A Robust Real Time Marker-Tracker for Embedded Applications

Ankush Gola
agola@princeton.edu

David Fridovich-Keil
dfridovi@princeton.edu

Abstract

In this project, we demonstrate a robust, efficient marker tracking and recognition system for use in embedded applications like autonomous vehicle navigation, surveillance, and aerial drone alignment. Our pipeline consists of (1) pre-processing the image using color filtering to segment out the background, (2) computing ORB keypoints in the scene and template, (3) doing keypoint matching and homography estimation between the template and the scene, (4) validating the bounding box to prevent false positives, (5) using a single-tap IIR (infinite impulse response) filter to smooth the bounding box's movement from one frame to the next, and (6) using each frame's bounding box to update the template and to restrict the keypoint search in the next frame. This last element is key, as our goal is real-time operation on an embedded system, and reducing the domain of keypoint matching reduces the computational expense of our algorithm.

1. Motivation

The problem we wish to solve is, given an image with a marker of interest, we would like to detect the location of the marker (defined by its bounding box) in the image. We are interested in detecting a small identifying marker in a larger scene for several reasons. For example, our system might be used to align two autonomous aerial drones so that they can identify one another and fly in formation. Additionally, we could use this system to recognize landmarks in a scene and use them as guides for autonomous vehicle localization and navigation. These particular applications are relevant to both of our senior theses. This system might also be used for surveillance applications.

For these types of end applications, our system must be robust against object occlusion, rotation, scaling, and inconsistencies in lighting. We would also like to detect the marker even if it is not moving. Furthermore, we require that our bounding box scale with the template – if the marker is far away, our algorithm must return a smaller bounding box than if the marker is nearby. This way, the scale of the bounding box could be used as input to a control system for a zoom camera, telling the camera when to zoom in and out. We also require that the algorithm be robust against false positives; we would prefer for it to fail to detect a marker that is present than for it to return a valid bounding box for an image that does not actually contain the marker. Finally, the tracking algorithm must be efficient enough to process video in real time on an embedded processor. As far as we can tell, no previous tracking algorithms satisfy all of our requirements.

2. Related Work

Some popular object tracking and detection methods and concepts include mean shift, HOG-SVM, and feature matching.

The mean shift algorithm [11] attempts to draw a bounding box around an area of an image with maximum pixel intensity. The image must be filtered in such a way to highlight the pixels of the object to track. Mean shift is an iterative algorithm that shifts the bounding window until the weighted mean of the pixel intensity is in the center of the window. For our purposes, mean shift will not suffice. The main drawback is that the size of the bounding box does not scale with respect to the size of the object we are tracking. Since the eventual goal is to use this algorithm with a pan/tilt/zoom camera system, knowing the scale of the bounding box could give us information about how to zoom the camera. Mean shift is also not robust to occlusions or severe changes in lighting.

The ensemble of exemplar SVMs object detector proposed by Malisiewicz et al [6] works by training a separate SVM on different instances of an object and then using a sliding window detector to calculate a response strength. The algorithm is robust but is unnecessarily complex and inefficient for our purposes. We are trying to detect a specific object, not a category of objects, and thus we suspect that SVM or another machine learning based classifier is unnecessary.

Feature based matching for object detection involves detecting correspondences between a template and scene. Our algorithm will be adapted from feature based matching because the method is robust to changes in scale, rotations, and small occlusions. Most importantly, the method can account for changes in perspective that occur as the angle between the camera and the marker changes over time.

From what we could find, an object tracking system similar to ours was built by Doyle, Jennings, and Black in 2013. Their pipeline consisted (1) get the current frame, store the previous one, (2) SURF feature detection (3) calculate optical flow from previous frame to current frame, (4) subtract estimated background movement, (5) threshold matching, (6) centroid detection and prediction with Kalman filtering. They were able to detect an object (toy helicopter) moving 1.2 m/s at 1 m distance, traveling 90 pixels at a system rate of 15 fps. To implement their system, they used OpenCV running on a Dell M6600 Precision laptop [9].

