How-to Detect Motion using USB Webcam and OpenCV

This document guides the user through:

1. Installing OpenCV
2. Installing additional modules to perform tracking
3. Code to detect motion from a USB webcam

This guide builds on a previous student guide from Fall 2019 that walks users through how to stream video using OpenCV - How to stream OpenCV proccessed images over the network. This guide uses a section of that previous guide to help users view their image stream to validate the motion detection algorithm.

1. Install OpenCV

1) ssh into the beaglebone
2) Run the following commands to install openCV and video for linux libraries
   (target)$ **sudo apt-get install libv4l-dev**
   (target)$ **sudo apt-get install libopencv-dev**
3) Troubleshooting:
   - Ensure that you are installing these packages on the target (the beaglebone), not the host
   - If the "apt-get" command is failing ensure that the beaglebone is connected to the internet through Dr. Fraser's Networking Guide
   - If the "apt-get" command is failing, try updating using the command below before running the above commands
     (target)$ **sudo apt-get update**
   - To double check that the packages installed correctly, run the following command for both of the packages we installed
     (target)$ **dpkg -s <package_name>**

2. Install Additional OpenCV Modules

The sudo apt-get install libopencv-dev command we ran above installs many of the libraries needed to stream and manipulate video. However to track motion we need some additional OpenCV modules.

1) On the host, clone the Github repository at https://github.com/opencv/opencv_contrib
   (host)$ **git clone https://github.com/opencv/opencv_contrib.git**
   or
   (host)$ **git clone git@github.com:opencv/opencv_contrib.git**
2) Change directory into the modules/tracking directory
   (host)$ **cd opencv_contrib/**
   (host)$ **cd modules/**
3) Change directory into the tracking/include/opencv2 directory
   (host)$ **cd tracking/**
   (host)$ **cd include/opencv2/**
4) Copy the contents into the shared NFS folder
   (host)$ **cp -r tracking/ ~/cmpt433/public**
   (host)$ **cp tracking.hpp ~/cmpt433/public**
5) Ssh into the beaglebone and mount the NFS folder
   (target)$ **./<yourMountScript>**
6) Change directory into the shared folder
   (target)$ **cd /mnt/remote**
7) Copy the 2 openCV files into the folder where the rest of the openCV modules are installed
   (target)$ **cp -r tracking/ /usr/include/opencv2**
   (target)$ **cp tracking.hpp /usr/include/opencv2**
8) Troubleshooting:
   - If the "git clone" command is failing ensure that the beaglebone is connected to the internet through Dr. Fraser's Networking Guide
   - If using the SSH key for the github, ensure that you have git installed and set up on your host

3. Motion Detection Code

The following is code adapted from https://gist.github.com/six519/6d2beee53038ebe8abd98063abfdad86 to detect motion using the tracking.hpp module in openCV

```
#include <opencv2/opencv.hpp>
#include <opencv2/tracking.hpp>
#include <opencv2/core/ocl.hpp>
#include<opencv2/imgproc.hpp>
#include <unistd.h>
#include<iostream>
#include<time.h>

using namespace cv;
using namespace std;

#define THRESHOLD 500

int main(int argc, char **argv) {
  // Section 1
  Mat frame, gray, frameDelta, thresh, firstFrame;
  vector<vector<Point> > cnts;
  VideoCapture camera(0); //open camera

  //set the video size to 512x288 to process faster
  camera.set(3, 512);
  camera.set(4, 288);

  sleep(3);

  // Section 2
  camera.read(frame);

  //convert to grayscale and set the first frame
  cvtColor(frame, firstFrame, COLOR_BGR2GRAY);
  GaussianBlur(firstFrame, firstFrame, Size(21, 21), 0);

  // Section 3
  while(camera.read(frame)) {

    //convert to grayscale
    cvtColor(frame, gray, COLOR_BGR2GRAY);
    GaussianBlur(gray, gray, Size(21, 21), 0);

    //compute difference between first frame and current frame
    absdiff(firstFrame, gray, frameDelta);
    threshold(frameDelta, thresh, 25, 255, THRESH_BINARY);

    dilate(thresh, thresh, Mat(), Point(-1,-1), 2);
    findContours(thresh, cnts, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
    // Section 4
    for(int i = 0; i< cnts.size(); i++) {
       if(contourArea(cnts[i]) > THRESHOLD) {
           putText(frame, "Motion Detected", Point(10, 20), FONT_HERSHEY_SIMPLEX, 0.75, Scalar(0,0,255),2);
       }
    }

   // Section 5
    std::vector<uchar> buff;
   //encode to jpg
   cv::imencode(".jpg", frame, buff); //write jpg to stdout so it can be piped
   fwrite(buff.data(),buff.size(),1,stdout);
   fflush(stdout);
  }
  return 0;
}
```

To compile the above code, run the following command:

(target)$ **g++ -O2 `pkg-config --cflags --libs opencv` -lrt <source_file>.cpp -o <exe file> -lpthread**

1) Troubleshooting:
    - Ensure that you saved the file with .cpp extension
    - If the code does not compile, ensure that you have g++ installed on your beaglebone because we are not cross-compiling this code

