**CMPSC 310**
**Artificial Intelligence**
**Fall 2018**
**Janyl Jumadinova**

**Lab 4 Part 2**
**1 October, 2018**
**Due: 22 October 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 and a Python programming language for computer vision applications. To correctly apply smoothing, blurring, thresholding and edge detection to images using OpenCV functions. To gain understanding of image processing techniques that can be used for specific computer vision applications. 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.

## Overall Goals and Structure

This laboratory assignment is divided into two 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. The three lab 4 grades will be assigned after the final due date stated in this lab sheet.

During the second part of the lab, you will apply the basic motion detection to be used on the raspberry pi using OpenCV algorithms. This portion of the lab is built on the tutorials by Dr. Rosebrock. You will then make this motion detection technique more intelligent by applying a machine learning algorithm. Finally, you will run extensive experiments in one of two environments: either a computer lab or a student studio in the Art Department at Allegheny. Your systems will be evaluated by the judges from the Art Department in addition to your instructor, and the best and proper operating motion detection systems will be invited to be installed in the selected environment.

## Reading Assignment

You should carefully read and follow the chapters from the supplemental reading. Also, please read the explanations provided in this lab assignment sheet and the given tutorials carefully.

Also, as you write your reflection, please refer to the relevant "GitHub Guides", available at https://guides.github.com/, that explain how to use many of the features that GitHub provides.

## 1. OpenCV and Raspberry Pi Set Up

### (a) Setting up the Raspberry Pi

You need to have a monitor, keyboard and a mouse to set up a raspberry pi. The external monitor must have HDMI interface and USB keyboard and mouse, or you can use HDMI to VGA converter for most of the lab machines. If you use lab machines, please make sure to connect everything back to the computer after you finished using them for your Raspberry Pi. After the installation of the operating system you can `ssh` into your raspberry pi through the special wifi network that was set up for us specifically (details available via Slack). Alternatively, you can use your laptop's screen, keyboard and a mouse (or a trackpad) to connect to your pi, there are many online tutorials available describing the steps to accomplish this task.

Please find the "CanaKit Raspberry Pi Quick-Start Guide" in your kit and follow instructions starting on page 3 and ending on page 7. Please note that you should install `Raspbian` operating system and you should change the password after initial set up as you will be using `ssh` to connect to it after the initial set up. **Please make a note of the password on the raspberry pi cardboard box.** Finally, please ensure you enable the camera and ssh in the configurations. You will need to reboot your raspberry pi for the configuration to take affect.

### (b) Using Raspberry Pi Camera on the Raspberry Pi computer

Follow the tutorial on https://www.raspberrypi.org/learning/getting-started-with-picamera/worksheet/ to learn how to use and then experiment with Raspberry Pi camera.

### (c) Setting up OpenCV on the Raspberry Pi

In order to use OpenCV on a raspberry pi with camera you will need to correctly install OpenCV and all relevant packages. This is a lengthy process and may take couple of hours, I suggest you connect to your pi via `ssh` to perform these operations so that you do not need a screen and a keyboard tied up to your pi. As you wait for the installations to take place, please feel free to move on to the step 2 of this lab. OpenCV 3.2.0 that works with Python 2.7.15rc1 and Python 3.6.6 is already installed on all Alden machines. Please follow instructions on Dr. Rosebrock's tutorial to complete these installations on your pi.

You will also need to install `imutils` package on your pi:

```
pip install --upgrade imutils
```

And if you are interested in having your motion detection system upload security photos to your Dropbox, you will also need the dropbox package:

```
pip install --upgrade dropbox
```

## 2. Motion Detection with Raspberry Pi

You will use OpenCV, Python programming, a raspberry pi computer and a pi camera module to create a motion detection system by going through the given instructions and explanations, and then extending the given programs by adding more intelligence. If you are not familiar with a Python programming language, you may refer to Python library documentation at https://docs.python.org/3.6/.

