Niwot High School Independent Study: Security Camera

By: Aidan Coopman
Year: Junior
Date: May 16, 2019

Overview

The goal was to make a security camera such as Arlo, using openCV C++ library with a standard webcam. The model used was a simple background subtraction detection where an input frame is subtracted from a background frame. The heart of the algorithm is creating the background frame. For example, if a static (non changing) background is used, and an object enters and remains in the camera's view, the object will constantly trigger a detection. To make the background frame dynamic (changing slowly) a exponential decay filter is used. The filter allows new objects to slowly become part of the background frame, hence avoiding constant detection triggers.

Other features include the input frame being blurred so small camera movements or small changes in the image such as leaves on the trees do not trigger a detection.

The final classification if an object should be detected is done by the percent of pixels that have changed from the background frame and the the standard deviation (or spread) of the changes pixels. For example, if the entire camera has moved, the std of the x and y location will be very larger whereas std of object such as a person will be low.

Demos to view before moving on in this paper are:

1) https://drive.google.com/file/d/1uXdBhN-I-ZA9AFPpU_SLfaysUR8SFi6f/view
2) https://drive.google.com/file/d/1-IJbS520U5C_N_e8JpYvUw1GoC2ujB0c/view
3) https://drive.google.com/file/d/1gKF4m_WWakgm2c_RYL2DB02F7sedZMzG/view

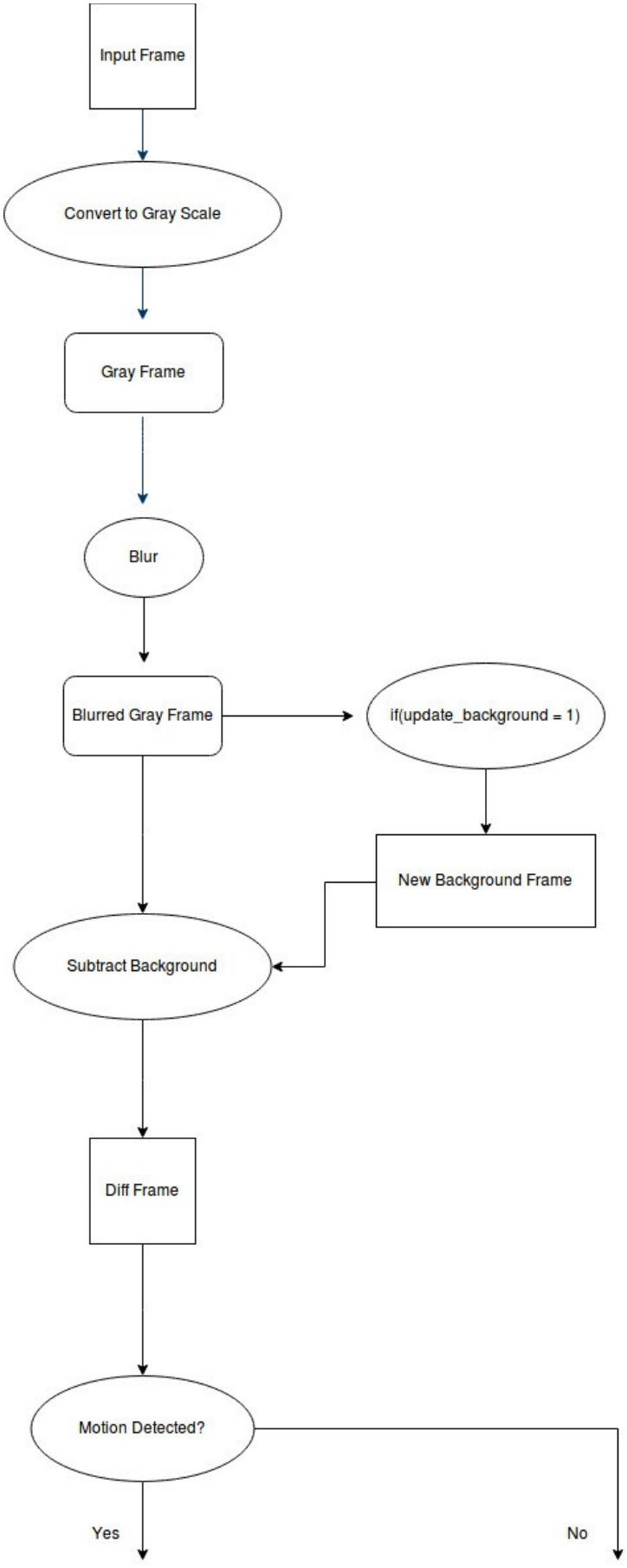
Code for this project:

https://github.com/acoopman/IndependentStudySecurityCamera

To run the program:

```
>> ./video_in  -s 0     //source is the default camera on a laptop
>> ./video_in  -s 1     //source is the usb camera
>> ./video_in -f input.avi -w output.avi  //reads the input file and writes the output file
```