Our pipeline utilizes steps similar to those used by Doyle, Jennings, and Black, but simplifies and optimizes parts of the process. For example, as mentioned later, we use ORB features as a faster alternative to SURF, and we use a simple IIR (infinite impulse response) filter for prediction. Additionally, we do not incorporate optical flow into our system as it is adds unnecessary complexity. We used cropped frames and templates in which, ideally, there exists no optical flow.

3. Algorithm Overview

Our project relies on four separate algorithms that operate in a closed loop. The first step is to pre-process each video frame (as well as the marker template) to remove as much background clutter as possible. This is not strictly necessary, but any reduction in the complexity of the problem can only improve system performance. Specifically, our approach here is to choose a brightly colored orange marker, and then filter out only the orange part of the color spectrum in each frame. We do this by converting the frame to the hue-saturation-value (HSV) representation and thresholding the value, or brightness, array according to the hue array. To rectify any small discontinuities in the binarization, we then run a 3x3 median filter.

Then, we extract robust keypoints from this pre-processed image (as well as from the template). There has been much prior research on this topic, and in general the SIFT (Scale-Invariant Feature Transform) [1] is accepted as one of the most robust algorithms available. SIFT works, broadly, by converting each image patch to a canonical representation and computing a feature vector from that canonical form. The algorithm is invariant to both scaling and rotation, unlike the simpler (and more efficient) Harris corner detector [2]. The Harris corner detector looks for the intersection of strong edges, which is rotationally invariant, but unfortunately not scale invariant (and hence is not suitable for our project). However, SIFT detection is rather computationally intensive, so we elected to replace SIFT with the more efficient ORB (oriented FAST and rotated BRIEF) detector [5]. ORB achieves computational efficiency by computing rotationally invariant versions of FAST (features from accelerated segment test) [7] and BRIEF (binary robust independent elementary features) [8], two other keypoint detectors/descriptors that are already quite efficient.

Third, we match keypoints between the template of our marker and the image. We use Lowe's ratio test [1] to filter out ”good” nearest-neighbor matches in feature space, and then estimate a transformation matrix using RANSAC [3], a randomized, iterative algorithm to estimate parameters from observed data (in our case, keypoint matches) containing outliers. From the transformation matrix (a homography) we map the corners of the template to the image plane to obtain a bounding box (not necessarily a rectangle) for the marker in the frame.

Last, given a sequence of locations of the marker in the image frame, it should be possible to extrapolate the next likely location. Mathematically, the formalization of this idea is the Kalman filter [4], a probabilistic state estimator that uses a series of measurements over time in order to estimate the values of unknown variables. However, since all we require is a very rough estimate of where to crop the next frame, we need not resort to the mathematical and computational complexity of a Kalman filter. Instead, we use a simple, single-tap recursive IIR filter. This is an especially simple case of generalized time-domain signal processing: if $y[n]$ is the estimate of the corner position at time n, and $x[n]$ is the calculated position at time n, then we calculate

$$y[n] = \alpha y[n-1] + (1-\alpha)x[n]$$

for some constant parameter α. The frequency response of this filter is plotted below in figure 1: note that the peak at zero frequency is sharper for higher α, and the asymptotic lower bound is smaller. This means that the filter attenuates high frequencies more strongly for high α. Empirically, we found that setting $\alpha = 0.2$ strikes a good balance between smoothing the results and allowing the bounding box to evolve quickly enough.

We use this mechanism to low-pass filter the locations (x- and y-coordinates) of the corners of the marker's bounding box from frame to frame, and in each frame to crop the region of interest around the filtered location from the previous frame. As above, this reduces the effective resolution and search space of our images, which correspondingly reduces the computational complexity of our problem significantly.

4. Implementation

We implemented our system in the Python language, and utilized the freely-available OpenCV library[10], taking advantage of its many real-time, highly optimized image processing and computer vision functions. Our entire algorithm is fairly compact, taking roughly 450 lines of code.

