You Are Being Watched - On The Detection of Spy Cameras With Image Processing Analysis

Ighor Tavares
McKelvey School of Engineering
Washington University in St. Louis
f.tavares@wustl.edu

***Abstract*—Over the past few years, the usage of Internet of Thing (IoT) devices have dramatically increased and the interest is still growing. These devices can be helpful for an individual's daily routine; however, they also can be used for malicious purposes. While there are many devices and methods of identification, none present an efficient result. In this investigation, we developed an innovative method for the identifying hidden cameras using image processing analysis. This technique is capability to analyzing raw data from extracted frames from either video or pictures. It also has he ability to detect patterns of lens reflection and lens body shape, dynamically as well as statically.**

***Index Terms*—IoT Spy Cams, image processing analysis,openCV, security, arduino**

I. Introduction

Security cameras have been around since the times of World War II, and they are becoming more popular than ever nowadays. Whether you use it to protect your home or business, there are a variety of cameras that you can purchase. The most popular ones are the "spy cameras." Most of the time, there are small IoT devices which allow you to connect to your phone or computer to watch it live. This has led to new challenges where people or businesses might be using these smart devices to not only protect themselves, but also to spy on people. A common issue has been recently found in Airbnb. Hidden cameras were being found in the commodities and customers were being watched without their knowledge or permission.

Spy camera detection is a new process for identifying hidden surveillance cameras. It will detect hidden cameras in environments where one's privacy is being violated. There are many techniques and tools which can be used. Most spy camera detectors mainly offer two ways to find hidden cameras: RF detection or reflective lights from their lens. New approaches have been introduced in recent studies, such as thermal detection and network analysis. Additionally, there are plenty of hardware which provides some sort of reflective detection, but they are found to be somewhat inefficient.

We leverage a new technique of detecting spy cameras using image processing analysis to detect lens and its reflection patterns while utilizing a custom built LED detector.

II. Custom LED Detector

Many detectors on the market are expensive and have limitations on what could be changed within the hardware. The custom LED detector was developed and built with the goal of being fully controllable and programmable.

Our goal for this project is to use a custom-built detector and to be able to control different blink frequencies, brightness, colors and provide essential data. This will help us understand how such differences could improve the detection of hidden cameras. To accomplish such a goal, we built this device with Arduino [1]. Much of the time spent required developing a consistent tool with the following requirements:

- Dynamically change blink frequency of LED
- Dynamically change LED brightness
- Multiple colors available
- Controlled via serial monitor

To achieve all the requirements of our custom detector, we were faced with a few engineering challenges. One challenge pertained to identifying which blink frequencies would work the best with the current hidden cameras. Another was to determine what type of lens they possess. We also had to consider the possibility that not every lens will the same reaction at certain frequencies. In addition to blink frequency, we also need to take the light intensity into account. We need to determine if light intensity influences the identification of the lens, and if there are any correlations between to how bright the LEDs are and how fast they blink. To achieve different light intensities, we used a potentiometer. This allows us to control the resistance of our circuitry and control the voltage going to the LED; hence changing the light intensity.

III. Image Processing Analysis

Image Processing [2] analysis is the domain which is frequently used in Computer Vision. It uses images as input and as a result of the image processing, the output is also an image. The goal of Image Processing analysis is to analyze raw data, perform the necessary pre-processing, and use it.

We used image processing analysis dynamically and statically. The objective of using dynamic image processing is to test and evaluate data executed in real-time, finding errors while data is being processed. Examining a live feed video, we can use it to detect any changes on the original data frame based on a threshold. The goal of using static image processing is to analyze raw data in static images. This will allow us to identify possible patterns which resemble circular shapes, like a camera lens.

The main goal for this investigation was to develop two scripts to identify patterns on live feed video and static images. The first script compares each frame from the input and processes it to identify errors. This would be the changes based on a threshold. This pattern would represent the reflection glare from the lenses caused by a light source. The second script used the 21HT method (2-1 Hough Transform) [3] for circle finding in a grayscale image.

IV. Implementation

A. Arduino Custom Detection

Arduino provides many useful tools within its own architecture. There are many compatible tools available to allow us to build and customize our own tools. However, working with Arduino involves two challenges: Hardware and Software communication. We decided to build our own LED Detector with the intention of identifying hidden cameras caused by its lens reflection.

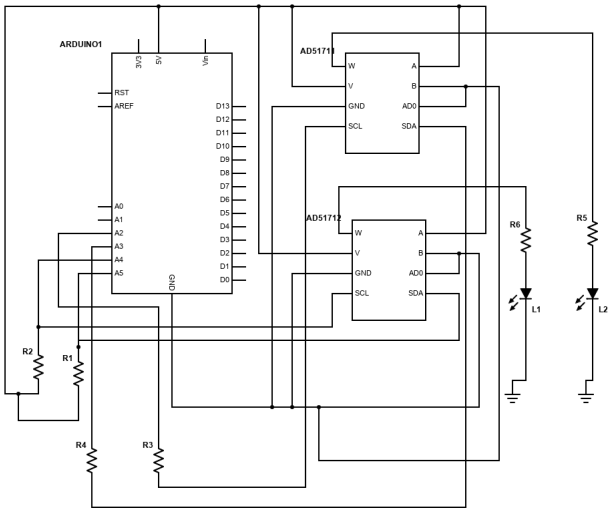

Fig. 1. Simplified Schematic of Custom LED Detector

To successfully build our custom device we needed the following hardware and components:

- Arduino Uno
- AD5171 Digital Potentiometer
- 3 x High Brightness LED
- 4 x 4.7k Ω Resistors
- 3 x 220 Ω Resistors
- Breadboard
- Hookup Wires

Using Arduino IDE, we build a sketch [4] in which during run time, takes two inputs from the serial terminal: LED color and blink frequency in Hertz (HZ). The sketch then starts our detector with default resistance for brightness, then uses our digital potentiometer to dynamically change resistance as needed. In this sense, our detector is flexible for different settings.

