A More Robust Approach to Home Security: Object Detection for Heightened Surveillance in Sentry, an iOS App

Charles Jacobson, COS '16 BSE
Princeton University
cmj3@princeton.edu

Mckervin Ceme, COS '16 AB
Princeton University
mceme@princeton.edu

Abstract

Advancements in home security have thus far been unable to take advantage of object-detection algorithms for video monitoring. In this project, we aimed to utilize object-detection in order to improve the usefulness of home security monitoring systems. The goal of having security systems is to protect personal belongings. However, many security solutions that take advantage of object detection do not produce actionable data that is useful to the client. Rather, many of the current home security solutions notify their users of false alarms do to inherent weaknesses in their approaches. We aimed to produce data that could be more definitive in assessing whether or not a user's belonging was in fact stolen, or merely if there was an obstruction in the camera view-angle. To achieve this, we looked at the ratios between the number of keypoint matches in frames with the object in place versus frames with the object removed by creating an iOS app called Sentry.

1. DOWNLOAD CODE

To download our code, head on over to `https://github.com/charliejl28/sentry`. Then, simply download the zip file using the link on the right. This is the code for the iOS app for both Mckervin Ceme and Charlie Jacobson.

2. Introduction

The field of home security has seen a dramatic rise in growth thanks to recent technological advancements, helping to make these types of security solutions more common among homeowners. Currently, approximately one in three homeowners have some sort of smart home security device enabled in their home. Given this, the applications for smarter home security has seen tremendous growth. However, there are not many computer-vision related applications to improving home security. Specifically, there has been a lack in securing valuable possessions when the homeowner is not present in the house. A valuable item can often be stolen from the home without tripping detectors. Furthermore, other home security devices are so *sensitive* to the smallest change in the environment that users do not heed caution when an actual burglary may take place. Therefore, in order to improve smart home camera usability we propose that integrating object detection into an iOS application that monitors valuable objects, we will help in reducing the rate of unnoticed house thefts. We see our application being useful for home settings in addition to the level of security that object detection can add to a video surveillance network in many commercial settings.

2.1. Object Detection

The object-detection framework supports the basis for our application. Rather than process an image for distinct shapes or the total number of objects in a given frame, we process multiple frames in a video sequence to confirm the location of an item tagged by the application user. Currently, there are moving object detection applications for video surveillance. While this may seem like a solution to our problem, currently moving-object detection face two challenges:

- This kind of application implies that there may in fact be many moving objects in a scene. In home settings, video-monitoring is often applied when the *least* amount of residents are home.

- Many of the objects that users would want to monitor, i.e. a large television or sofa, are significantly more likely to be *stationary* objects as opposed to moving objects.

In light of our desired application for our video surveillance system, we opted for an object-detection algorithm that *a.)* could be readily ported over to an iOS application and *b.)* would be able to be performed relatively quickly. While an SVM with HOG (Historgram of oriented gradients) trains large datasets through hard-negative mining, the

speed at which these calculations are performed - speed relative to wall time - is poor, considering that constant video monitoring over small, regular intervals is necessary. As a result, we decided to implement object-detection over a video sequence using feature-based object detection by utilizing SIFT Feature matching between a frame and its given template. The template that will be provided is the object in the room that the user interactively selects through their iOS device - simply pointing the camera at an object and taking a photo of the object sends the template data to our implementation of Shift Feature matching to begin monitoring a room over a long time period.
Equation 3 highlights the ratio calculated by our application. Its purpose is to determine the probability that an object is simply being temporarily obstructed, or if there is an actual home burglary.

$$R = room \quad (1)$$

$$O = Object \quad (2)$$

$$O\,Visibility = \frac{\#keypoints\,in\,current\,image}{\#keypoints\,in\,initial\,image} \quad (3)$$

$$R\,Visibility = \frac{\#R + O\,keypoints\,in\,current\,image}{\#R + O\,started\,monitoring} \quad (4)$$

$$Theft\,likeliness = R\,visibility - O\,visibility \quad (5)$$

2.2. Assessment

In order to test the function of our application, we ran a series of tests on trained sets of images. We cropped the image of the *item* and used that as a template for the entire scene. We were able to produce a few different kinds of ratios. We used three metrics to determine whether or not an object was in fact "missing." We aimed to use these *general* metrics (actual ratios that we used were dependent upon the data that we produced from our application). These ratios were calculated using Equation 3.

1. *75 % or greater* - The object is still in the frame.
2. *50 % or greater* - The object in question is most likely being obstructed by something (i.e. person walking bye, etc.)
3. *25 % or less* - The trained object is not in the image. This can ultimately that either the material was stolen our there is significant obstruction.