Object Detection Pipeline



1) Frame Capture

For frame capture I used the openCV library and used this tutorial:
https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

```
VideoCapture cap(1); // open the default camera is 0, usb camera is 1
   if(!cap.isOpened())  // check if we succeeded
     return -1;

//main loop of the program
   while(1)
    {
            //use openCV's VideoCapture class to get an input frame from camera
            cap >> frame;

            //get the height and width parameters
            int height  = frame.rows;
            int width = frame.cols;
…}

   cap.release();
```

2) Converting to Gray Scale

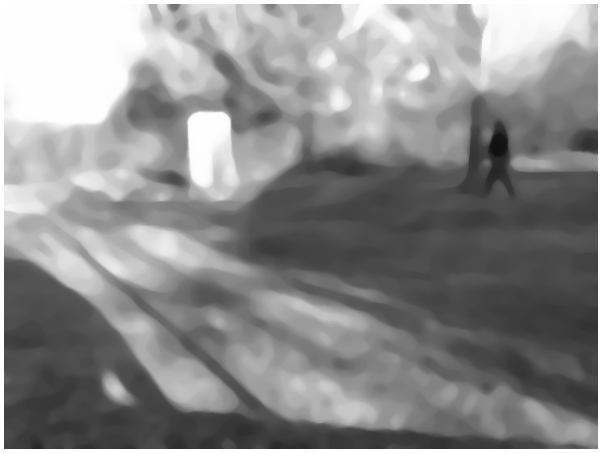
Image frames come in RGB format but for processing gray scale is used. Gray scale is just a “black and white” image which makes images a lot more easy to control and work with. A pixel is stored as a byte which has values between 0-255 with 0 being black and 255 being white. For this conversion the program uses the openCV function below:

```
//covert the input frame to a black and white frame
cvtColor(frame, gray_frame, CV_BGR2GRAY);
```

3) Image Blurring

Image blurring gets rid of edges in the image. Small movements such as leaves moving on a tree can cause a detection which is a false positive. By blurring the images we get rid of these false positives.

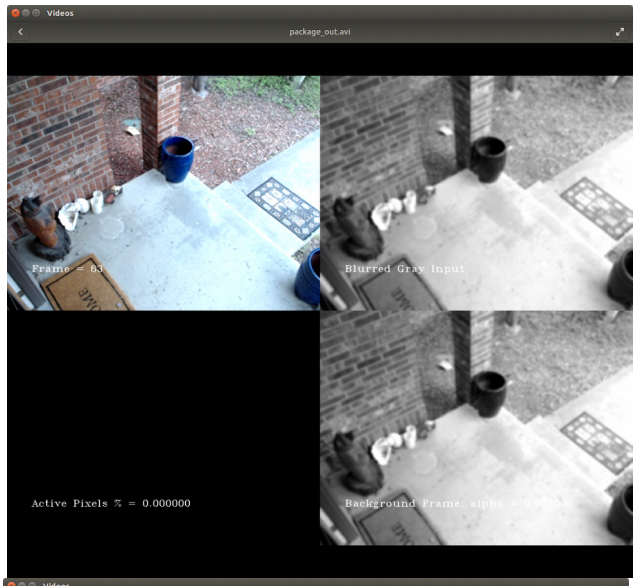
```
void image_blur(Mat & in, int N)
{
  GaussianBlur( in, in, Size( 11, 11 ), 0, 0 );
}
```