There are many algorithms in OpenCV for performing motion and face detection, tracking, and analysis. In motion detection we generally assume that the background is mostly static, this way we can model it and then monitor for big changes. Many of the algorithms in OpenCV (Gaussian Mixture Model, Bayesian Model) use sophisticated methods to segment the background from the foreground. Since our goal is to perform motion detection and tracking on a raspberry pi, we will use a relatively simple approach.

## (a) Understand and explore the basic motion detection program

In the `lab4/src` directory of your lab4 repository, you will find `motiondetector` program and two videos to use with this program. As you read the explanations of the program below, map each step to the lines in the program. Finally, run the program from your lab machine to test both videos provided to you.

- The first few lines import the necessary packages. If the `imutils` package, which is a set of convenience functions to make basic image processing tasks easier, does not work for you, you can install it locally via pip: `pip install imutils`.
- Next, we parse the command line arguments. The `--video` argument is optional. It simply defines a path to a pre-recorded video file that you can detect motion in. If you do not supply a path to a video file, then OpenCV will utilize your webcam to detect motion.
- Then, we define `--min-area`, which is the minimum size (in pixels) for a region of an image to be considered actual motion (small regions of an image could change substantially without motion, likely due to noise or changes in lighting conditions). That's why we define a minimum size of a region to combat and filter out these false-positives.
- The next few lines handle grabbing a reference to the camera object. In the case that a video file path is not supplied, we take a reference to the webcam. And if a video file is supplied, then we create a pointer to it.
- Next, we declare and initialize a variable called `firstFrame`, which stores the first frame of the video file/webcam stream. We make an assumption that the first frame of our video file will contain no motion and just background - therefore, we can model the background of our video stream using only the first frame of the video.
- Now we can start looping over each of the frames. A call to `camera.read()` returns a 2-tuple. The first value of the tuple is grabbed, indicating whether or not the frame was successfully read from the buffer. The second value of the tuple is the frame itself.
- We also define a string named `text` and initialize it to indicate that the room we are monitoring is "Unoccupied". If there is indeed activity in the room, we can update this string.
- The `break` statement (to get out of the loop) is used in the case that a frame is not successfully read from the video file.
- Now we can start processing our frame and preparing it for motion analysis. We first resize it down to have a width of 500 pixels - there is no need to process the large, raw images straight from the video stream. We also convert the image to grayscale since color has no bearing on our motion detection algorithm. Finally, we apply Gaussian blurring to smooth our images. It is important to understand that even consecutive frames of a video stream will not be identical! Due to tiny variations in the digital camera sensors, no two frames will be 100% the same as some pixels will most certainly have different intensity values. However, we need to account for this and apply Gaussian smoothing to average pixel intensities across an 11 x 11 region. This helps smooth out high frequency noise that could hinder the motion detection algorithm.

- Since we need to model the background of our image, we make the assumption that the first frame of the video stream contains no motion and is a good example of what our background looks like. If the `firstFrame` is not initialized, we store it for reference and continue on to processing the next frame of the video stream.

- Given the static background image (modeled through `firstFrame` variable, we can now actually perform motion detection and tracking. We use `firstFrame` variable to calculate the difference between the initial frame and subsequent new frames from the video stream. Calculating the difference between two frames is a simple subtraction, where we take the absolute value of their corresponding pixel intensity differences as : $delta = |background\_modelcurrent\_frame|$

- To show regions of the image that only have significant changes in pixel intensity values, we apply thresholding to the `frameDelta`, such that if the delta is less than 25, we discard the pixel and set it to black (i.e. background), and if the delta is greater than 25, we set it to white (i.e. foreground).

- Now, given this thresholded image, we apply contour detection to find the outlines of these white regions. We loop over each of the contours, where we filter the small, irrelevant contours. If the contour area is larger than our supplied `--min-area`, we draw the bounding box surrounding the foreground and motion region. We also update our text status string to indicate that the room is "Occupied".

- Finally, we draw the room status on the image in the top-left corner, followed by a timestamp (to make it feel like "real" security footage) on the bottom-left. The results allow us to visualize if any motion was detected in our video, along with the frame delta and thresholded image so we can debug our program.

