



University of Colorado
Boulder

EXERCISE 3

BY

SIDDHANT JAJOO

UNDER THE GUIDANCE OF:

PROFESSOR SAM SIEWERT,
UNIVERSITY OF COLORADO BOULDER

6/28/2019

BOARD USED: NVIDIA JETSON NANO.

TABLE OF CONTENTS

QUESTION 1	2
SUMMARY	2
QUESTION 2	3
SCREENSHOTS	3
CODE DESCRIPTION.....	5
ANALYSIS	5
QUESTION 3	6
SCREENSHOTS	6
CODE DESCRIPTION.....	6
ANALYSIS	6
QUESTION 4	7
SCREENSHOTS	7
CODE DESCRIPTION.....	8
ANALYSIS	8
QUESTION 5	8
SCREENSHOTS	8
CODE DESCRIPTION.....	8
ANALYSIS	9
QUESTION 6	9
SCREENSHOTS	9
CODE DESCRIPTIONS.....	10
ANALYSIS	11
REFERENCES	11

QUESTION 1

SUMMARY

There has been a rapid shift from analog to digital media. This is nonetheless due to the great advantages of digital media and even the demand from the current viewers. The Analog services are now a long lost thing which has been replaced by digital media that is on-demand and has integrated a lot of other services in its course of development. The advent of digital media systems has paved way to interactive media systems such as Augmented Reality (AR). The paper discusses about the development of applications for such interactive media systems. Some of the interactive media system tools and methods mentioned in the paper include Mobile-to-cloud operating systems, digital video and audio encoding, graphical rendering and digital video frame annotation, advanced user interaction and sensing.

Uncompressed video frames are required to process for machine vision related applications but at the same time these videos need to be uploaded to the cloud for cloud analytics. This leads to the videos being compressed in order to transport them. In order to achieve encoding MPEG is used which has good compression ratios. MPEG2 is the technology being used currently and is slowly being replaced by MPEG4 which has a better compression ratio and better digital video quality. The MPEG compression takes place in such a way that: firstly it subsamples red and blue color compared to the green color. The entire frame is divided into blocks of 8x8 which are called as macroblocks. DCT is then applied to each of these macroblocks to separate the lower and higher frequency components. The DC component of the macroblock is stored in the upper left area of the macroblock and the higher components are stored in the bottom right corner of the block. After the DCT transformation is obtained, it is scaled and quantized to eliminate the higher frequency components. Then zigzag lossless compression of each macroblock is done. In this way the image is encoded to transfer. The decoding takes place in the exact reverse order.

It is not possible to recover the entire image after encoding it but ideally we only require key features to understand the scene. Efforts are being made to fully understand a surrounding and recognize objects. There are possible scenarios in which machine vision can be superior to the human vision. These scenarios are where AR can come into picture. The videos can be annotated in real time about certain things that the human eye cannot see.

This is the very future of machine vision and AR is an application. In order to make all this happen, better compression techniques are so that AR is possible in as small as goggles. This paper shows us the importance of machine vision, interactive media systems and the related tools and technologies that can bring about the change in future.