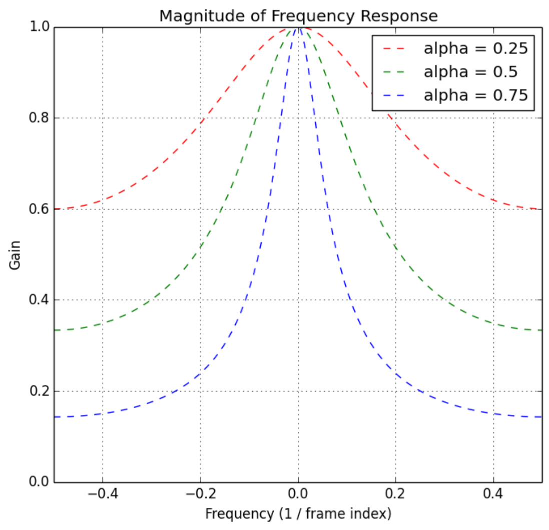

Figure 1. Absolute value of IIR filter frequency response, plotted at different α levels.

Due to time constraints, we were not able to port our code to an embedded platform such as a Raspberry Pi; instead, we developed on a Macbook Pro (early 2011), with an Intel i7 2 GHz processor and 8 GB of RAM.

The following pseudocode provides a detailed overview of the high-level pipeline described in broad terms above. Note that we made several design decisions that are reflected in the code. First, we have fairly strict conditions for what constitutes a valid box (convex hull consists of exactly four points; top- and bottom-left corners are left of the centroid, top- and bottom-right corners are right of the centroid, top corners above centroid, and bottom corners below centroid; bounded area in each box and only small changes between boxes). Second, we treat all errors equivalently: whenever the system cannot find a good bounding box (for whatever reason), it does nothing except reset the cropping parameters so as not to crop the following frame. This means that the system is fairly robust to momentary lapses; however, if the bounding box becomes worse and worse then the template also degrades, which means that in some cases the algorithm never recovers.

Regarding hardware, as mentioned, we developed and tested our algorithm on a laptop running Mac OS X (Yosemite), with an Intel i7 processor and 8 GB of RAM, rather than an embedded system like a Raspberry Pi. We also used the built-in video camera, which has a relatively low resolution (1280x1024 pixels). As discussed in the results section below, this imposed a very hard limit on the distance at which we could detect the marker reliably (since the camera does not support optical zoom). In future implementations (used in our theses), we will use a higher quality camera, with better resolution as well as optical zoom. The higher resolution will allow us greater flexibility in cropping the object (while still maintaining adequate resolution in the cropped section to detect ORB keypoints), and the optical zoom (which we will adjust using closed-loop PID (proportional, integral, derivative) control aimed at maintaining a constant resolution of the cropped section) will give us the ability to detect markers robustly at an even greater distance.

Algorithm 1 Marker-tracking

```
$template \leftarrow$ imread(TEMPLATEFILE)
$template \leftarrow$ filterColor($template$)
$kp\_temp, des\_temp \leftarrow$ getORB($template$)
$corners\_temp \leftarrow$ corners of $template$
$cropParams \leftarrow$ None
while True do
  $frame \leftarrow$ getCameraFrame()
  if $\exists$ $crop\_params$ then
    $frame \leftarrow$ cropFrame($crop\_params$)
  end if
  $frame \leftarrow$ filterColor($frame$)
  $kp\_frame, des\_frame \leftarrow$ getORB($frame$)
  $matches \leftarrow$ nearestNeighbors($des\_frame, des\_temp$)

  $good\_matches \leftarrow$ ratioTest($matches$)
  if not enough good matches then
    $cropParams \leftarrow$ None
  else
    $H \leftarrow$ estimateHomography($good\_matches$)
    $corners\_frame \leftarrow$ transform($corners\_temp, H$)
    if not isValidBox($corners\_frame$) then
      $cropParams \leftarrow$ None
    else
      $corners\_frame \leftarrow$ filterIIR($corners\_frame$)
      drawOutput($frame, corners\_frame$)
      $crop\_params \leftarrow$ getBox($corners\_frame$)
      $corners\_temp \leftarrow corners\_frame$
      $kp\_temp, des\_temp \leftarrow kp\_frame, des\_frame$
    end if
  end if
end while
```

We provide a link to our project's GitHub repository in the appendix to this paper, which includes all of our code, as well as several sample videos and marker templates.

