ACKNOWLEDGEMENT

It is a matter of great pleasure and privilege for me to present this report on the basis of the knowledge gained by me during internship period at Sanskrut Corporation, Udaipur during May-June, 2018.

I use this opportunity to express gratitude and debtness to Dr. Navneet Kumar Agrawal sir, Training Incharge (ECE), CTAE, UDAIPUR.

I want to express my thanks to Mr. Sandeep Koduri, CEO of Sanskrut Corporation for giving me valuable time and kind co-operation and other employees for their further co-operation to gain better knowledge.

Parth Jai
Third year, ECE

Table of Contents

Introduction to the Programming Environment

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source softwareand has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

Another well-known alternative is MATLAB. MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of

user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

Introduction to Image Processing and Computer Vision

Before getting into the subject of image processing and computer vision, one must first understand what "image" is and how image processing is different from computer vision. What is the fine line that separates the two.

An image can be thought of as:

(i) A 2-D array of numbers ranging from some minimum to some maximum.

(ii) A function I of x and y i.e I(x,y)

(iii) Something generated by a camera.

We think of an image as a function, f or I, from R^2 to R where f(x,y) gives intensity or value at position (x,y). For practical purposes, we define an image over a rectangle with a finite range such that:

$$f: [a, b] \times [c, d] \rightarrow [min, max]$$

A colour image is just three functions "stacked" together. We can write this as a "vector-valued" function:

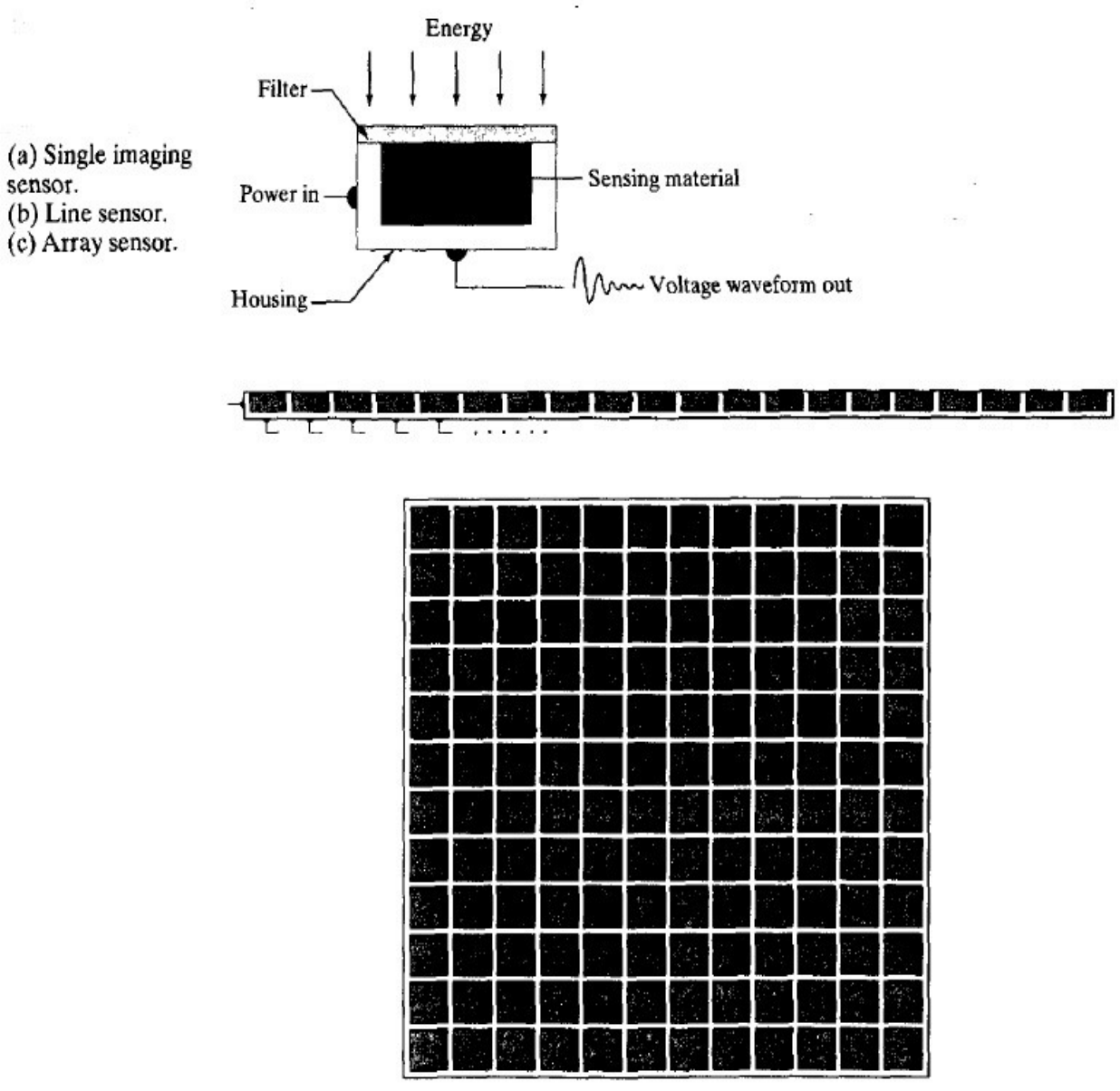
$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

In other words, an image is a collection of intensities at different location. For colour images, value at any co-ordinate is a vector. In our definition of image, if (x,y) are finite and the intensity or gray level of a point is also finite, then the image is called digital image. The field of digital image processing refers to processing digital images by means of a digital computer. A digital image is composed of a finite number of elements each of which has a particular location and value. These elements are referred to as picture elements, image elements, pels or pixels.

Image formation:

The types of the image in which we are interested in are generated by the combination of an "illumination" source and the reflection or absorption of energy from that source by the elements of the "scene" being imaged. We enclose illumination and scene in quotes to emphasize the fact that they are considerably more general than the familiar situation in which a visible light source illuminates a common everyday 3-D scene. For example, the illumination may originate from a source of electromagnetic energy such as radar, infrared, or X-ray energy. But it could also originate from less traditional sources such as ultrasound or even a computer-generated illumination pattern. Similarly, the

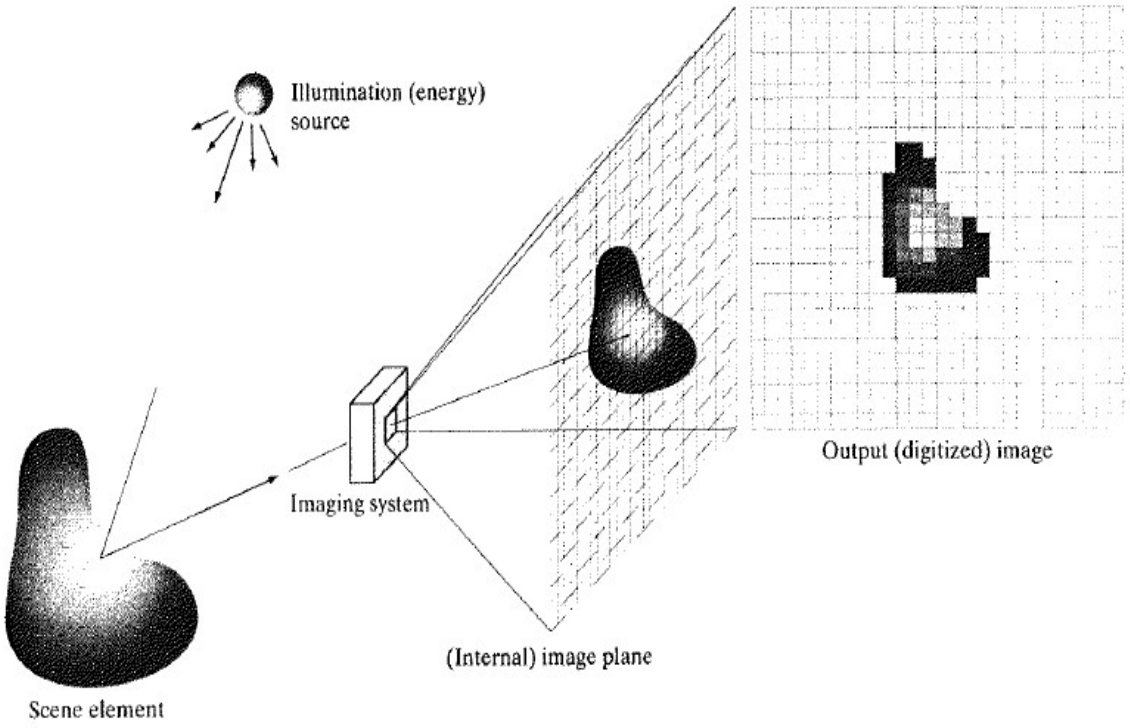
scene elements could be familiar objects, but they can just as easily be molecules, buried rock formations, or a human brain. Depending on the nature of the source, illumination energy is reflected from, or transmitted through, objects. An example in the second category is when X-rays pass through a patient's body for the purpose of generating a diagnostic X-ray film. The figure below shows the three principal sensor arrangements used to transform illumination energy into digital images. The idea is simple: incoming energy is incident upon the sensing material which outputs an analog voltage depending on the level of illumination incident.