QUESTION 2

SCREENSHOTS

Grabbing a frame using ffmpeg:

```
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 2$ ffmpeg -i Dark-Room-Laser-Spot.mpeg -ss 00:00:06.000 -vframes 1 before_median_filter.ppm
ffmpeg version 3.4.6-0ubuntu0.18.04.1 Copyright (c) 2000-2019 the FFmpeg developers
  built with gcc 7 (Ubuntu/Linaro 7.3.0-16ubuntu3)
  configuration: --prefix=/usr --extra-version=0ubuntu0.18.04.1 --toolchain=hardened --libdir=/usr/lib/aarch64-linux-gnu --incdir=/usr/include/aarch64-l
nux-gnu --enable-gpl --disable-stripping --enable-avresample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libass --enable-libbluray --enab
e-libbs2b --enable-libcaca --enable-libcdio --enable-libflite --enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable-libgme --enable-l
ibgsm --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librubberband --enable-
librsync --enable-libshine --enable-lbsnappy --enable-libsoxr --enable-lspspeex --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvorbis
--enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzmq --enable-libzvi --enable-omx --en
able-opengl --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --enable-frei0r --enable-libopen
v --enable-libx264 --enable-shared
libavutil      55. 78.100 / 55. 78.100
libavcodec     57.107.100 / 57.107.100
libavformat    57. 83.100 / 57. 83.100
libavdevice    57. 10.100 / 57. 10.100
libavfilter    6.107.100 / 6.107.100
libavresample  3.  7.  0 / 3.  7.  0
libswscale     4.  8.100 / 4.  8.100
libswresample  2.  9.100 / 2.  9.100
libpostproc   54.  7.100 / 54.  7.100
Input #0, mpeg, from 'Dark-Room-Laser-Spot.mpeg':
  Duration: 00:00:42.14, start: 0.500000, bitrate: 1431 kb/s
    Stream #0:0[0x1e0]: Video: mpeg1video, yuv420p(tv), 1920x1080 [SAR 1:1 DAR 16:9], 104857 kb/s, 29.97 fps, 29.97 tbr, 90k tbn, 29.97 tbc
    Stream #0:1[0x1c0]: Audio: mp2, 44100 Hz, stereo, s16p, 224 kb/s
Stream mapping:
  Stream #0:0 -> #0:0 (mpeg1video (native) -> ppm (native))
Press [q] to stop, [?] for help
[swscale @ 0x55ac1d2a20] No accelerated colorspace conversion found from yuv420p to rgb24.
Output #0, image2, to 'before_median_filter.ppm':
  Metadata:
    encoder      : Lavf57.83.100
  Stream #0:0: Video: ppm, rgb24, 1920x1080 [SAR 1:1 DAR 16:9], q=2-31, 200 kb/s, 29.97 fps, 29.97 tbn, 29.97 tbc
  Metadata:
    encoder      : Lavc57.107.100 ppm
Frame= 1 fps=0.3 q=-0.0 Lsize=N/A time=00:00:00.03 bitrate=N/A speed=0.0111x
video:6075kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: unknown
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 2$
```

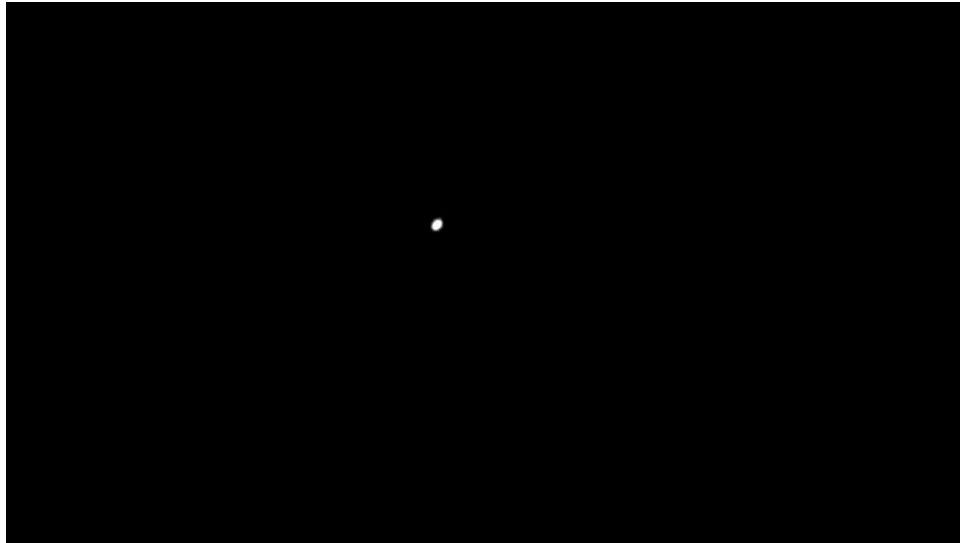
The frame before applying median filter:



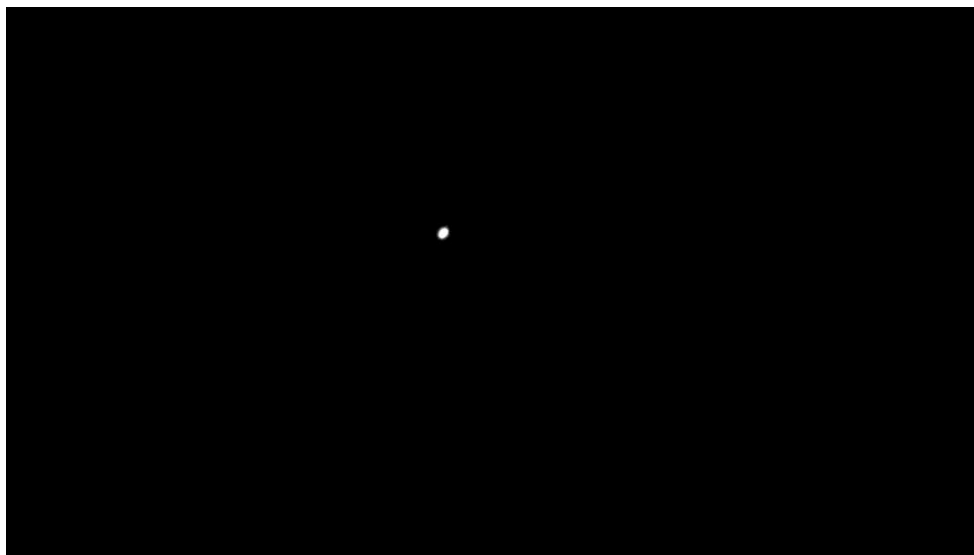
Run Median Filter Program:

```
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 2$ make to obtain G band grayscale image
g++ -O0 -g -c median_filter.cpp -I /usr/lib/opencv/include
g++ -O0 -g -o median_filter median_filter.o `pkg-config --libs opencv` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 2$ ./median_filter
Input Image as an argument.
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 2$ ./median_filter Screenshots\ and\ Images\
before_median_filter.jpg before_median_filter.ppm get_fram_ppm.png gwt_fame.jpg.png
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 2$ ./median_filter Screenshots\ and\ Images\before_median_filter.ppm
Gtk-Message: 21:15:43.039: Failed to load module "canberra-gtk-module"
```

Frame with G band:



Median Filter Screenshot:



CODE DESCRIPTION

In this code, the image name is taken as a command line argument to apply the median filter on the respective image. The image is read using `imread()` as a colour image. This colour image has 3 different channels namely BGR. In order to obtain the G band, the image is split into three grayscale images with single channels. The three split images have pixel values of B, G and R respectively. This is achieved by using the API `split(src, dest)`. Median filter is then applied to the G band image obtained using the `medianBlur(src, dest, kernel matrix size)` API call. The obtained image is then display on a window using `imshow(window_name, img)` and saved using `imwrite(img)` API command.

Finally the code exits on pressing ESC button while the window is open. On pressing the ESC button, the memory allocated by the window is freed up.

ANALYSIS

Application

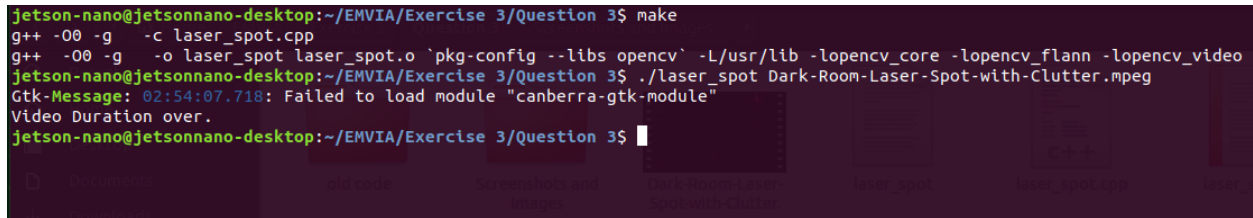
The Median Filter in this section was used to reduce the overall noise in the image. It also helped to smoothen the image overall by replacing the pixel values by the median of the values of its neighbouring pixels. The Median Filter is used in object detection in order to reduce the noise surrounding the target object. It can be used in edge detection and object tracking so that the edges can be smooth and we can get a stark contrast in the background and the object to detect it continuously.

Did the Median Filter help enhance the laser spot edge boundary at all?

As the Median filter successfully reduced and smoothened the overall noise in the image. It also did help to smooth and enhance the edge boundary of the laser spot. The screenshots of the image are attached above. The difference before the median filter and after applying median filter can be observed by zooming it as required. Ideally it should smoothen the boundary/edge if one understands the concept and working about median filter.