5. Results

In this section we present results from each stage of development, as well as links to videos indicative of our final results.

Initially, we intended to use a (black and white) QR code as the marker. However, we found that keypoint matching was very inaccurate, as shown in figure 2. As shown below, many keypoints in the template match to very nearly the same point in the test frame. We hypothesize that this happens because, essentially, all corners in a QR code look the same in the context of their local neighborhood. Since ORB only looks at the neighborhood around each keypoint to compute its descriptor, it maps all the corners to the very similar feature vectors. For this reason, we replaced the QR code with a less regular pattern – orange and black zebra stripes (see figure 3). Combined with color filtering, this dramatically improved our matching.

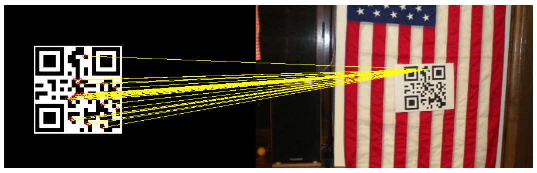
Figure 2. Example of failed QR code detection due to the similar appearance of each corner.

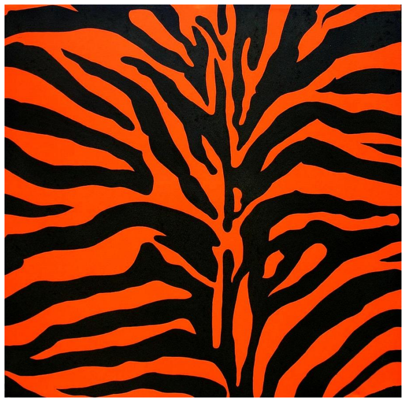
Figure 3. Our orange zebra template.

As we developed our algorithm, we found it crucial to establish a strict set of conditions for what makes a valid bounding box. Without sufficient (or any) conditions of this sort, the quality of the bounding box easily degrades over time, and since we crop each frame around the prior frame's bounding box, as the bounding box becomes less reliable, the cropping begins to remove important parts of the image, and it becomes impossible for the algorithm to ever recover. Bounding box validation dramatically lowered our rate of false positives. Figure 4 below is an example of a poor quality bounding box that can occur without adequate error checking. The box shown would not pass either our centroid test or our convex hull test. By contrast, the bounding box shown in Figure 5 passes all our valid box conditions.

Finally, we recorded two trial videos using the laptop's built-in webcam, and ran our algorithm with and without each major component (IIR corner filtering, color filtering, and cropping), and recorded the results in Table 1. We calculate the raw runtime as the total amount of time spent processing the video, not counting startup time or the time

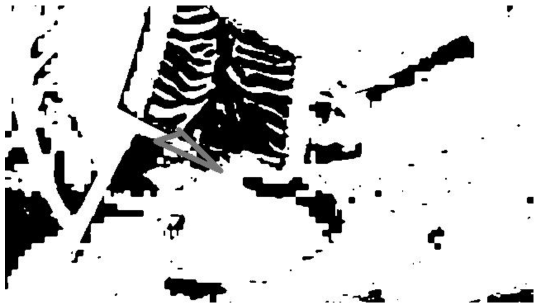
Figure 4. False positive detected as a result of poor bounding box validation.