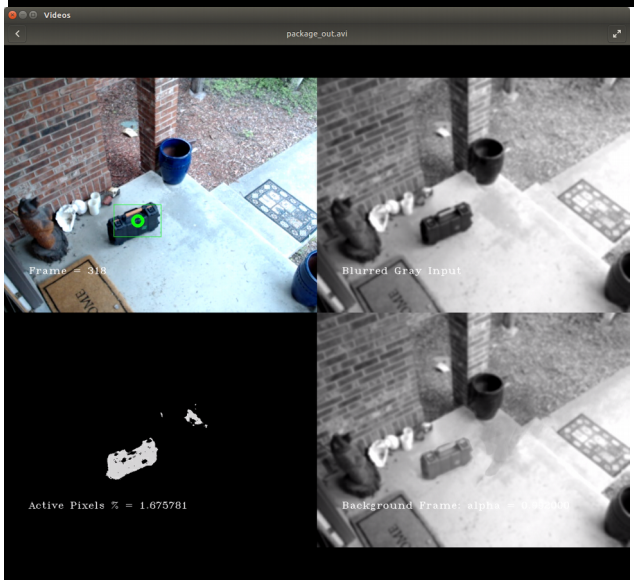


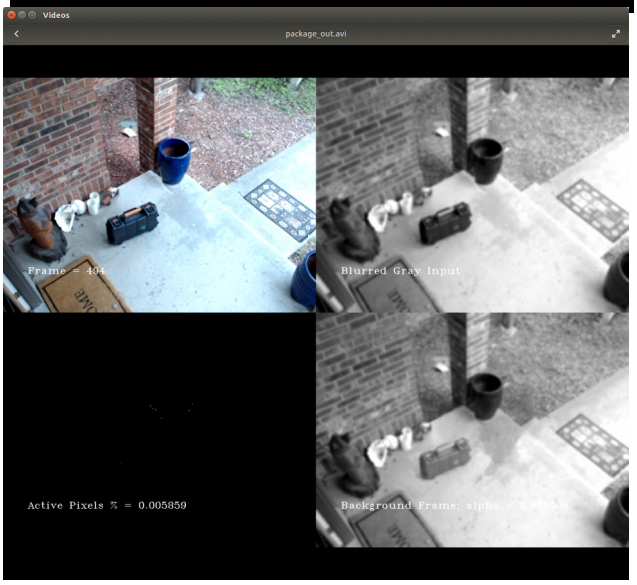
4) Creating a background frame

The background frame is initialized to the current frame. The background frame then is updated by copying the current frame periodically every “params→update_frequency” frames as long as an object is not being actively detected. At this stage if an object was left in the camera view, this would trigger a detection and the background frame would not update. This would cause continuous triggering for all frames.

To get around this issue an exponential decay filer is used. Essentially the background frame fades and the current frame slowly enters and becomes part of the background frame. Below is an demonstration of this process.







The windows are as following:
1. Top left window is current frame used for detection
2. Top right window is the current frame converted to grey frame and blurred
3. Bottom left window is the active pixels from the blurred input frame subtracting the background frame
4. Bottom right frame is the background frame

At frame 63 (top left photo) the background is set and no detection is found. At frame 207 (top right photo) a package is dropped off and an object is detected. At frame 318 (bottom left photo) the person is gone and the package is left and still being detected. At frame 494 (bottom right photo) the package has completely merged into the background and detection has stopped.

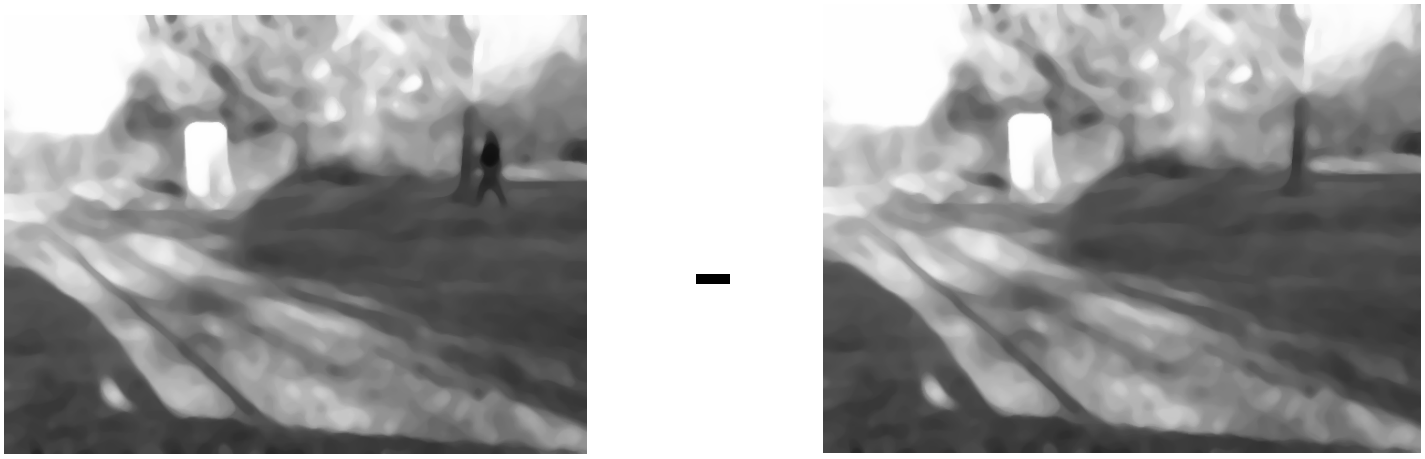
The code below is used for background updates and exponential filters.