QUESTION 3

SCREENSHOTS



```
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 3$ make
g++ -O0 -g -c laser_spot.cpp
g++ -O0 -g -o laser_spot laser_spot.o `pkg-config --libs opencv` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 3$ ./laser_spot Dark-Room-Laser-Spot-with-Clutter.mpeg
Gtk-Messsage: 02:54:07.718: Failed to load module "canberra-gtk-module"
Video Duration over.
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 3$
```

CODE DESCRIPTION

This code was written to eliminate the background clutter in the video and preserve only the moving laser spot. The code takes input video as a command line argument. Initially VideoCapture and VideoWriter objects are created to read a video and write a video respectively. In order to obtain frame differencing, the first frame is read from the input video file and is cloned in a Mat image as a reference to subtract from. The next image is then read. This image is subtracted from the previous stored image in order to obtain the foreground object. i.e the laser spot. This current image that is read is then cloned and stored in a Mat image for repeating the same steps above. In order to write the video, the video properties are copied from the input video and all the frames are written one by one a while loop after frame differencing using the object.write(image).

ANALYSIS

Frame differencing has given acceptable results for the video. It has eliminated all the clutter in the background preserving the laser spot in the subsequent frames. Since, it was a dark background video, the differencing between the frames yielded zero pixel intensity values. There was a little noise in the image but the intensity values of these noise pixels were less than the laser spot thus it did not interfere with the algorithm. In order to eliminate noise, filters can be applied. These filters would be effective, beneficial and important in scenarios where there is more light in the background.

The re-encoded video can be found in a zip folder along with the code.

QUESTION 4

SCREENSHOTS

Run Code:

```
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 4$ make
g++ -O0 -g -c grayscale.cpp
g++ -O0 -g -o grayscale grayscale.o `pkg-config --libs opencv` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 4$ ./grayscale Dark-Room-Laser-Spot.mpeg
Gtk-Messsage: 15:54:30.821: Failed to load module "canberra-gtk-module"
[1]

VIDEO WINDOW
```

Rencode Video Command:

```
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 4$ ffmpeg -i Grayscale_PGM_frames/grayscale_frame_0d.pgm ffmpeg_reencode.mpeg
ffmpeg version 3.4.6-0ubuntu0.18.04.1 Copyright (c) 2000-2019 the FFmpeg developers
  built with gcc 7 (Ubuntu/Linaro 7.3.0-16ubuntu3)
  configuration: --prefix=/usr --extra-version=0ubuntu0.18.04.1 --toolchain=hardened --libdir=/usr/lib/aarch64-linux-gnu --incdir=/usr/include/aarch64-linux-gnu --enable-gpl --dis
vresample --enable-avisynth --enable-gnutls --enable-ladspa --enable-libass --enable-libbluray --enable-libsbs2b --enable-libcaca --enable-libcdio --enable-libflite --enable-libfor
type --enable-libfrbidi --enable-libgme --enable-libgsm --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable
librsync --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvorbis --enable-libvpx --enable-l
p --enable-libx265 --enable-libx264 --enable-libxvid --enable-libzmq --enable-libzvt --enable-omx --enable-opengl --enable-sdl2 --enable-libdc1394 --enable-libdrm
enable-chromaprint --enable-freetype --enable-libopencv --enable-libx264 --enable-shared
  libavutil      55. 78.100 / 55. 78.100
  libavcodec     57.107.100 / 57.107.100
  libavformat    57. 83.100 / 57. 83.100
  libavdevice    57. 10.100 / 57. 10.100
  libavfilter     6.107.100 / 6.107.100
  libavresample   3. 7. 0 / 3. 7. 0
  libswscale     4. 8.100 / 4. 8.100
  libswresample   2. 9.100 / 2. 9.100
  libpostproc    54. 7.100 / 54. 7.100
Input #0, image2, from 'Grayscale_PGM_frames/grayscale_frame_0d.pgm':
  Duration: 00:00:50.40, start: 0.000000, bitrate: N/A
  Stream #0:0: Video: pgm, gray, 1920x1080, 25 fps, 25 tbr, 25 tbn, 25 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (pgm (native) -> mpegivideo (native))
Press [q] to stop, [?] for help
[mpeg @ 0x5591c95860] VBV buffer size not set, using default size of 130KB
If you want the mpeg file to be compliant to some specification
Like DVD, VCD or others, make sure you set the correct buffer size
Output #0, mpeg, to 'ffmpeg_reencode.mpeg':
  Metadata:
    encoder      : Lavf57.83.100
  Stream #0:0: Video: mpegivideo, yuv420p, 1920x1080, q=2-31, 200 kb/s, 25 fps, 90k tbn, 25 tbc
  Metadata:
    encoder      : Lavc57.107.100 mpegivideo
  Side data:
    cpb: bitrate max/min/avg: 0/0/200000 buffer size: 0 vbv_delay: -1
[mpegivideo @ 0x5591c96760] warning, clipping 1 dct coefficients to -255..255
Last message repeated 2 times
frame= 8 fps=0.0 q=2.0 size= 34kB time=00:00:00.20 bitrate=1392.6kbts/s[mpegivideo @ 0x5591c96760] warning, clipping 1 dct coefficients to -255..255
Last message repeated 1 times
frame= 14 fps= 13 q=1.6 size= 68kB time=00:00:00.44 bitrate=1266.0kbts/s[mpegivideo @ 0x5591c96760] warning, clipping 1 dct coefficients to -255..255
frame= 20 fps= 13 q=2.0 size= 72kB time=00:00:00.68 bitrate= 867.4kbts/s[mpegivideo @ 0x5591c96760] warning, clipping 1 dct coefficients to -255..255
Last message repeated 2 times
frame= 26 fps= 12 q=1.6 size= 106kB time=00:00:00.92 bitrate= 943.8kbts/s[mpegivideo @ 0x5591c96760] warning, clipping 1 dct coefficients to -255..255
Last message repeated 5 times
```