(a) Single imaging sensor.
(b) Line sensor.
(c) Array sensor.

The figure given below shows individual sensors arranged in the form of a 2-D array. Numerous electromagnetic and some ultrasonic sensing devices frequently are arranged in an array format. This is

also the predominant arrangement found in the digital cameras. A typical sensor for these cameras is a CCD array which can be manufactured with a broad range of sensing properties and can be packaged in rugged arrays of 4000 X 4000 elements or more. The response of each sensor is proportional to the integral of the light energy projected onto the surface of the sensor, a property that is used in astronomical and other applications requiring low noise images.



An example of the digital image acquisition process. (a) Energy ("illumination") source. (b) An element of a scene. (c) Imaging system. (d) Projection of the scene onto the image plane. (e) Digitized image.

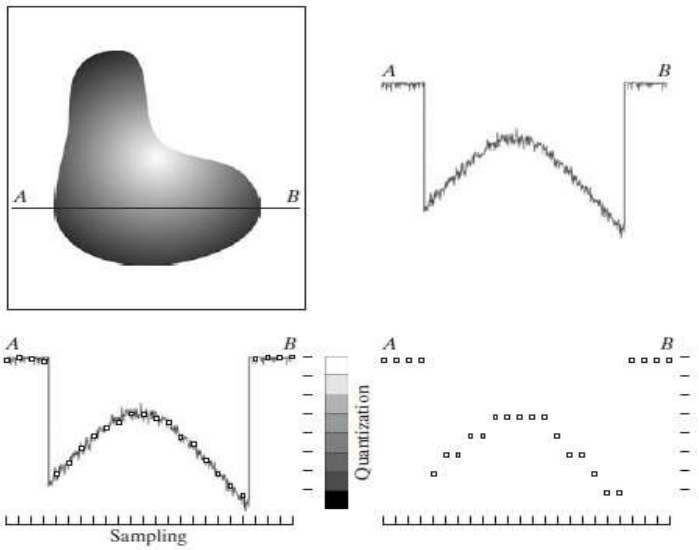
Actually there are numerous ways of acquiring an image, but our objective in all is the same: to generate digital images from sensed data. The output of most sensors is a continuous voltage waveform whose amplitude and spatial behaviour are related to the physical phenomenon being sensed. To create a digital image, we need to convert the continuous sensed data into digital form. This involves two processes: sampling and quantization.

Sampling and quantization in images:

The basic idea behind sampling and quantization is illustrated. An image may be continuous with respect to the *x*- and *y*-coordinates, and also in amplitude. To convert it to digital form, we have to sample the

function in both coordinates and in amplitude. Digitizing the coordinate values is called *sampling*. Digitizing the amplitude values is called *quantization*. The one-dimensional function in fig is a plot of amplitude (intensity level) values of the continuous image along the line segment *AB* in fig. The random variations are due to image noise.To sample this function, we take equally spaced samples along line *AB*, as shown in figThe spatial location of each sample is indicated by a vertical tick mark in the bottom part of the figure. The samples are shown as small white squares superimposed on the function. The set of these discrete locations gives the sampled function. However, the values of the samples still span (vertically) a continuous range of intensity values. In order to form a digital function, the intensity values also must be converted (*quantized*) into discrete quantities. The right side of fig shows the intensity scale divided into eight discrete intervals, ranging from black to white.The vertical tick marks indicate the specific value assigned to each of the eight intensity intervals.The continuous intensity levels are quantized by assigning

one of the eight values to each sample.The assignment is made depending on the vertical proximity of a sample to a vertical tick mark. The digital samples resulting from both sampling and quantization are shown in fig. Starting at the top of the image and carrying out this procedure line by line produces a two-dimensional digital image. It is implied in fig that, in addition to the number of discrete levels used, the accuracy achieved in quantization is highly dependent on the noise content of the sampled signal. Sampling in the manner just described assumes that we have a continuous image in both coordinate directions as well as in amplitude. In practice, the method of sampling is determined by the sensor arrangement used to generate the image.



a b
c d

Generating a digital image. (a) Continuous image. (b) A scan line from *A* to *B* in the continuous image, used to illustrate the concepts of sampling and quantization. (c) Sampling and quantization. (d) Digital scan line.

Computer Vision: meaning and importance:

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. "Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding." As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

The computer vision discipline was developed in the wake of artificial intelligence. The term was meant to mimic the human visual system, as a stepping stone to endowing robots with intelligent behaviour. What distinguishes computer vision from image processing is that in computer vision, we are looking to extract useful features for analysis purposes. So basically, digital image processing is image in, image out while computer vision is image in and some useful features or information (knowledge of the scene) form image, out. Image processing is mainly focused on processing raw images and preparing them for other tasks such as in Computer Vision. Understanding the meaning of those images is a Computer Vision task.

Computer Vision tries to do what a human brain does with the retinal input, it includes understanding and predicting the visual input. That could consist of segmentation, recognition, reconstruction (3D) and prediction (over video data). These give us the overall scene understanding.

Classically, many Computer Vision algorithms employed image processing and machine learning or sometimes other methods (e.g Variational Methods, Combinatorial approaches,...) to do the mentioned tasks. For example they used Image Processing techniques such as Edge Detection (e.g. Sobel Filter) to create Image Descriptors (e.g. SIFT) and then fed them to a Machine Learning algorithm to classify (for a recognition task).

Computer Vision and signal processing:

The computer vision holds a strong relation with the field of signal processing. Many methods for processing of one-variable signals, typically time-varying signals, can be extended in a natural way to processing of two-variable signals or multi-variable signals in computer vision. However, because of the

specific nature of images there are many methods developed within computer vision which have no counterpart in processing of one-variable signals. Together with the multi-dimensionality of the signal, this defines a subfield in signal processing as a part of computer vision. In signal processing, great importance is given to the frequency representation of a signal. Similarly, images can also be thought as a signal instead of a data structure. In signal processing, frequency domain of a signal represents the power of the various frequency components in it. In case of images, thee frequency represents how fast the features of the image are changing when one moves from one point to other in an image.

For example, consider an image in which geometrical transitions are present. In such an image, the boundary that separates the two different geometries forms the high frequency component of an image. So if an image has a lot of geometrical transitions, then it has large high frequency components, both in number and magnitude. On the other hand, if there are gradual transitions, then low frequency components are present. The more gradual the transitions are, the less will be the high frequency components present.