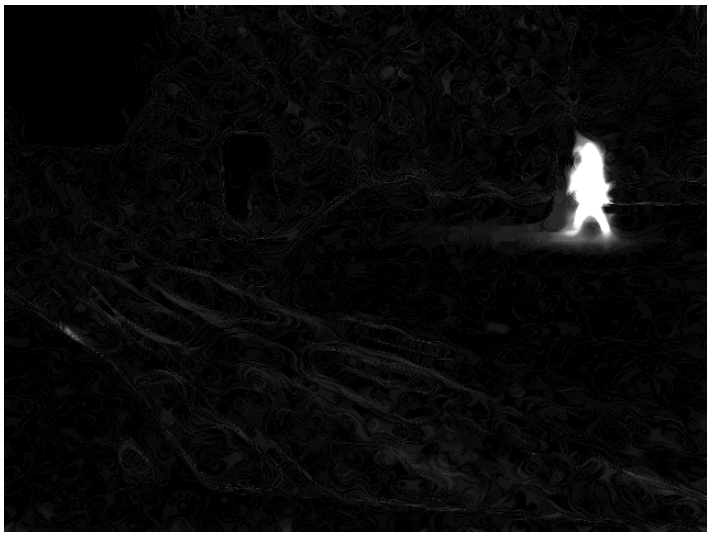
```
void update_background(Mat & input, Mat & background, motion_detect_params_t * params,
                                    int motion_flag, int frame_count)
{
  if(( (frame_count % params->update_frequency) == 0) && (!motion_flag))
    {
      background = input.clone();
      cout << "Background has changed\n";
    }

  //expontial use_exponential_filter
  float alpha = params->alpha;  //.99 - 4 seconds .999 -
  //background is an int where alpha is a float, so possible round off error
  background = alpha*background + (1-alpha)*input;
}
```

5) Subtracting the Background

After the background is set, a difference frame is made by subtracting the current frame by the background frame.

```
void subtract_background(Mat & diff_frame, Mat & input, Mat background_frame)
{
  diff_frame = abs(input - background_frame);
}
```





6) Detecting a trigger

After the subtraction between the blurred input frame and the background frame is obtained, a detection result has to be made. The features used to determine the detection are: std_y, std_x, percent of pixels that have changed.

We determine the number of active pixels by thresholding. The threshold value is determined by finding the image's mean then adding 4 standard deviations. If the image is not changing, its standard deviation and mean would be 0 which would make the threshold value 0 also. If the threshold equals 0 then any noise in the frame would act as an active pixel and be detected as an object. By setting a minimum threshold value gets rid of this problem.

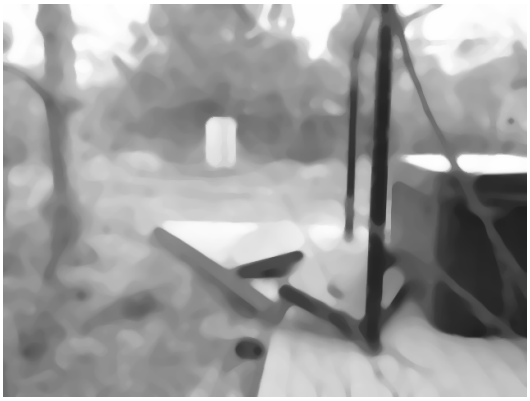
After a threshold is set, the number of active pixels is determined along with the standard deviation of x and y. Next, if the standard deviation and the number of active pixels is above the threshold, motion_flag is turned on.

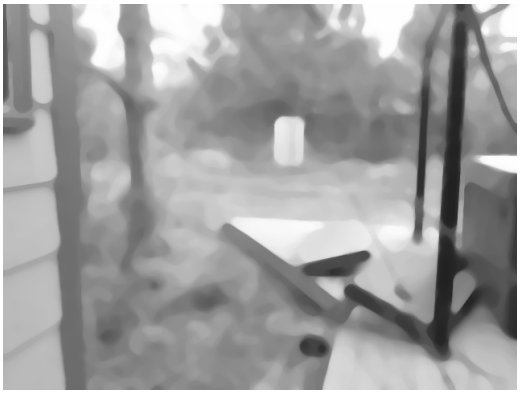
```
int make_decision(features_t * features,  motion_detect_params_t * param)
{

  //if the camera moves or a bug lands on the
  //camera lens this gets rid of the false positives
  if(features->std_x > param->std_x_thresh)
     return 0;
   if(features->std_y > param->std_y_thresh)
     return 0;

    float percent_change = features->percent_pixels_changed;
    int motion_flag = 0;

    //if the pixels changed is greater than the set threshold, say there is motion
    if(percent_change > param->pixel_percent_threshold)
      {
              motion_flag = 1;
      }
    else
      {
              motion_flag = 0;
      }

    return motion_flag;
}
```