Using these metrics as guidelines, we showed that when a ratio fell below the recommended level, we sent a notification to the user's phone (if the user is using their phone as the surveillance device, then email as well). One of our goals in this project was to eliminate the amount of false alarms that other home security solutions generate. As an example, consider the surveillance system Dropcam (`http://www.dropcam.com`). Their software solution runs into the problem of detecting too many false positives in the video stream. One of our main goals in this assignment was to not just minimize the amount of notifications users receive, but to increase th likelihood that these notifications meant a more serious transgression occurred. By using SURF features and ratios, our algorithm becomes less susceptible to light while the hardware is able to function as an object-detection-based home surveillance system.

2.3. SIFT Feature Matching

SIFT (Scale-invariant feature transform) describes the features of an image that are independent of local changes to a scene. These changes could include potential lighting differences, affine distortion, scaling, and orientation. Since SIFT is at least partially invariant to these variables, we used this type of feature matching in our object-detection application. As a result, sudden light changes in the environment are less likely to cause false positives than in other home security systems. We hypothesized that the invariant nature of SIFT would be critical to improving home security solutions. Changes in weather, affine distortion (slight bumps in the camera), and the like will not significantly reduce the number of keypoints. Retaining keypoints is critical to our ratio method, because if too much sunlight causes a large variation in the number of SURF/SIFT detectors, then this would defeat our ease-of-usability standard, something we strove for throughout the project.

3. Approach

3.1. MATLAB Preliminaries

Our first approach involved using MATLAB library functions to experiment whether or not using ratios of features would produce reliable data. We used the following functions to generate and match keypoints between two images:

- *detectSURFFeatures* - This detects the SURF features in a grayscale image.
- *extractFeatures* - An explicit call to extract the features from the previous computation.
- *matchFeatures* - matches the extracted features between two images.

When we ran a series of tests on MATLAB, we saw that when we inputted images that contained the object in question, we were able to see an output of a high number if matches as opposed to the case where the object was removed from the image. In one example, we trained our application on an image of a chair. We were able to detect 400 keypoints. When we took another still image with the image

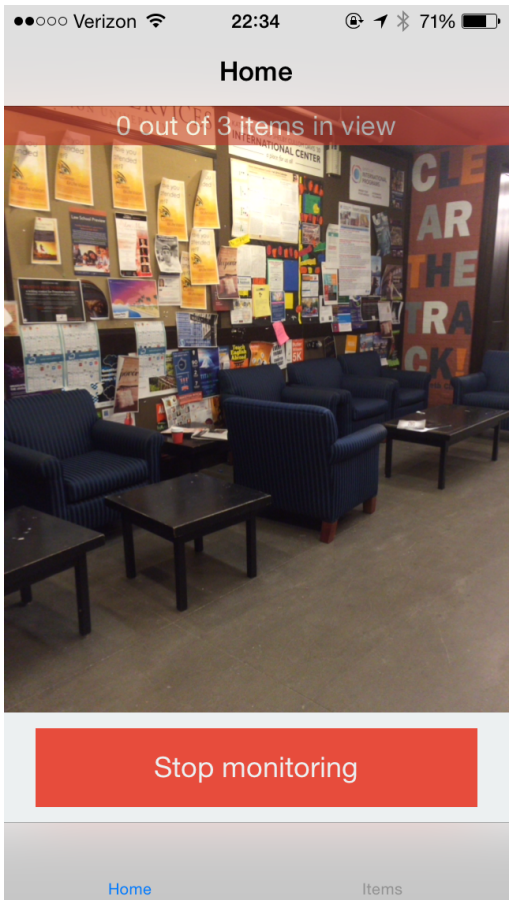

Figure 1. Example screen of our monitoring system.

still in the frame, MATLAB was able to find 380 of those keypoints. However, when the chair was removed from the following image, that number dropped all the way down to twenty. It is important to note that we took a zoomed-in image of our template - other we would be doing 'room detection' as opposed to object detection.
After performing a variety of tests on multiple images, we hypothesized that using a ratio to determine when an object went 'missing' in a room could be a great way to achieve our goal.

3.2. OpenCV Brute-Force Method

However, in order to be able to port the MATLAB-style functions to a more user friendly format, we decided to port over the functions to an iOS framework. In order to do so, we took advantage of the OpenCV library, an open-source computer vision repository with numerous computer vision algorithms at our disposal.[1] Our first attempt at producing reliable data involved using the Feature2D library in addition a brute-force keypoint matching algorithm between the two images.

After implementing this in iOS (Figure 1), we produced data that had both positive and negative aspects. This method was able to produce a large amount of distinct keypoints for the given object - approximately 400. In addition, when we took a picture of just the room without our object, we were able to generate another 400 keypoints. However, when we implemented out brute-force feature matching, we still generated too many positive keypoint matches between the two images. When we ran our code, we still generated too many positive matches between our two images (our first image contained the chair while the second image contained an identical setting, minus the object that we were looking for). Specifically, this method would produce almost all the points as matches.

Therefore, we concluded that using the Brute force and feature2D methods from OpenCV were not helpful in determining like objects between two objects.