So we see that high frequency components give useful information about the edges in the image. The concepts of signal processing can be applied to images too as image is nothing but a 2-D function of space as opposed to 1-D function of time in signal processing. In this way, there is a strong relation between signal processing and computer vision.

Also, all the so-called processing in images is done by passing the image through filters. For obtaining different features from image, we may pass the image through wide variety of filters. The action of these filters is same as that in the case of signal processing – convolution. So the filters in digital image processing are actually linear so that convolution can be performed. We design a window of a filter. This window is actually an odd ordered matrix. This matrix is slided over the entire image to obtain the output of the filter. This is pretty much what we do with 1-D time varying signals in signal processing.

Object Tracking

Now a day, video surveillance is a part of our day to day life. In every private institute, company, government hospitals, offices, school, colleges, everywhere we need object tracking system for security purpose. Visual monitoring of activities using cameras automatically without human intervention is a challenging problem. Moving object detection is very important in intelligent surveillance. In this paper, an improved algorithm based on frame difference is presented for moving object detection. The method of motion detection and tracking is background subtraction. This paper presents a new object tracking model that systematically combines region and shape features. We design a new object detector for accurate and robust tracking in low-contrast, in noisy environment and complex scenes, which usually appear in the commonly used surveillance systems.

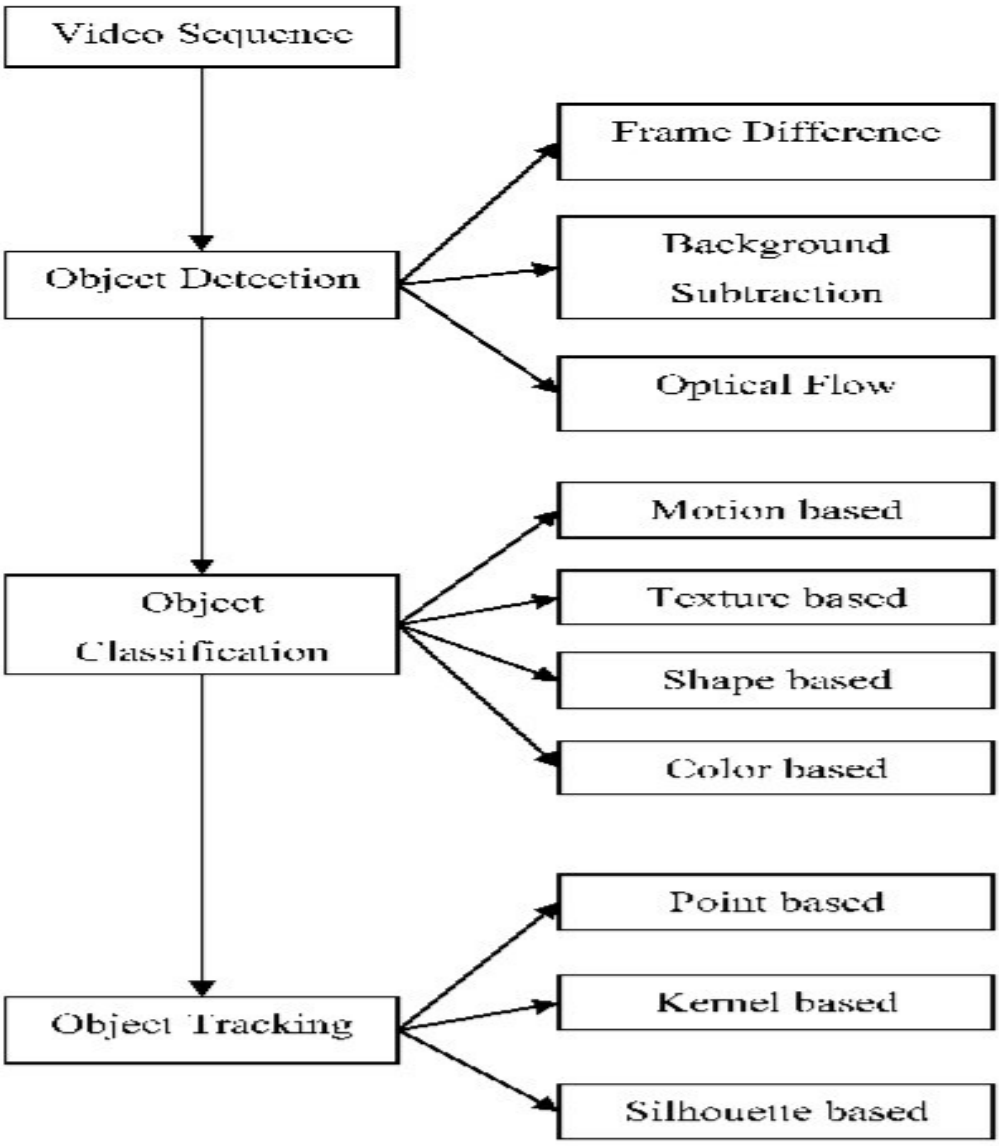
Tracking a moving object in a video sequence is an important application of image sequence analysis. Tracking is useful in many problems like collision avoidance, surveillance, gesture recognition, distance education with live teacher, searching sport clips, etc. In most tracking applications, a portion of an object of interest is identified in the first frame and we need to track its position through the sequence of images. Ideally, if the object to be tracked can be modelled well so that its presence can be inferred by detecting some feature sets in each frame we can look for objects with required features. In many scientific and commercial applications, it is usually necessary to predict what an object might be doing soon. Difficulties in object tracking

- Abrupt object motion.
- Changing appearance patterns of both the object and the scene.
- Non-rigid object structures.
- Object-to-object and object-to-scene occlusions.
- Camera motion Tracking.

Motion detection is the process of confirming a change in position of an object related to its surrounding or the change in the surrounding relative to an object. Motion detection helps to save CPU time since the region of investigation is narrowed. Object detection is the process of detecting and spotting object in an image. Object detection is a process of scanning an image for an object of interest like people, faces, computers, robots or any object.

Methodology:

There are various methods to perform object tracking. Each method has its own advantages and disadvantages. The choice particularly depends on the background, camera, colour of the object, accuracy needed, computational power etc. The following block diagram shows the different methods:



The techniques which are used in the project will be discussed here.

1. **Colour Based:**

   Colour is easily acquired and is relatively stable under viewpoint variations, unlike other image features. Low computational cost of algorithms proposed makes a colour a desirable feature to exploit when appropriate. In real time to detect and track the vehicles, colour histogram based algorithm is used. To segment image into background and object and to describe the colour

distribution within the sequence of images, a model is created according to a Gaussian Mixture method. The occlusion buffer is used to handle the object occlusion.

To do the segmentation based on colour, one needs to transform the image from rgb model to hsv model. To understand what colour models are, one first needs to understand what colour spaces are.

A device colour space simply describes the range of colours, or gamut, that a camera can see, a printer can print, or a monitor can display. Editing colour spaces, on the other hand, such as Adobe RGB or sRGB, are device-independent. They also determine a colour range you can work in. Their design allows you to edit images in a controlled, consistent manner. A device colour space is tied to the idiosyncrasies of the device it describes. An editing space, on the other hand, is gray balanced — colours with equal amounts of Red, Green, and Blue appear neutral. Editing spaces also are perceptually uniform; i.e. changes to lightness, hue, or saturation are applied equally to all the colours in the image.

Now a question arises what a colour space contains.

Imagine a box containing all the visible colours. The farther from the center of the box you go, the more saturated the colours become — Red towards one corner, Blue towards another, Green towards the third (our box has a curious shape). A Cyan, Magenta, Yellow colour space works the same way, except that the primary colours are CMY rather than RGB. For simplicity, we will refer only to RGB spaces, but the comments apply equally to CMY(K) colour spaces. A colour space can be represented as a balloon blown up inside the box. The space taken up by the balloon is the portion of the total number of visible colours that fall within the particular colour space. Larger balloons contain more colours, or have a larger gamut, while smaller balloons hold fewer colours. The surface of the balloon has the most saturated colours that the colour space can hold. Any colours falling outside the balloon can't be reproduced in that colour space.