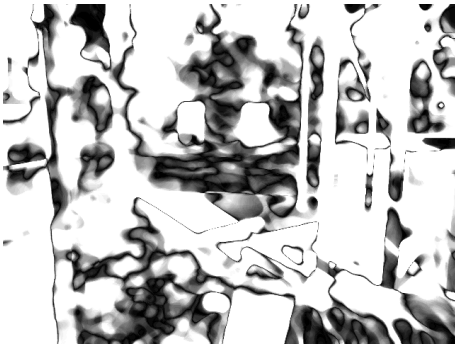
Using Standard Deviation to get rid of False Positives

Below is an example of when the whole camera moved causing a false positive because all the pixels changed. To filter out these false positives, standard deviation is used. If the standard deviation is above a threshold then object detection is false.

Grey Frame – Background Frame = diff Frame:

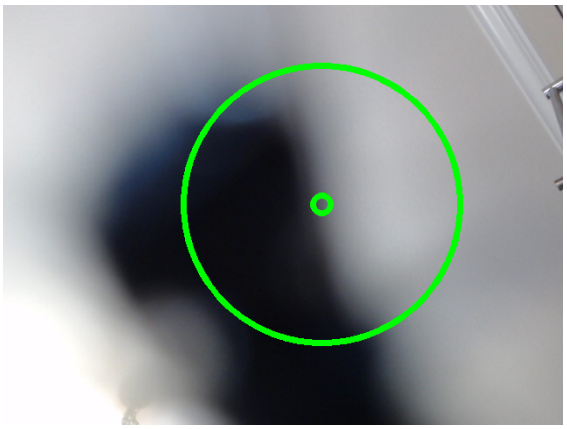






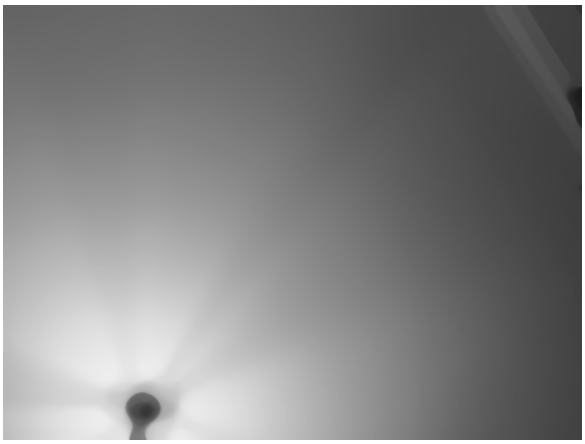
Object detected
center_x = 401 center_y = 192
Pixel changes = 58719 threshold = 100
std_x = 162.855 std_y = 139.275

False Positive: Bug Land on Lens

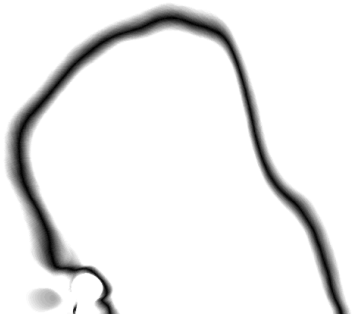


Grey Frame – Background Frame = Difference Frame:

-



=



Object detected
center_x = 364 center_y = 228
Pixel changes = 188176 threshold = 100
std_x = 178.82 std_y = 138.999

In the two above examples the standard deviation is over 140, while the example of the person walking in the backyard is roughly 25. We can use this information to get rid of false positives.

Default Settings

Function	Tuning Parameters	File
image_blur()	kernel size = 11x11	detect_motion.cc
subtract_background()	none	detect_motion.cc
extract_features()	param→std_factor = 4 param→min_pixel_diff = 25	feature_extract.cc
make_decision()	std_x_thresh = 200 std_y_thresh = 200 param→pixel_percent_threshold = 0.1	decision.cc
update_background()	params->alpha = .9899 (decay rate) params→update_frequency = 3000 frames	detect_motion.cc

https://github.com/acoopman/IndependentStudySecurityCamera