B. Image Processing Analysis

Python is one of the most flexible programming languages; known as a general-purpose programming language. Python is perfect for data analysis; based on the fact it is an open source programming language providing an abounding number of available libraries to be used. We chose Python to develop our first script to analyze multiple frames from a live feed video of a video input device, such as a webcam. We also choose python for our second script to perform a static analysis from an image, in this case a single frame.

Specifically, for the dynamic image processing script, we created two frame variables that are updated up to 30 times per second. The first frame variable is defined as `None`, and the second frame variable is defined as a grey version of the original frame from the camera. Once both variables are set, the first frame is constantly updated with the grey frame after they are both checked. In this, we can compare both frames at run-time. The reasoning for making the frame grey is for when defining a threshold, we use a OpenCV [5] python library function $THRESH\_BINARY$ in which defines a binary threshold for the pixels of the frame. Once the threshold is defined, we use a copy of this frame to then run another CV2 function, `findContours`, to find specific patterns of contour in a binary image. Once possible contours are found in the binary image, we store those results in a python data structure `list`. In this, we compare all the circles with another threshold and pattern for lens reflection, which is then "painted" on top of the output frame.

Likewise, the script for the static image processing utilizes functions from OpenCV [5]. The script takes an input of an image and creates a variable to read it as $cv2.IMREAD\_COLOR$ which converts the image to the 3 channel RGB color image, followed by $cv2.COLOR\_BGR2GRAY$. In this, we convert the 3 channels from RGB to grayscale, making it easier to work with the raw data of the image. The process is followed by blurring the image with $cv2.blur$, which is used to remove high frequency content, such as noise, from the image data. Once the data is normalized, we proceed to analyze with the Hough Transform algorithm [3] to detect circles in our processed image. With the help of NumPy [6], we convert the circle's parameters from the algorithm into integers in which we can use to draw the circumference of the resulting circle onto the image.

The process of development of these scripts compromised two major steps. First, we have to normalize the raw data from the input frames. Then we had to analyze the raw data frames for specific thresholds and patterns. From here, we developed a main script using bash to automate the testing process.

In the example below in "Fig. 2", we offer an output example of the image processing analysis.

Usage

The code repository can be found at $https://bit.ly/35rfkgs$ [7] . Our main script executes both dynamic and static image processing analysis. The script

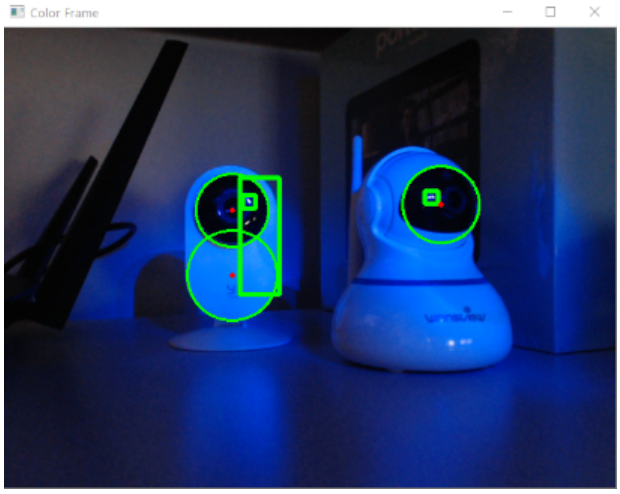

Fig. 2. Example of image processing analysis script output

can be used through the command line using the format found in ”Fig. 3.”

```
bash ./run.sh <output name>
```

Fig. 3. Main script usage

Alternatively, you can manually run each individual script through the command line using the formats found in ”Fig. 4.”

```
python motion_detector.py [-h] [-o OUTPUT]
python static_ana.py [-h] [-i INPUT]
```

Fig. 4. Individual script usage

Learning And Future Work

Within our proof-of-concept image processing analysis, we applied filters to reduce the noise in our outputs. However, our scripts' results still had a great deal of noise. This made small changes in the binary of the frame as fitted for our pattern. Future work would involve improving our script to eliminate such noise and increase the accuracy to our pattern.

In addition, our scripts can be extended upon in future work to mobile applications or a more sophisticated portable device. These may include having all processing happen remotely through an API for faster processing.

Conclusion

Using Image Processing Analysis, we demonstrated how it is possible to process normalized raw data from an image. We were also able to identify patterns which lead to the detection of possible cameras within an environment. Python provided us with many tools and flexibility to accomplish the goal of this project.

We successfully built a custom LED detector and extensible python scripts that handles dynamic and static image processing analysis for further detection of hidden cameras. We look forward to others making use of this to further develop different approaches for hidden camera detection.

Acknowledgment

I want to acknowledge and send my special appreciation to Professor Ning Zhang for his guidance and teaching. I also want to send my thanks to my friend Cheris Black for helping me with the digital potentiometer.

References

[1] Website, ”Arduino”, https://www.arduino.cc/
[2] Webiste, ”Intro to Image Processing in Python With OpenCV”, https://stackabuse.com/introduction-to-image-processing-in-python-with-opencv/
[3] Website, ”2-1 Hough Transform”, http://www.bmva.org/bmvc/1989/avc-89-029.pdf
[4] Sketch, https://www.arduino.cc/en/tutorial/sketch
[5] Website, ”OpenCV”, https://opencv.org
[6] Website, ”NumPy”, https://numpy.org/
[7] Webiste, https://github.com/itavares/Master-s-Project—Image-Processing-Analysis