Colours inside the balloon are described using (R,G,B) coordinates. The most saturated (i.e. purest) red in any colour space has an R-value of 255. Since larger colour spaces have larger balloons, they contain both more air volume (i.e. more colours), and the surface of the balloon is farther from the center of the box (i.e. the colours are more saturated). Therefore, larger colour spaces such as Adobe RGB contain both more colours and more highly saturated colours than smaller spaces like sRGB. A comparison of the Adobe RGB and sRGB gamuts is below. As you can see, working with Adobe RGB allows you to see and print more of most colours. Adobe RGB was designed to contain the entire colour gamut available from most CMYK printers. sRGB is an HP/Microsoft defined colour space that describes the colours visible on a low end monitor.

Colour models are actually what we should be concerned with right now. An 8 – bit image actually has all its pixels with values between 0 and 255. That is actually a colour model that we are following, dictates us to do that. Wikipedia says that for a tuple of 3 integers (intensity value of pixel in 3 channels), if we associate with it a precise description of how this tuple is to be interpreted, then it becomes a color space. There are many color models that are used. RGB is most commonly known and is used to display images on a computer (I think that's what made this model popular). In it, a tuple of 3 integers denotes color of a particular pixel in the order of (R,G,B). The HSV model which has been used in the implementation stands for Hue, Saturation and Value. It closely aligns with the way human vision perceives colour-making attributes. In these models, colours of each hue are arranged in a radial slice, around a central axis of neutral colours which ranges from black at the bottom to white at the top. The HSV representation models the way paints of different colours mix together, with the saturation dimension resembling various shades of brightly coloured paint, and the value dimension resembling the mixture of those paints with varying amounts of black or white paint. The HSL model attempts to resemble more perceptual colour models such as NCS or Munsell, placing fully saturated colours around a circle at a lightness value of 1⁄2, where a lightness value of 0 or 1 is fully black or white, respectively.

HSV has a cylindrical geometry with hue, their angular dimension, starting at the red primary at 0°, passing through the green primary at 120° and the blue primary at 240°, and then wrapping back to red at 360°. In each geometry, the central vertical axis comprises the neutral, achromatic, or grey colours, ranging from black at lightness 0 or value 0, the bottom, to white at lightness 1 or value 1, the top.

The implementation uses the HSV values to differentiate between the object to be tracked and it background. Hue is actually the color portion of color model. The hue limits for various colors are:

Red	0-60
Yellow	60-120
Cyan	180-240
Blue	240-300
Magenta	300-360
Green	120-180

Saturation is the amount of gray in the color, expressed by %.
Value is the intensity of color, expressed in %.

In the implementation, the HSV transformation is applied only to a region of interest instead of the entire frame. The advantage of doing this is that we can reduce false positives to a great extent. But it is helpful only when the camera is centered at a fixed at a point instead of making it to follow a ball.

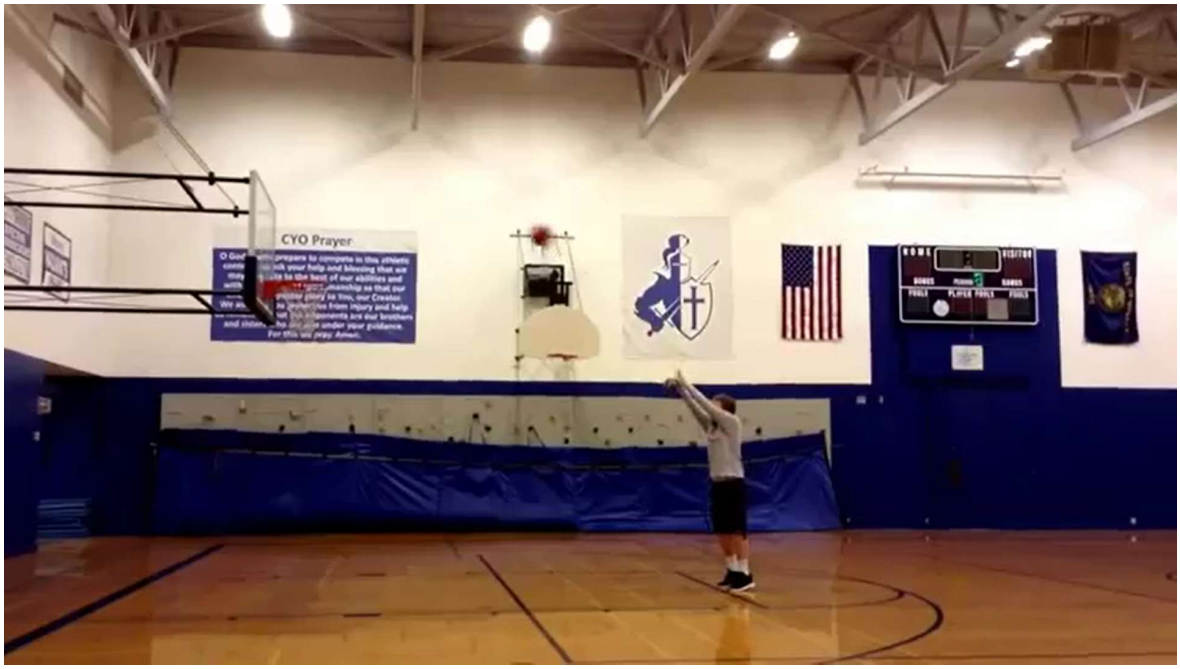
So for tracking by colours, you specify the range of hue (remember range of saturation and value is constant). If one has a background that is different from the colour one has specified, then tracking is easy. The closer the colours are, more painful will it be to track.

The implementation is shown below:

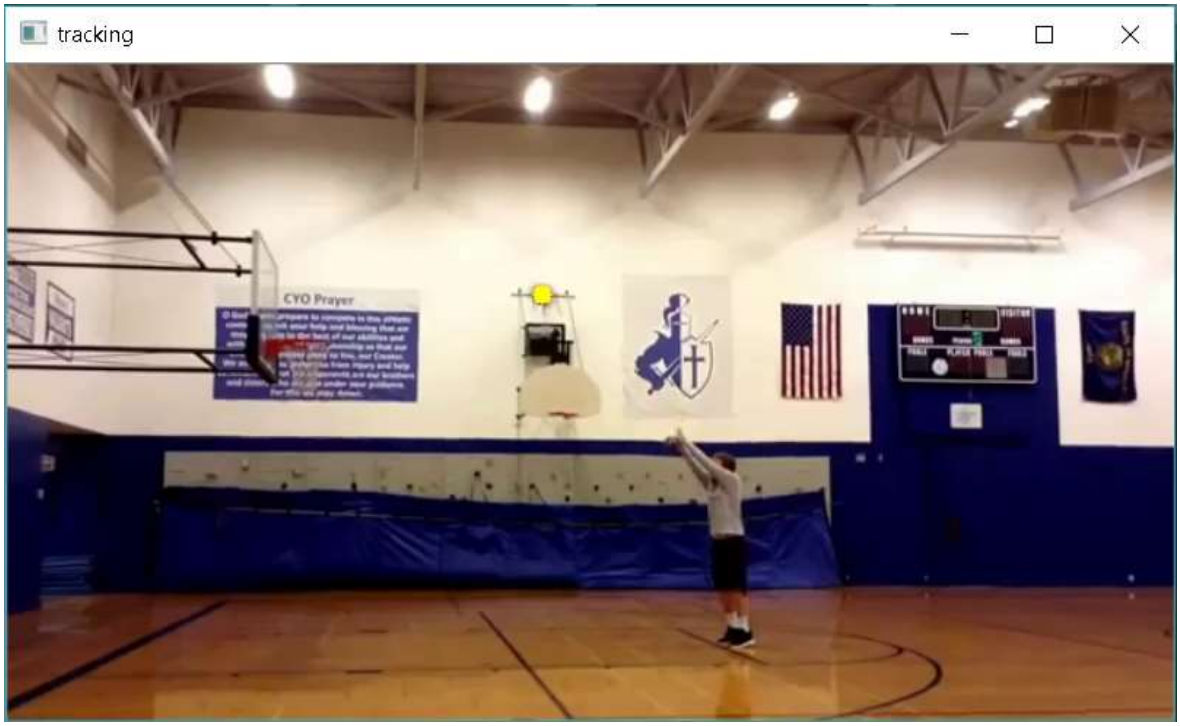
```
while(True):
      _, image = cap.read()
      if _ == False:
            break
      height, width, __ = image.shape
      print(image.shape)
      roi_vertices = [
      (width / 4, height / 4.7),
      (width / 3.8, height / 1.8),
      (width / 1.64, height / 1.52),
      (width / 1.8, height / 4)

      ]
      vertices = np.array([roi_vertices], np.int32)

      mask1 = np.zeros_like(image)
      mask_colour = (255,255,255)
      cv2.fillPoly(mask1, vertices, mask_colour)
      masked_image = cv2.bitwise_and(image, mask1)



      blur_masked_image = cv2.GaussianBlur(masked_image, (3, 3),
2)
      hsv = cv2.cvtColor(blur_masked_image, cv2.COLOR_BGR2HSV)
```

```
      lower_limit = np.array([loHue,loSaturation,loValue])
      upper_limit                                                    =
np.array([high_hue,high_saturation,high_value])
      mask2 = cv2.inRange(hsv, lower_limit, upper_limit)
      res  =  cv2.bitwise_and(image,  blur_masked_image,  mask  =
mask2)
      erode_element  =  cv2.getStructuringElement(cv2.MORPH_RECT,
(3, 3))
      dilate_element  =  cv2.getStructuringElement(cv2.MORPH_RECT,
(3, 3))
      erosion = cv2.erode(mask2, erode_element, iterations = 1)

      erosion = cv2.medianBlur(erosion,3)
      dilation = cv2.dilate(erosion, dilate_element, iterations =
2)
      copy_dilation = dilation.copy()

      _,  contours,  hierarchy  =  cv2.findContours(copy_dilation,
cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
      center = None

      if len(contours) > 0:
            c = max(contours, key = cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center  =  (int(M["m10"]  /  M["m00"]),  int(M["m01"]  /
M["m00"]))
```

The results are also exciting. The frame shown below is a randomly captured original frame:



And the same processed frame is:



So, the colour tracking is very accurate as long as the object to be tracked is different from the background. In the implementation functions like medianBlur, erosion, dilation, etc. are also used. Those will be discussed after this section.

2. Background subtraction:

Background subtraction is a major preprocessing steps in many vision based applications. For example, consider the cases like visitor counter where a static camera takes the number of visitors entering or leaving the room, or a traffic camera extracting information about the vehicles etc. In all these cases, first you need to extract the person or vehicles alone. Technically, you need to extract the moving foreground from static background.

If you have an image of background alone, like image of the room without visitors, image of the road without vehicles etc, it is an easy job. Just subtract the new image from the background. You get the foreground objects alone. But in most of the cases, you may not have such an image, so we need to extract the background from whatever images we have. It become more complicated when there is shadow of the vehicles. Since shadow is also moving, simple subtraction will mark that also as foreground. It complicates things.

Several algorithms were introduced for this purpose. OpenCV has implemented three such algorithms which is very easy to use. We will see them one-by-one.

BackgroundSubtractorMOG

It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It was introduced in the paper "An improved adaptive background mixture model for real-time tracking with shadow detection" by P. KadewTraKuPong and R. Bowden in 2001. It uses a method to model each background pixel by a mixture of K Gaussian distributions (K = 3 to 5). The weights of the mixture represent the time proportions that those colours stay in the scene. The probable background colours are the ones which stay longer and more static.

While coding, we need to create a background object using the function, cv2.createBackgroundSubtractorMOG(). It has some optional parameters like length of history, number of gaussian mixtures, threshold etc. It is all set to some default values. Then inside the video loop, use backgroundsubtractor.apply() method to get the foreground mask.

BackgroundSubtractorMOG2

It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It is based on two papers by Z.Zivkovic, "Improved adaptive Gausian mixture model for background subtraction" in 2004 and "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction" in 2006. One important feature of this algorithm is that it selects the appropriate number of gaussian distribution for each pixel. (Remember, in last case, we took a K gaussian distributions throughout the algorithm). It provides better adaptibility to varying scenes due illumination changes etc.

As in previous case, we have to create a background subtractor object. Here, you have an option of selecting whether shadow to be detected or not. If detectShadows = True (which is so by default), it detects and marks shadows, but decreases the speed. Shadows will be marked in gray color.

BackgroundSubtractorGMG

This algorithm combines statistical background image estimation and per-pixel Bayesian segmentation. It was introduced by Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg in their paper "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation" in 2012. As per the paper, the system ran a successful interactive audio art installation called “Are We There Yet?” from March 31 - July 31 2011 at the Contemporary Jewish Museum in San Francisco, California.

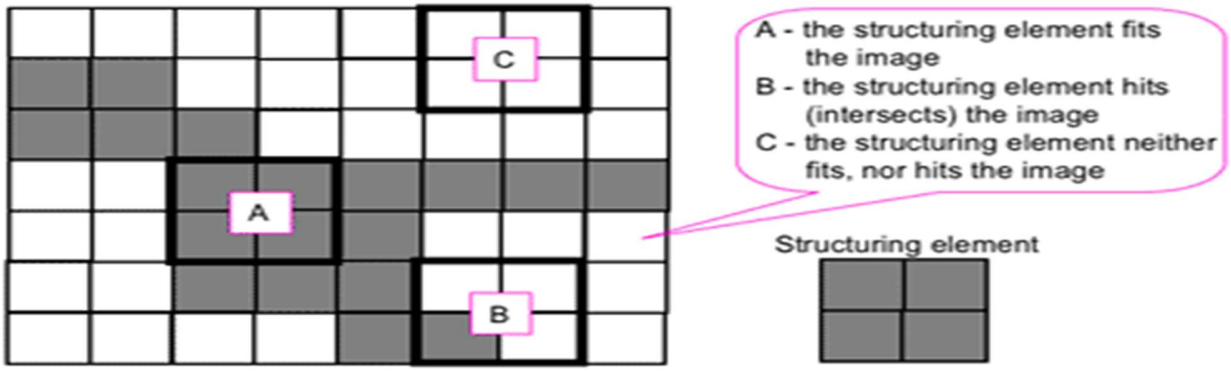
It uses first few (120 by default) frames for background modelling. It employs probabilistic foreground segmentation algorithm that identifies possible foreground objects using Bayesian inference. The estimates are adaptive; newer observations are more heavily weighted than old observations to accommodate variable illumination. Several morphological filtering operations like closing and opening are done to remove unwanted noise. You will get a black window during first few frames. It would be better to apply morphological opening to the result to remove the noises.

Morphological Operations:

Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image. According to Wikipedia, morphological operations rely only on the

relative ordering of pixel values, not on their numerical values, and therefore are especially suited to the processing of binary images. Morphological operations can also be applied to greyscale images such that their light transfer functions are unknown and therefore their absolute pixel values are of no or minor interest.