### (b) Raspberry Pi + Python + OpenCV + Dropbox

Now, we will extend the previous technique to be used on a raspberry pi computer. Design choice: *you will need to select a method for storing and sending your detection photos.* In my instructions, following the tutorial of "pyimagesearch", we will refer to usage of the Dropbox. However, you do not need to use Dropbox. For example, you can store your detected images locally and upload them to your repo or copy them to a remote location using `scp`. You can also choose to send your detected images as a text message, for example, by using the Twilio API. The Twilio API is free (with some minor restrictions) and is very simple to use. You will need to first register for an account on Twilio's website, after which you will automatically be assigned a phone number that you can use for sending messages. You will also want to make a note of your *AccountSID* and *AuthToken* which are the credentials used to access the Twilio API and can be found on your Twilio Account page. Then, to send messages you can do something like:

```
python
>>> from twilio.rest import Client
>>> TWILIO_SID = "your SID from account page"
>>> TWILIO_AUTH = "your AUTH KEY from account page"
>>> client = Client(TWILIO_SID, TWILIO_AUTH)
>>> TO = "+18140001111"
>>> FROM "your Twilio phone number"
>>> client.messages.create(to=TO, from_=FROM, body="Hey!",
media_url="https://site_or_repo/image.png")
```

If you choose your motion detection system upload detection photos to the Dropbox account, make sure you have installed the dropbox package: `pip install --upgrade 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 `src/lab4` directory of your repository, you will find `piMotionDetection` 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 `piMotionDetection` 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 and the basic motion detection program you studied earlier in this part of the lab. 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 `imagesearch/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 from your raspberry pi after your camera has been properly connected, type:

```
python piMotionDetection.py --conf conf.json
```

You should experiment with this program by running it in any environment for at least a few minutes and collecting results for your documentation (see the relevant section below).

### (c) Human Face/Body 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/body detection. Since one of the objectives of lab4 is to utilize your cameras to detect human motion in the spaces of the Art Department at Allegheny, we want to isolate and record human faces/bodies visiting the art spaces. You should modify and save/send motion images of only human faces/bodies.

For face/body 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, etc.) 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 for your Extended Program

After you have conducted preliminary tests to ensure your extended program runs 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 set up, raspberry pi and camera set up (how to position, etc.) and the collection of results. You need to run your experiment for at least one hour. 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 initial experiments in any environment, including Alden Hall. However, your final experiments must be conducted in the assigned environment in the Art Department (see #labs Slack channel for your assignments).

## Required Deliverables

This assignment invites you to submit electronic versions of the following deliverables through your lab4 team GitHub repository.

1. Correctly set up and run motion detection program (the given program without face/body detection, with possibly minor adjustements).
2. Correctly implemented and properly documented extended motion detection program with face/body detection.
3. README documentation written in Markdown that contains a broad overview of the system and contains sections that detail:
    - Step by step instructions of setting up your system. Include all of the neccessary installations. Anyone should be able to follow your instructions to get your system up and running.
    - Overview descriptions of each program (1. and 2. above) with instructions on how to run each.
    - Output section (see below).
4. (a) Description of your experimental set up and the results from running the motion detection program (without face/body detection, clearly label this as such). You should provide two selected images to showcase your ability to run the program correctly. (b) Description of your full experimental set up and the results from running your extended program with face/body detection in the assigned space in the Art Department. These results should include five selected images obtained by your program. Images should be incorporated into your `README.md` Markdown document.
5. Response to a team member evaluation document (Google Form document to be sent through Slack).
6. An addition to your `writing/reflection` document from part 1 of lab 4 that outlines the following:
    - A description of your human face/body detection technique that was added to the motion detection problem. 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? You have to produce and incorporate into your Markdown document at least one visual graph that demonstrates the performance of your system in a quantitative way.
    - Comment on any hardware or any other challenges you have encountered during this lab.
    - Outline your team work strategy and include details on the work completed by each team member.