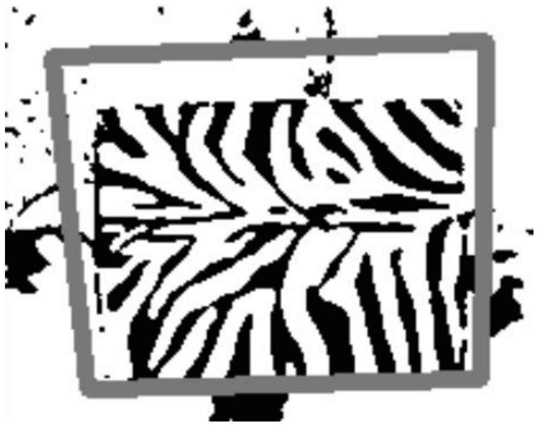
Figure 5. Example of a valid bounding box.

Table 1. Timing results

Version	Trial	Adjusted Runtime	Accuracy
full	1	37.1	0.39
full	2	23.9	0.57
no IIR	1	42.9	0.31
no IIR	2	111.1	0.14
no HSV	1	25.8	0.38
no HSV	2	44.8	0.25
no crop	1	35.1	0.42
no crop	2	39.6	0.44

that it takes to save each frame (since that would not be part of our final version designed for an embedded system). However, most of the computation time occurs for frames in which there are a large number of valid keypoint matches that then (very often) end up passing our valid box conditions. So, we adjust the measured runtime, dividing it by the fraction of time during each video that the algorithm finds a valid bounding box (we call this ratio ”accuracy”).

Links to each video are provided in the appendix at the end of this paper.

6. Evaluation

From these results, we see that full algorithm achieves nearly the highest accuracy in trial 1, and by far the highest accuracy in trial 2 – as we expect, given that it includes all elements of the pipeline. The full version also achieves among the lowest adjusted runtime for each trial. This makes sense because the versions without IIR corner and HSV color filtering each have one fewer computational task per frame, making them relatively fast (indeed, the version without color filtering has the fastest adjusted runtime in the first trial). Meanwhile, the version without cropping must filter larger images and estimate a homography from far more keypoints, which makes it run slower.

However, our metric of "adjusted runtime" is only an approximate measure of computational efficiency, and should not be misinterpreted as a direct measure of speed. For example, the version without IIR corner filtering appears to run quite slowly on the second trial, with an adjusted runtime of 111.1 seconds. But this can be explained by its incredibly low accuracy for that particular trial (14%). This means that it skips far more frames than any other version, and thus a more significant portion of its total runtime is taken up processing invalid bounding boxes. This in turn inflates the adjusted runtime, which assumes that the vast majority of the raw runtime is due to processing valid bounding boxes.

In addition to the data in the table above, it is clear from watching the videos that the full algorithm outperforms any other version on tests such as: detecting a rapidly moving marker, detecting the marker at a distance, detecting a rotating marker, detecting a partially occluded marker, and not detecting objects with the same colors as the marker.

7. Analysis

For the reasons discussed above, it is difficult to analyze and quantify the success of our algorithm in concrete terms. It is clear, though, that we have achieved our main goals: efficient (real-time) marker tracking, with variably-sized bounding boxes, and robust to rotation, scale, and false positives. We did no not able to test our system on an embedded processor, but we are confident that our algorithm could be adapted to run in that environment without very much trouble. In particular, OpenCV can be compiled for the GPU in some devices (like the NVidia Jetson), drastically improving performance beyond even the capabilities of a laptop CPU.

Still, there are several other ways in which our algorithm and implementation might be improved. One of the features of our algorithm is that it automatically crops the region around the marker, effectively zooming in on the relevant region of the image. As described above, this reduces the size of the frame and hence the scale of our problem and the algorithm's runtime. It also improves the reliability of keypoint matching by removing most of the background and reducing the domain of keypoint matching to the region immediately surrounding the marker. Unfortunately, these benefits come at the cost of producing a highly pixelated image when the marker is far from the camera. Indeed, we were forced to restrict the size of valid bounding boxes so that they do not become so small as to introduce noise into the keypoint matching process (Lowe's ratio test is not very robust for dense keypoints). This lower bound on the size of the bounding box limits the distance at which we can detect the marker to roughly 7 feet. In future implementations of this algorithm, we plan to solve this problem by using a higher quality camera, with higher resolution as well as optical zoom.