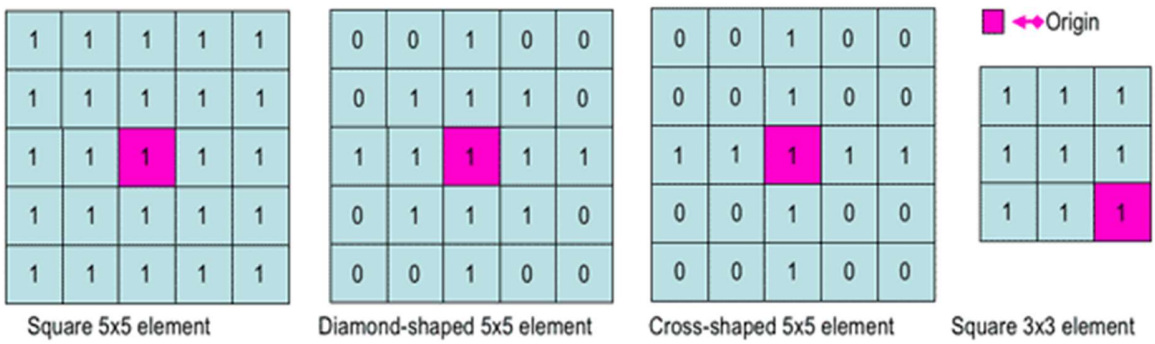
Morphological techniques probe an image with a small shape or template called a structuring element. The structuring element is positioned at all possible locations in the image and it is compared with the corresponding neighbourhood of pixels. Some operations test whether the element "fits" within the neighbourhood, while others test whether it "hits" or intersects the neighbourhood:



A morphological operation on a binary image creates a new binary image in which the pixel has a non-zero value only if the test is successful at that location in the input image.

The structuring element is a small binary image, i.e. a small matrix of pixels, each with a value of zero or one:

- The matrix dimensions specify the size of the structuring element.
- The pattern of ones and zeros specifies the shape of the structuring element.
- An origin of the structuring element is usually one of its pixels, although generally the origin can be outside the structuring element.



Fundamental operations

More formal descriptions and examples of how basic morphological operations work are given in the Hypermedia Image Processing Reference (HIPR) developed by Dr. R. Fisher et al. at the Department of Artificial Intelligence in the University of Edinburgh, Scotland, UK.

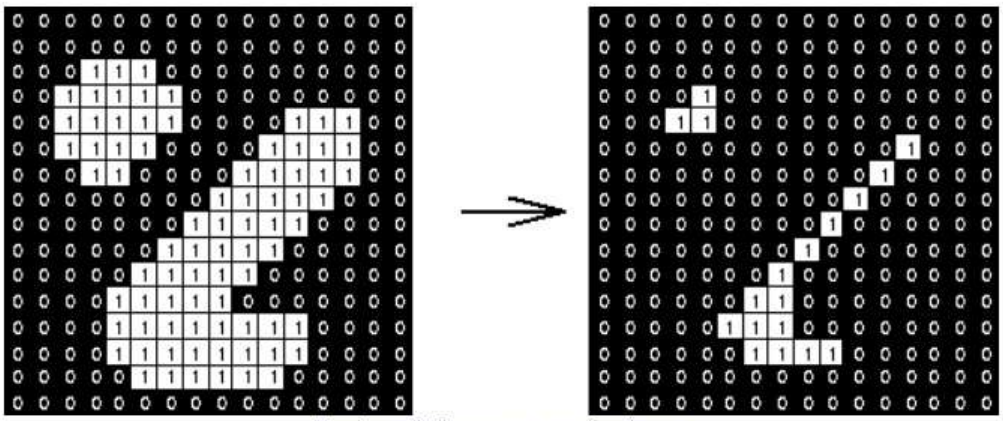
Erosion and dilation

The erosion of a binary image f by a structuring element s produces a new binary image g = f s with ones in all locations (x,y) of a structuring element's origin at which that structuring element s fits the input image f, i.e. g(x,y) = 1 is s fits f and 0 otherwise, repeating for all pixel coordinates (x,y).

Greyscale image Binary image by thresholding Erosion: a 2×2 square structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

Erosion with small (e.g. 2×2 - 5×5) square structuring elements shrinks an image by stripping away a layer of pixels from both the inner and outer boundaries of regions. The holes and gaps between different regions become larger, and small details are eliminated:



Erosion: a 3×3 square structuring element

(www.cs.princeton.edu/~pshilane/class/mosaic/).

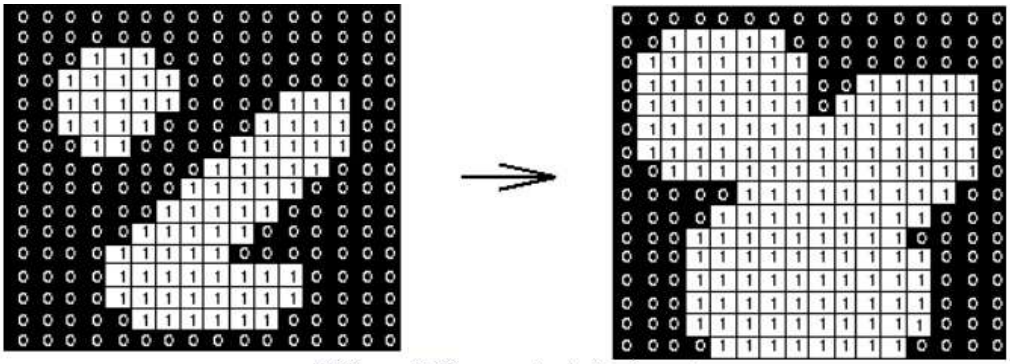
The dilation of an image f by a structuring element s produces a new binary image g with ones in all locations (x,y) of a structuring element's orogin at which that structuring element s hits the the input

image f, i.e. g(x,y) = 1 if s hits f and 0 otherwise, repeating for all pixel coordinates (x,y). Dilation has the opposite effect to erosion -- it adds a layer of pixels to both the inner and outer boundaries of regions.

Binary image Dilation: a 2×2 square structuring element

http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html

The holes enclosed by a single region and gaps between different regions become smaller, and small intrusions into boundaries of a region are filled in:



Dilation: a 3×3 square structuring element
(www.cs.princeton.edu/~pshilane/class/mosaic/).