CODE DESCRIPTION

In this code, it was required to convert the entire video to a graymap by using the G band and then re-encode the frames to an mpeg4 video. The intermediate frames were required to be saved as .pgm images. The G band graymap was obtained by splitting the frame obtained from the video. The splitting was done in the same way as done in the 2nd question. The frames obtained by splitting the image were then re-encode in mpeg4 format using VideoWriter as done in the 3rd question.

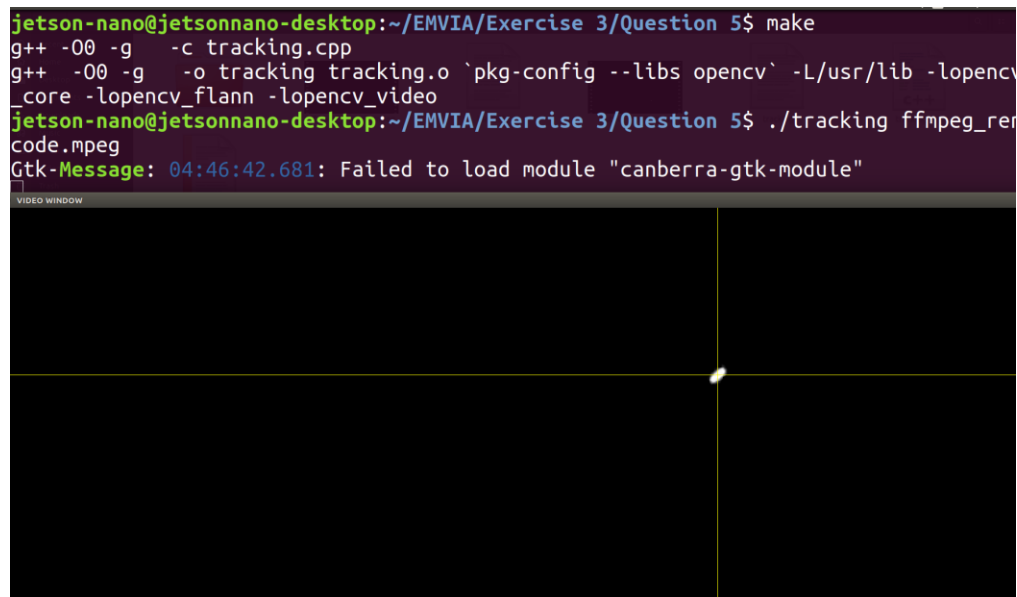
ANALYSIS

The video was successfully reencoded and was subsequently used in the 5th Question as an input video.

The re-encoded video and the .pgm images can be found in a zip file along with the code.

QUESTION 5

SCREENSHOTS



CODE DESCRIPTION

In this code, a threshold function is written to find the laser spot and find the extreme x and y coordinates on the laser spot. The input video used here is the one which was reencoded in question 4.

The steps involved to track the laser spot are:

1. Read the frame and access the G band.
2. Apply median filter to the G band in the frame to reduce the Noise in the frame.
3. Apply threshold function to obtain the coordinates of the laser spot by choosing an appropriate threshold value.

4. Compute the COM of the laser spot.
5. Draw a yellow crosshair on the laser spot through the COM.
6. Create an annotated output video using the above computed frames.

All the above steps are carried out in a while loop till the end of video. No opencv functions were used in order to execute thresholding and compute COM.