4. Code Explanation

Section 1:

- Defines the variables for the entire program
    - The frames that we'll be comparing
    - The vector where we'll store the results of comparing the frames
    - The camera
- Initializes the camera settings

Section 2:

- Reads a single frame from the camera, all future frames will be compared to this frame to check if there is motion
- Converts the frame to grayscale for easier comparison and stores it into the variable firstFrame
- If you want to update the initial frame, you can export this section into its own function and call it periodically

Section 3:

- A while loop that will keep looping as long as the camera is able to read frames
- Converts the frame to grayscale for easier comparison
- Compares frame to firstFrame and stores result of comparison in cnts

Section 4:

- Loops over cnts and checks if any of the points are greater than a user-defined THRESHOLD
- If it is greater than the THRESHOLD, then add a message to the frame and show the frame

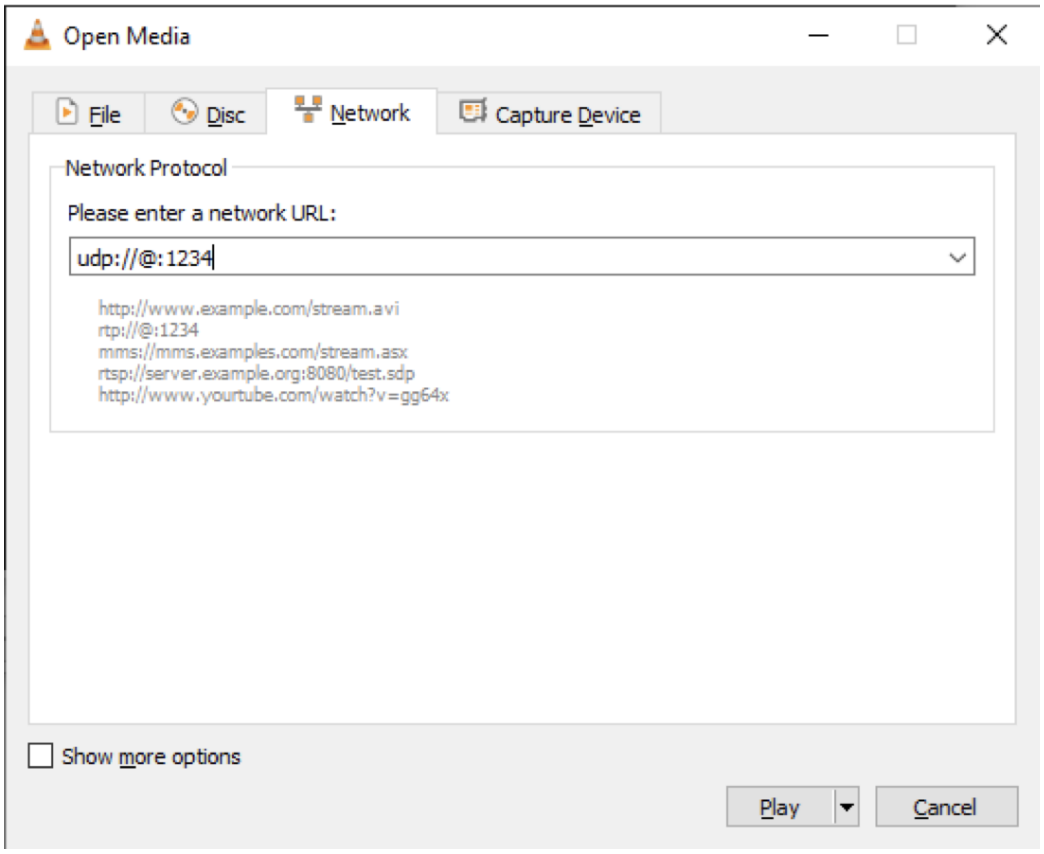
Section 5:

- Encodes the frame as jpg, and then writes the frame to stdout
- This section of code was taken from another student guide on Dr. Fraser's website from Fall 2019 - How to stream OpenCV proccessed images over the network

5. Running the Executable

This portion of the guide is taken from another student guide on Dr. Fraser's website from Fall 2019 - How to stream OpenCV proccessed images over the network, section 3. For a full explanation on how the streaming component works, you should read their guide. This guide is focused on detecting motion and uses the streamed video to validate the motion detection algorithm. A brief summary of section 3 of their guide is below.

1) To run the executable, run the following command:
   (target)$ **./<exe file> | avconv -vcodec mjpeg -r 5 -i pipe:0 -f mpegts udp://192.168.7.1:1234**
2) To view the streamed video, open VLC on your host and click on **Media > Open Network Stream and input**
3) In the text box to enter a network url, enter **udp://@:1234**



Open Media

Network Protocol

Please enter a network URL:

udp://@:1234

http://www.example.com/stream.avi
rtp://@:1234
mms://mms.examples.com/stream.asx
rtsp://server.example.org:8080/test.sdp
http://www.yourtube.com/watch?v=gg64x

4) Press **Play**