CMPSC 370
Introduction to Artificial Intelligence
Fall 2016
Janyl Jumadinova

Lab 4 Part 2
18 October, 2016
Due: 1 November by 2:30 pm
This is a team lab (same teams as in lab 4 part 1).

Objectives

To learn how to use a computer vision `OpenCV` software, a Python programming language and machine learning techniques for the problem of video surveillance. To correctly execute basic motion detection on Raspberry Pi using pi camera. To extend the basic motion detection to identify humans using a machine learning algorithm. To build and validate a Bayesian network using prior information.

Overall Goals and Structure

This laboratory assignment is divided into two 2-week parts. The stated deliverables for each part of the lab are to be submitted by the due date. This lab will consist of three separate (3 lab grades): one for the tasks in each part of the lab and one for the overall system design, experimentation and analysis. A completed and well-documented lab with the results of your experiments and your analysis is due on November 1. The three lab 4 grades will be assigned after this date, although you will receive a pending grade for part 1 before then.

During the second part of the lab, you will extend the basic motion detection (from part 1) to be used on the raspberry pi. First, you will make motion detection system more robust for continuous run and you will integrate it with the Dropbox API to allow your program to automatically upload security photos to your Dropbox account. This portion of the second part of the lab is built on the tutorials by Dr. Rosebrock `http://www.pyimagesearch.com/2015/06/01/`. Then, you will enhance your motion detection system by adding human recognition using one of classifiers implemented on OpenCV (cascade, HOG/SVM, etc.) and will run extensive experiments. Finally, you will build a Bayesian Network for predictive surveillance modeling.

1. Motion Detection with Raspberry Pi

(a) Set Up: Pi Camera via OpenCV

1. Make sure the camera is correctly inserted into the Pi and that it is enabled. To enable the camera you can type `sudo raspi-config`, use your arrow keys to scroll down to Option 5: “Enable camera”, enter to enable the camera, and then select “Finish” button. You will need to reboot your Raspberry Pi for the configuration to take affect.
   It is also a good idea to test (again) your camera to ensure that it is properly connected

and working, before you get into OpenCV. To quickly test the camera functionality, type `raspistill -o output.jpg`, which should display an image and save it into an output file.

2. You can use Raspberry Pi camera through Python using `picamera module`. First, you will set up virtualenv to cleanly install and separate `picamera` module from the system Python and packages. Type:

```
source ~/.profile
workon cv
pip install "picamera[array]"
```

(b) Raspberry Pi + Python + OpenCV + Dropbox

You will have your surveillance system upload security photos to your Dropbox account, thus install the dropbox package: `pip install dropbox`. You will also need to register with Dropbox Core API (`https://www.dropbox.com/developers`) to get your public and private API keys.

In the `cs370f2016-share/lab4/part2/surveillance` directory, you will find `PiSurveillance` program, which implements the main functionality, `imagesearch` package for organization purposes (temporarily write images to disk before sending them to Dropbox), and JSON configuration file (`conf.json`) that will store command line arguments. The JSON configuration file stores important variables described below:

- `show_video` : A boolean indicating whether or not the video stream from the Raspberry Pi should be displayed to our screen.
- `use_dropbox` : Boolean indicating whether or not the Dropbox API integration should be used.
- `dropbox_key` : Your public Dropbox API key.
- `dropbox_secret` : Your private Dropbox API key.
- `dropbox_base_path` : The name of your Dropbox App directory that will store uploaded images.
- `min_upload_seconds` : The number of seconds to wait in between uploads. For example, if an image was uploaded to Dropbox 5m 33s after starting our script, a second image would not be uploaded until 5m 36s. This parameter simply controls the frequency of image uploads.
- `min_motion_frames` : The minimum number of consecutive frames containing motion before an image can be uploaded to Dropbox.
- `camera_warmup_time` : The number of seconds to allow the Raspberry Pi camera module to "warmup" and calibrate.
- `delta_thresh` : The minimum absolute value difference between our current frame and averaged frame for a given pixel to be "triggered" as motion. Smaller values will lead to more motion being detected, larger values to less motion detected.

- `resolution` : The width and height of the video frame from our Raspberry Pi camera.
- `fps` : The desired Frames Per Second from our Raspberry Pi camera.
- `min_area` : The minimum area size of an image (in pixels) for a region to be considered motion or not. Smaller values will lead to more areas marked as motion, whereas higher values of min_area will only mark larger regions as motion.

Now, go through the Python programs, following the explanations below:

- The first few lines in the `PiSurveillance` program import the necessary packages and handle the command line arguments. Then, we filter warning notifications from Python (e.g., generated from `urllib3` and `dropbox` packages). And, load JSON configuration dictionary from disk and initialize the Dropbox client.
- Next, we check with our JSON configuration to see if Dropbox should be used or not. If it should, the next lines start the process of authorizing and integrating with Dropbox.
- Now, we setup a raw capture to the Raspberry Pi camera by allowing the Raspberry Pi camera module to warm up for a few seconds, ensuring that the sensors are given enough time to calibrate. Then, we initialize the average background frame, along with some bookkeeping variables. Finally, we loop over frames directly from our Raspberry Pi video stream: we pre-process our frame by resizing it to have a width of 500 pixels, followed by converting it to grayscale, and applying a Gaussian blur to remove high frequency noise and allowing us to focus on the structural objects of the image – this should all be familiar based on lab 4 part 1 code. However, in the basic motion detection program, we made the assumption that the first frame of our video stream would be a good representation of the background we wanted to model. But as the time of day (lighting conditions) changes and as new objects are introduced into our field of view, our system will falsely detect motion where there is none. To combat this, we instead take the weighted mean of previous frames along with the current frame. This means that our program can dynamically adjust to the background, even as the time of day changes along with the lighting conditions. This is still quite basic and not a perfect method to model the background versus foreground, but it is much better than the previous method. Based on the weighted average of frames, we then subtract the weighted average from the current frame, leaving us with what we call a frame delta. We can then threshold this delta to find regions of our image that contain substantial difference from the background model – these regions thus correspond to "motion" in our video stream. To find regions in the image that pass the thresholding test, we simply apply contour detection. We then loop over each of these contours individually and see if they pass the `min_area` test. If the regions are sufficiently larger enough, then we can indicate that we have indeed found motion in our current frame. Then, we compute the bounding box of the contour, draw the box around the motion, and update our `text` variable. Finally, we take our current timestamp and status `text` and draw them both on our frame.
- Uploading to Dropbox: We check to see if we have indeed found motion in our frame. If so, we make another check to ensure that enough time has passed between now and the previous

upload to Dropbox - if enough time has indeed passed, we will increment our motion counter. If our motion counter reaches a sufficient number of consecutive frames, we then write our image to disk using the TempImage class, upload it via the Dropbox API, and then reset our motion counter and last uploaded timestamp. If motion is not found, we simply reset our motion counter to 0.

- Finally, we build in functionality to see if we want to display the security stream to our screen or not. We make a check to see if we are supposed to display the video stream to our screen (based on our JSON configuration), and if we are, we display the frame and check for a key-press used to terminate the script.

- As a matter of completeness, we also define the `TempImage` class in our `pyimagesearch/tempimage.py` file. This class constructs a random filename, followed by providing a cleanup method to remove the file from disk once we are finished with it.

To run this program, type: `python PiSurveillance.py --conf conf.json`

2. Human Face Detection

So far, we have used the background image subtraction method to detect motion in the video stream. Now, we will make our surveillance more intelligent by adding a task of human face detection. Since one of the objectives of lab4 is to utilize your surveillance cameras to monitor human instrusion in the local community garden, we want to isolate and record human faces visiting the garden.

For face detection we will use a supervised learning algorithm, a classifier, that uses certain features of the image to correctly label them as human faces or not. Design choice: you need to select a learning algorithm that you will utilize (e.g., Cascade, SVM) and a feature descriptor that your algorithm will use (e.g., Haar, HOG, LBP). I suggest using LBP or HOG features, as they are integer, fewer and more discriminant in contrast to Haar features, so both training and detection with LBP and HOG are several times faster then with Haar features. Although, you can collect your own labeled data (images) and then train your classifier with those, that task is time intensive. To save time, select a classifier that either has already been trained using specific data sets and a certain feature descriptor in OpenCV or select a feature descriptor that already has compiled labeled data set that can be used for training. You can utilize class programs and OpenCV documentation for usage examples of various methods.

3. Experimentation

After you have conducted preliminary tests to ensure your programs run without errors and you are satisfied with your detection accuracy, you need to design your experiments. Design choice: you need to decide on your testing environment, Raspberry Pi and camera set up (use a case) and the collection of results. You need to run your experiment for at least one hour, it could be either continuous or in intervals of a few minutes. In addition to the images generated by your system, you should collect other results that can demonstrate the accuracy of your system. These results could include automatically collected values from your program, such as the number of faces detected over time, and/or the manually observed values, such as the number of correct/incorrect detections, etc. You are required to produce visual graph(s) depicting your results.

You are welcome to conduct your experiments in any environment, including Alden Hall. However, you will receive 10 extra credit points for experimentation if you test your system in the garden area on campus.

4. Prediction using Bayesian Network

This part of the lab is more conceptual and is used to provide you with practical experience of the process after data collection, which may involve prediction and inferencing. Design choice: you will construct a Bayesian Network (at least 4 nodes) that will represent the surveillance scenario in the garden. You should construct some reasonable prior probabilities based on your observations. I have some data concerning the theft in the Meadville community garden, if it is helpful for your network.

You should create your Bayesian Network in AI Space `http://aispace.org/bayes/`. Then, select a query (what do you want to predict) and run a variable elimination algorithm through it. Save the network you create as a problem in AI Space and take a snapshot of your variable elimination output.

Required Deliverables

This assignment invites you to submit electronic versions of the following deliverables through your Bitbucket repository (`cs370f2016-<your-username>/lab4/part2/`).

1. Three snapshots from running the `PiSurveillance.py` program or other deliverable demonstrating the correct set up and run of the `PiSurveillance.py` program.

2. Correctly implemented and properly documented extended surveillance program with face detection.

3. Results from your full experimentation. The results should include 5 selected images obtained by your program and at least one visual graph that demonstrates the output in a quantitative way.

4. Bayesian network with full justification of the nodes and edges in the network and the CPT tables.

5. Response to a team member evaluation document (Google Form document to be sent through Slack).

6. A reflection document that contains the following:

    - A description of your face detection addition to the motion detection. Comment on the method you selected, why you selected this method and the observations of your results.
    - Provide analysis of your experiments. What general trend did you observe? What assumptions did you make? What worked well and what didn't?

- A description of the Bayesian Network, justification of its set up. Discuss how you constructed the CPT tables.
-
- Comment on any hardware or any other challenges you have encountered during this lab.