ANALYSIS

In order to obtain laser spot tracking with precision, a median filter was applied in order to remove all the noise. The median filter easily removed all the stray noise from the frames. Thus we were only left with the laser spot in the whole frame with high intensity pixel values. Now a thresholding function was applied to find the x and y coordinates of only the laser spot pixels. This thresholding function would eliminate the unnecessary computation of the darker pixels. After acquiring the required x and y coordinates, the xbar and ybar are calculated using formulas. This was a simple straightforward algorithm to find the COM and track the laser spot. Removal of noise, applying the threshold function and finally calculating the COM are the only steps involved. The video uploaded shows that the algorithm is 100% accurate for tracking a laser spot in a dark room.

The re-encoded video can be found in a zip file along with the code.

QUESTION 6

SCREENSHOTS

```
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 6$ make
g++ -O0 -g -c tracking_clutter.cpp
g++ -O0 -g -o tracking_clutter tracking_clutter.o `pkg-config --libs opencv` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video
jetson-nano@jetsonnano-desktop:~/EMVIA/Exercise 3/Question 6$ ./tracking_clutter Light-Room-Laser-Spot-with-Clutter.mpeg
Gtk-Message: 22:01:56.683: Failed to load module "canberra-gtk-module"
```



CODE DESCRIPTIONS

This code uses the same thresholding function, the noise removal filter such as the median filter, COM calculating strategy. The input video here is a different one as compared to the previous function. The video has much more light as compared to the video in the 5th question. So it becomes a little difficult to track the laser spot as compared to the one in the 5th question. In order to achieve tracking, a few additions have been done to the algorithm.

The steps involved to track the laser spot are:

1. Read the frame.
2. Apply `cvtColor` to obtain the grayscale image in order to reduce the pixel values of RGB.
3. Apply frame differencing algorithm to eliminate the background of the image so that only the foreground objects are visible.
4. Convert the image frame to a binary image in order to remove intermediate pixel intensity values.
5. Apply median filter to the frame to reduce the Noise in the frame. The kernel chosen here was size 5.
6. Apply threshold function to obtain the coordinates of the laser spot. The threshold function here is slightly different as compared to the previous one. Here it checks for the highest pixel value i.e 255 since the frame has been converted to a binary image.
7. Compute the COM of the laser spot.
8. Draw a yellow crosshair on the laser spot through the COM.
9. Create an annotated output video using the above computed frames.

All the above steps are carried out in a while loop till the end of video. No opencv functions were used for frame differencing, converting the image to a binary image, thresholding and Computing COM.

ANALYSIS

This algorithm was a little different as compared to the previous one. This was because the video was brighter as compared to the previous video. So much more noise and higher intensity pixel values had to be handled. Now this video was a colour one with clutter, the first step is obviously to convert it to grayscale in order to reduce computations at a later stage. This was achieved using `cvtColor`. After converting the image to a grayscale image, frame differencing was carried out to eliminate all the background objects. This would eliminate the interference with the laser spot. This was necessary to convert common pixel values to zero. This step was one of the most important steps in object tracking. After removal of background and preserving the laser spot, there were still some areas in the image other than the laser spot that were bright and would have varying pixel intensity values. Just applying a median filter was not sufficient to eliminate all the noise from the image because noise had varying levels of pixel value. The median filter as the names suggests would settle for a median value when an appropriate kernel would be applied. In order to eliminate all the noise, the image was first converted into a binary image. The binary image has only two levels 0 and 255. This eliminates all the intermediate pixel values. Now, applying median filter to this binary image would eliminate most of the noise since most of the neighbouring pixel values are zero. The next step is to apply the thresholding function which would compute the x and y coordinates of only those pixels which have 255 as their pixel value thus indicating the laser spot. The next steps include finding COM, annotating the frame to track the COM.

Employing the above techniques improved the laser spot tracking to a great extent on the light video as compared to the previous algorithm. The video uploaded shows that it almost accurately detects the laser spot unless there is a completely white background. This is because the difference between the grayscale laser and the background in frame differencing is close to zero which is below threshold and thus fails to detect and compute COM. If I lower the threshold in order to cover this up, there are unnecessary inclusions of white spots which are now above the threshold and the output is much worse.

This may or may not have been resolved by using the OpenCv threshold and moment functions. I tried to implement it without using those and did get satisfactory results.

The re-encoded video can be found in a zip file along with the code.

REFERENCES

- <https://opencv.org/>