Another way we might improve overall performance is by improving our color filtering. Currently, the filter is somewhat noisy; we believe this is due both to the imprecision of RGB to HSV conversion (especially in the hue dimension), as well as to changes in lighting. If we could remove more of the noise from this filter – for example, by blurring the image before converting to HSV, or by retuning our filter parameters so as to make the binarization more stable – that could make the rest of the pipeline more reliable. For example, without so much noise, there would be a fewer noise speckles that get inadvertently chosen as keypoints (matches involving such keypoints are often meaningless since the noise appears to be random and uncorrelated between frames).

8. Individual Contributions

We worked together very closely on this project. Apart from separately drafting initial versions of our algorithm in MATLAB (done by Ankush) and Python (done by David), we were both equally involved with all other elements of this projects, including implementing our algorithm, devising a strict set of bounding box conditions, debugging, and testing.

References

[1] David G. Lowe, Object Recognition from Local Scale-Invariant Features. IEEE International Conference on Computer Vision, 1999.

[2] Chris Harris and Mike Stephens, A Combined Corner and Edge Detector. Alvey Vision Conference, 1988.

[3] Martin A. Fischler and Robert C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. ACM Graphics and Image Processing, June 1981.

[4] Rudolph E. Kalman, A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME–Journal of Basic Engineering, 1960.

[5] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski, ORB: an efficient alternative to SIFT or

SURF. IEEE International Conference on Computer Vision, 2011.

[6] Tomasz Malisiewicz, Abhinav Gupta, Alexei A. Efros, Ensemble of Exemplar-SVMs for Object Detection and Beyond, IEEE International Conference on Computer Vision, 2011.

[7] Edward Rosten and Tom Drummond, Fusing Points and Lines for High Performance Tracking. IEEE International Conference on Computer Vision, 2005.

[8] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, BRIEF: Binary Robust Independent Elementary Features. European Conference on Computer Vision Lecture Notes in Computer Science, 2010.

[9] Daniel D. Doyle, Alan L. Jennings, and Jonathan T. Black, Optical flow background estimation for real-time pan/tilt camera object tracking Journal of the International Measurement Confederation, 2014.

[10] Gary Bradski The OpenCV Library, http://opencv.org, 2000.

[11] Dorin Comaniciu and Peter Meer, Mean Shift Analysis and Applications. IEEE Conference on Computer Vision, 1999.

Appendix

The GitHub repository for our project is hosted at https://github.com/agola11/qr_tracker and is freely downloadable. Our main implementation can be found at qrTrackerPython/track_qr_eval.py, and the IIR filter class is located in the file qrTrackerPython/filter2d.py. Several raw and processed videos can be found in the videos/ directory, and in particular the videos discussed in the results are discussed in the videos/benchmark/ directory. Note: the entire project is quite large (approximately 1 GB), since it includes so many videos and images.

We have also uploaded the eight videos for which we report data in the results section. They can be found at the following YouTube links.
Full, run 1: www.youtube.com/watch?v=dgAC-PNq4GI
Full, run 2: www.youtube.com/watch?v=jEjFIkEJ4Dc
No IIR, run 1: www.youtube.com/watch?v=3aT2qYhEK28
No IIR, run 2: www.youtube.com/watch?v=IbCD22XNJPo
No HSV, run 1: www.youtube.com/watch?v=v1Pwz4lYNLY
No HSV, run 2: www.youtube.com/watch?v=n8r8fyjSL0c
No crop, run 1: www.youtube.com/watch?v=lO8LiI33lio
No crop, run 2: www.youtube.com/watch?v=mwBryXAGId4