y} = {M_{10}/M_{00}, M_{01}/M_{00}}

The application uses the area of the contour to filter out situations, where the only a part of the object is visible and the centroid might give rapidly changing false values. The centroid is passed to the curve fitting calculator that handles location approximation. If the centroid value is present, the calculator passes the value to the visualization subprocess. If the object is no longer visible, the calculator approximates the object's possible location based on previous location values. In this case, the calculated values are passed to the visualization subprocess. [6]

V. Trajectory calculation

A. Curve fitting

The developed solution can approximate the object’s location in case it leaves the camera’s field of view. This is done by retaining a history of previous coordinates and timestamps and calculating its path by fitting curve of polynomial of third degree to the data and using it to calculate next possible location of the object. The curve fitting is done by using a function from Python numpy module called *polyfit*. This function returns polynomial coefficients of best fitting curve. This is achieved using the least squares method where the error to be minimized is calculated by equation 1. [7]

$$E = \sum_{j=0}^{k} \left| p(x_j) - y_j \right|^2 \quad (1)$$

The x and y coordinates of the object are considered as independent variables and therefore two curves are calculated: one for x coordinate and one for the y coordinate. Both use current time in milliseconds as the second parameter for curve fitting. The timestamps correspond to capturing of the frame from which the x and y coordinates were calculated from.

B. Operation logic

In case the object is detected on the screen and its center coordinate is known the calculation part of the solution simply outputs its current coordinate and saves it to its internal history. For optimization reasons the polynomial equation describing best fitting curve of current coordinate history is calculated only at the moment the object leaves. Until the object returns to the cameras field of view the object is predicted to be moving per the calculated equation and the corresponding coordinate is put to the output. As soon as the object returns to the frame the coordinate history is cleaned and normal operation continues.

VI. Visualization

The final output of the application is video feed where the moving center of tracked object is marked by a small red dot. The dot is drawn for every processed frame. The number of dots retained in history can be specified within the application. This is used to visualize the currently known and proposed trajectory of the object. Example output can be seen in figure 3.

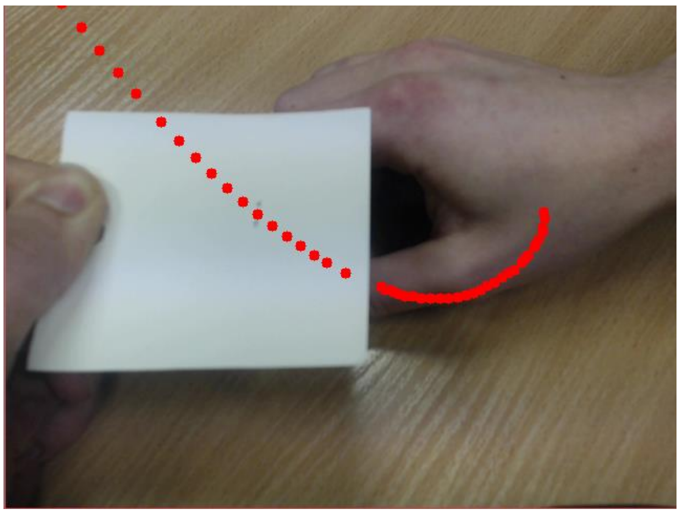

Figure 3: Example of object trajectory visualization

VII. Conclusion

The solution was developed to illustrate one possible method of tracking an object, observing its current trajectory and using it to calculate the approximation of its future trajectory.

Image processing was done using various Python OpenCV methods. The basis of object recognition was contour detection. A third-degree polynomial curve fitting was used for trajectory estimation.

VIII. References

[1] Peter Mountney, Danail Stoyanov & Guang-Zhong Yang (2010). "Three-Dimensional Tissue Deformation Recovery and Tracking: Introducing techniques based on laparoscopic or endoscopic images." IEEE Signal Processing Magazine. 2010 July. Volume: 27". IEEE Signal Processing Magazine. 27 (4): 14–24. doi:10.1109/MSP.2010.936728

[2] MathWorks https://se.mathworks.com/help/vision/examples/motion-based-multiple-object-tracking.html

[3] MathWorks https://se.mathworks.com/discovery/image-thresholding.html

[4] Introduction to color theory http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html

[5] OpenCV documentation - findContours http://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a

[6] Wikipedia - Image moment https://en.wikipedia.org/wiki/Image_moment

[7] numpy.polyfit documentation https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html