3.3. SIFT Feature Detection

Rather than using the feature2D library, we then opted for more nuanced solutions. In particular, we implemented SIFT using the FLANN algorithm. In OpenCV, FLANN stands for Fast Library for Approximate Nearest Neighbors. FLANN is able to perform highly-optimized nearest neighbor searches, especially on large datasets. We decided that implementing FLANN could produce features that were more distinctive. In other words, by producing less features on the whole template image, this increases the likelihood that there are more features *on the actual object* as opposed to background outliers. The template that we used can be found in Figure 2. We first turned our image into a grayscale image, and we were able to generate significantly less feature points. Furthermore, the feature points that we did generate were much more unique to the image. For a comparison, we produced approximately 100 keypoints from Figure 2 (we also produced approximately 200 points from the frame of the room). With less keypoints, we then ran the matching algorithm using FLANN to produce a fast result. Indeed, the number of matches decreased significantly. We outputted approximately 40-60 matches out of the original 100-200 with the chair in the room. However, when matching with an image without our object in sight, we still produced a similar number of matches as if the object had been in the room.

3.4. SURF Feature Detection

After having tun two different experiments with mixed results, we tried to use algorithms more closely associated with our MATLAB functions from before. Specifically, we changed our approach slightly by using SURF feature detection as opposed to SIFT. SURF (Speeded Up Robust Features) is a faster implementation of SIFT that uses a box filter to approximate the Laplacian of Gaussian. The OpenCV

Figure 2. The grayscale image of our object, with colored dots illuminated the calculated features.

repository also contains algorithms for performing SURF on images. Since it is based upon SIFT and is an approximation of the LoG, we theorized that the calculated feature points would differ from the SIFT method. Ultimately, we ran multiple tests using this approach and we are able to produce similar ratios of keypoints on the template/when the object was in the room (100 keypoints for the object and 200 for the object in the room). We were able to produce around half the number of matches that the SIFT implementation gave us.

In addition, we were able to get *some* actionable data that showed some differences between the number of features in the room with the template vs. the number of features with the object removed. For certain objects, such as a chandelier, we were able to produce a $\frac{2}{3}$ ratio of keypoints with object present vs. total keypoints. When the object was removed, the ratio decreased to $\frac{2}{3}$. However, for other objects such as the chair from Figure 2, we saw that ratio of keypoints remained very high, even though the object was removed from the frame.

What we have been able to conclude is that our method can provide *some* actionable data for objects with very distinct surfaces. However, for more generic objects, our method is harder to verify when the object has gone missing from the room.

4. Related Work

Our main two challenges were categorizing pixels that belonged to our objects versus the room and then accurately determining if those objects were still present in a later frame. Such a problem relies on two main bodies of research: object recognition and object tracking. Traditionally, these types of problems - traffic controlling, sports officiating, manufacturing - can sacrifice object detection accuracy for improvements in tracking latency. For example, you can accept inaccurate vehicle identifications (was that a car or a large SUV?) for nearer real-time tracking (is that vehicle in the right lane or left lane? Is it going 40 mph or 50 mph?) in deciding when and how to change traffic lights. However, in our scenario, the opposite is true. The precision of real-time tracking is not nearly as valuable as proper object identification. If your Picasso is stolen, you might be curious as to how quickly and in which direction it left the room. However, you would probably be much more concerned with understanding if in fact your Picasso **was** stolen or if it was your dog in front of the camera again.

5. Conclusion

Based on the various methods that we have applied, we can conclude that actionable data can be, at this current time, best implemented using SURF feature detectors on more unique objects. While this is certainly not as high of a ratio as we would have liked, by using SURF we were able to produce data that coincided with our hypothesis of using features to determine when an object is missing from a scene. Furthermore, we see the potential applications of object-detection for surveillance in more professional settings such as art galleries, or even department stores. For example, in an art gallery, one can have stationary cameras that are solely pointed at an object. We could, at regular time intervals, run SURF on the template (the art piece) and the environment to see if the object has moved. This is especially beneficial, because even if a potential thief could tamper with what the camera actually sees, at the next time interval for the camera, it can sense that the feature points for the current viewpoint (say, a blurry screen) would have a significantly lower ratio, and as a result, can notify the proper authorities when an object is stolen.

In order to further improve upon our implementation, we propose running SURF in parallel for multiple different trained images. This would grant users the luxury of not having to necessarily lump all of their values into the viewpoint of the camera (this also helps them not have to

buy too many video-surveillance cameras). Furthermore, we propose that combining a SURF feature analysis with RANSAC to estimate the homographies of given frames. This can help to solve the viewpoint bias, which is a problem for many different classes of objects (i.e. chairs, tables, etc.). Nevertheless, Sentry has granted user's the ability to specify what it is that they are trying to keep track of in a room, hallway, or art gallery. By using iOS, we hop to increase the portability of this type of application, so that it can be applied into a vast sea of scenarios. We believe that our approach is very promising, and can lead to further actionable data from larger and larger datasets. We hope to be able to put our application into practice in the near future, thereby helping to show computer vision's usefulness in saving the world.

References

[1] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.