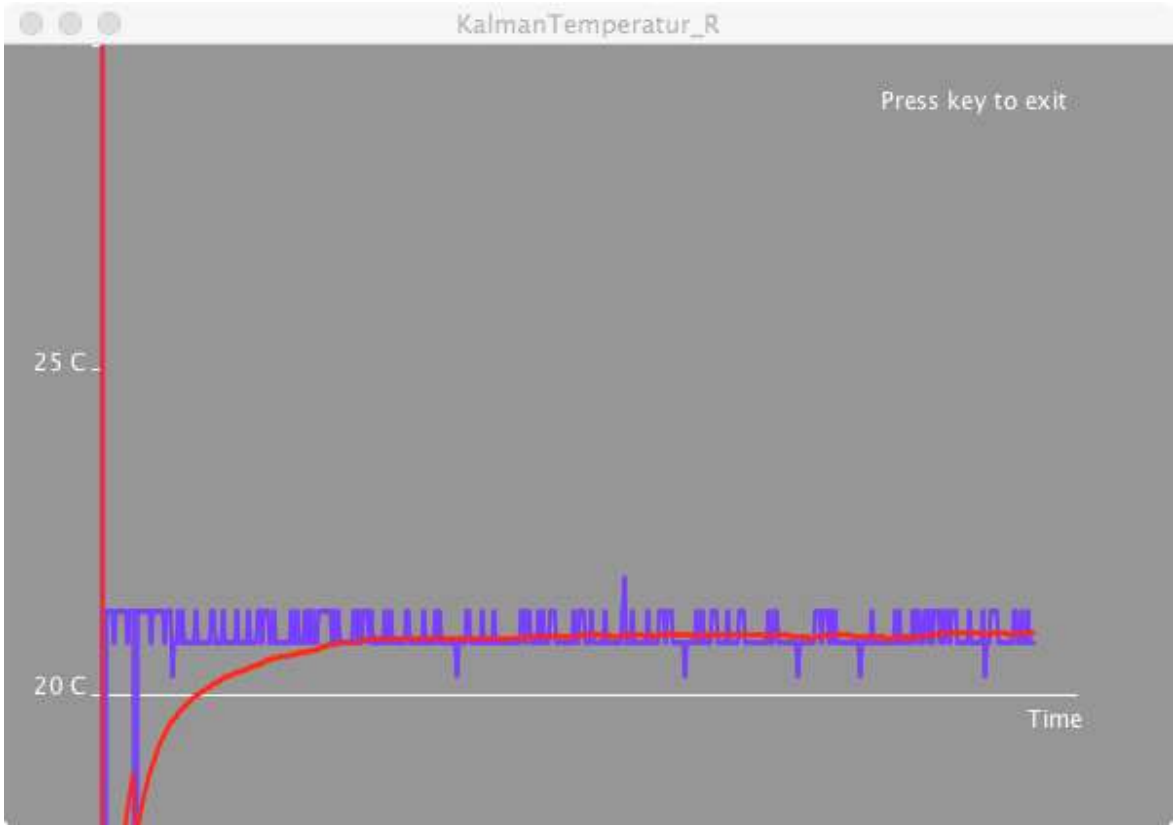
Prediction

The prediction of the trajectory of projectile motion can again be done with many methods. These methods include Kalman filter, euler's numerical integration, RK4 method for numerical integration and equations for projectile motion.

Kalman filter:

Before actually using Kalman filter, we must first see what Kalman filters actually are and how do they work.

It is an iterative mathematical process that uses a set of equations and consecutive data inputs to quickly estimate the true value, position, velocity, etc. of the object being measured, when the measured values contain unpredicted or random error, uncertainty or variation.



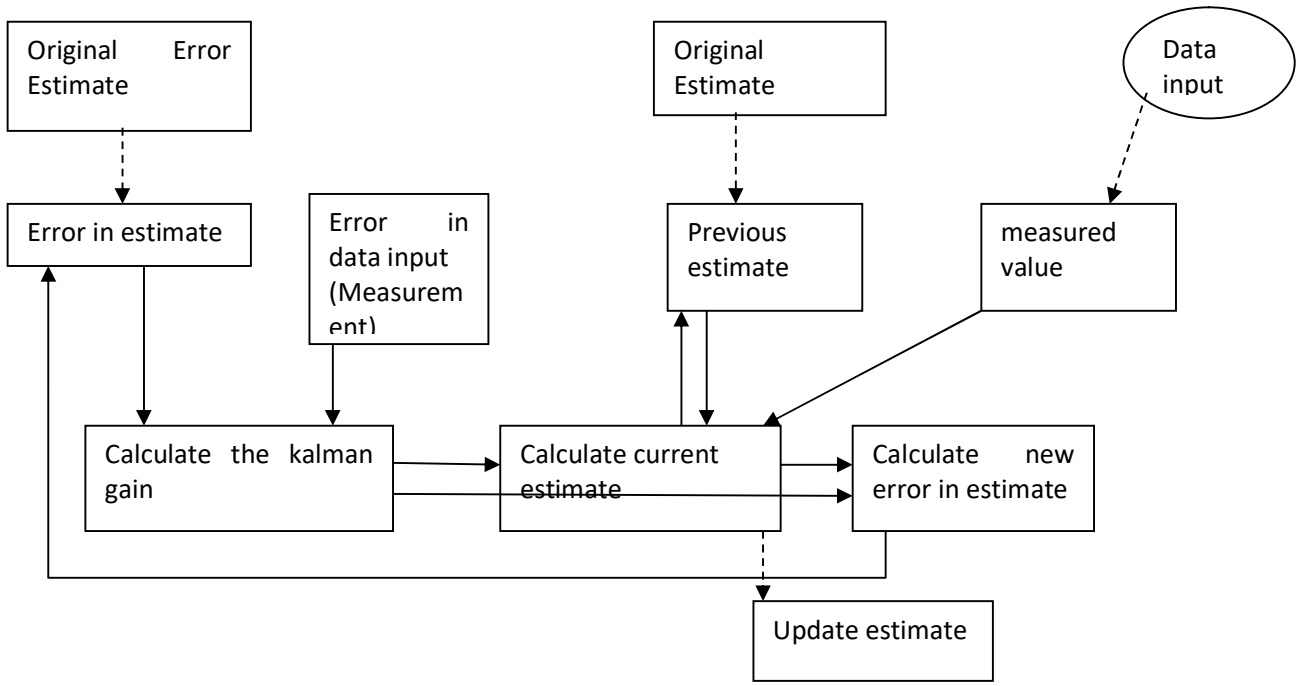
The operation of kalman filter can be understood with the above example. The blue lines are the measured values while the red line is the output of a kalman filter. You can see that the red line is actually the average of the blue lines.

So the kalman filter is all about taking averages? Why do so many people on the internet go on ranting about how tough they are to understand?

Actually there are subtle differences in taking average of data set and working of kalman filter. Computing average is too painful computationally because you need to remember all your past values. Also, real time averaging is not possible (I think) so it would take a lot of time for the output of averaging filter to converge to the true value. The advantage of using kalman filter is that it does this averaging with just single measurement along with the information of the previous state.

So the strength of kalman filters is that they are easier to design computationally.

Now lets see how they work.



The above block diagram shows how kalman filter works.

1. The three calculations of kalman gain, current estimate and error in estimate are iterative and happen over and over again.
2. The gain is calculated by error in estimate i.e the error in previous estimate.
3. When we first start we take original error estimate but for further calculations, we do not need it.
4. The gain puts relative importance to the two blocks.
5. If error in previous estimate is less then we give more importance to it. If error in data input is less, then it is given more relative importance (how this is actually done will be shown later).
6. The kalman gain feeds the current estimate block. It depends on previous estimate and measured value. The gain decides by how much previous estimate and measured value are to be weighted.
7. New error in estimate is calculated by kalman gain and current estimate.

8. In every estimate, a value comes out which brings us to the actual value.

For the expressions given below:

KG = kalman gain

Error in estimate = E_{EST}

Error in measurement = E_{MEA}

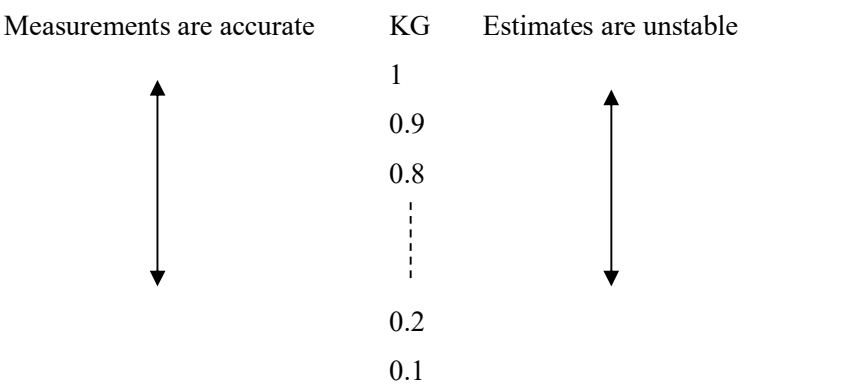
Current estimate = EST_t

Previous estimate = EST_{t-1}

$$KG = \frac{E_{est}}{E_{est} + E_{mea}}$$

$$0 \leq KG \leq 1$$

$$EST_t = EST_{(t-1)} + KG[MEA - EST_{(t-1)}]$$



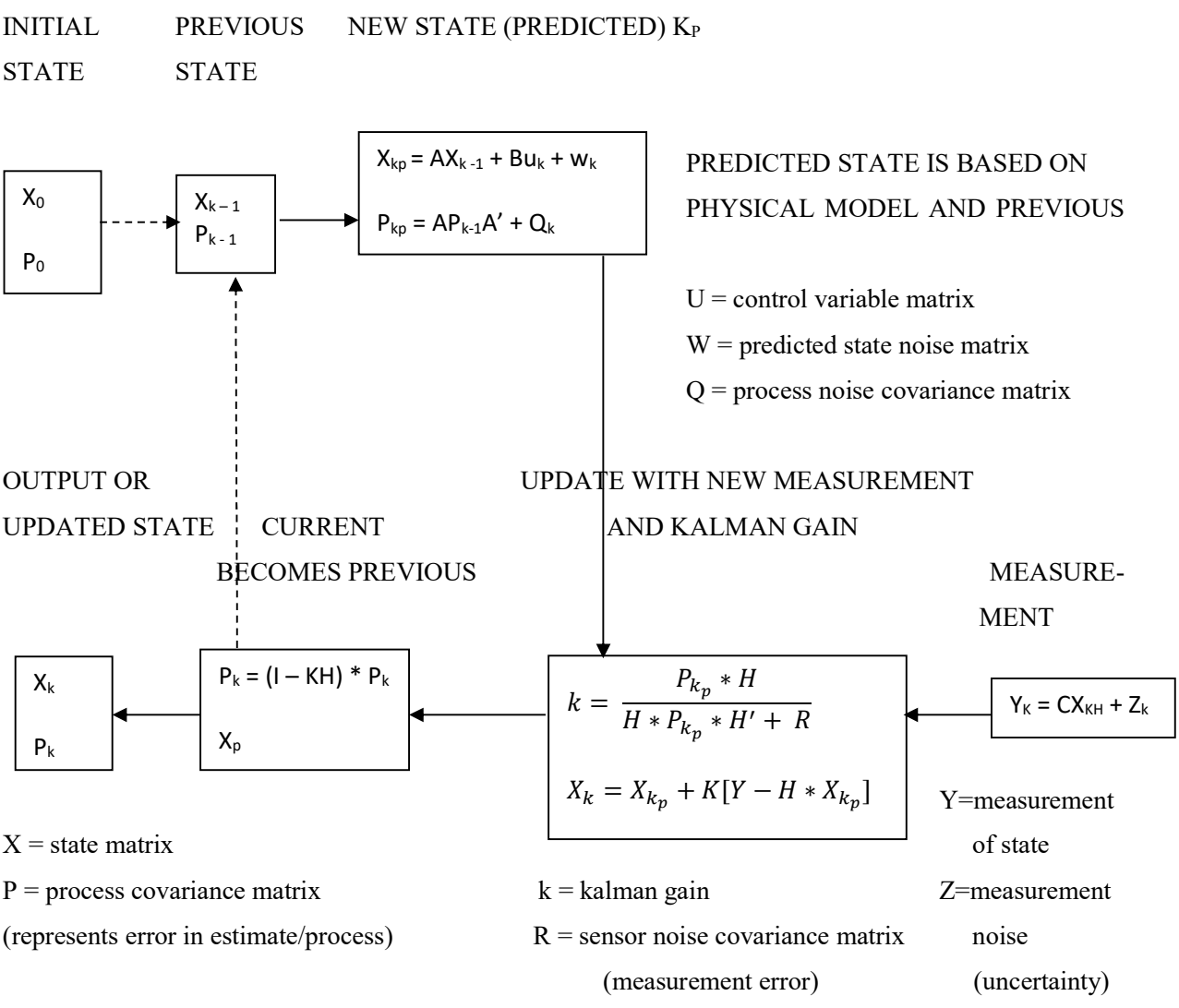
If KG is small then we give more weight to previous estimates. If KG is large then more weight is given to measured value. Overtime, KG becomes very low. This is desirable as this means that we are very close to the actual value.

$$E_{est(t)} = \frac{E_{mea} * E_{est(t-1)}}{E_{mea} + E_{est(t-1)}}$$

$$E_{est(t)} = [1 - KG]E_{est(t-1)}$$

Multidimensional Model:

Everything discussed up to this point forms the backbone of this entire discussion. So one should give it a read once more over so that it is understood well.
In this part we will be dealing with the equations that give kalman filter its power.



We will now discuss some of the calculations above.

State Matrix:

The state matrix is defined as:

$$X_k = A * X_{k-1} + Bu_k + w_k$$

Where X can be

$\begin{bmatrix} X \\ X' \end{bmatrix}$ or $\begin{bmatrix} Y \\ Y' \end{bmatrix}$ or $\begin{bmatrix} X \\ Y \\ X' \\ Y' \end{bmatrix}$ or $\begin{bmatrix} X \\ X' \\ Y \\ Y' \end{bmatrix}$

Take 3 examples of a state matrix-

Rising fluid	**Falling object**	**Moving object**
$X = \begin{bmatrix} Y \\ Y' \end{bmatrix}$	$X = \begin{bmatrix} Y \\ Y' \end{bmatrix}$	$X = \begin{bmatrix} Y \\ Y' \end{bmatrix}$
$A = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}$	$A = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}$	$A = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix}$
$AX = \begin{bmatrix} Y + \Delta TY' \\ 0 + Y' \end{bmatrix}$		

At first, it seems that AX is incorrect since position also depends upon acceleration and velocity also varies. But it is taken care by BU_K term.

$B = \begin{bmatrix} 0.5 * \Delta T^2 \\ \Delta T \end{bmatrix}$	$B = \begin{bmatrix} 0.5 * \Delta T^2 \\ \Delta T \end{bmatrix}$	$B = \begin{bmatrix} 0.5 * \Delta T^2 \\ \Delta T \end{bmatrix}$
U=0	U=g	u=a
$BU = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$BU = \begin{bmatrix} g * 0.5 * \Delta T^2 \\ g * \Delta T \end{bmatrix}$	$BU = \begin{bmatrix} a * 0.5 * \Delta T^2 \\ a * \Delta T \end{bmatrix}$

$$X_K = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} y_{K-1} \\ y_K \end{bmatrix} + \begin{bmatrix} 0.5 * \Delta T^2 \\ \Delta T \end{bmatrix} * g$$

$$X_K = \begin{bmatrix} y_{K-1} + \Delta T * y'_{K-1} + 0.5 * \Delta T^2 * g \\ y'_{K-1} + \Delta T * g \end{bmatrix}$$

The above matrix represents the next state predicted by the previous state.

State matrix in 2D:

The state matrix X in 2D is-

$$X = \begin{bmatrix} X \\ Y \\ X' \\ Y' \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$AX = \begin{bmatrix} X + \Delta T * X' \\ Y + \Delta T * Y' \\ X' \\ Y' \end{bmatrix}$$

$$\text{OR, } X = \begin{bmatrix} X \\ X' \\ Y \\ Y' \end{bmatrix} \qquad A = \begin{bmatrix} 1 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$AX = \begin{bmatrix} X + \Delta T * X' \\ X' \\ Y + \Delta T * Y' \\ Y' \end{bmatrix}$$

The BU_K matrix is:

$$\begin{bmatrix} 0.5 * \Delta T^2 & 0 \\ 0 & 0.5 * \Delta T^2 \\ \Delta T & 0 \\ 0 & \Delta T \end{bmatrix} * \begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} 0.5 * \Delta T^2 * a_x \\ 0.5 * \Delta T^2 * a_y \\ \Delta T * a_x \\ \Delta T * a_y \end{bmatrix}$$

Finally, the predicted state for a 2D state matrix is:

$$X_K = \begin{bmatrix} X_{K-1} + X' * \Delta T + a_x * 0.5 * \Delta T^2 \\ Y_{K-1} + Y' * \Delta T + a_y * 0.5 * \Delta T^2 \\ X'_{K-1} + a_x * \Delta T \\ Y'_{K-1} + a_y * \Delta T \end{bmatrix} + W_K$$

Please note that the noise in the process measurement is neglected until now. This makes discussion easier.

Also it must be noted that these above matrices for three dimensions are same as the two dimension case. Please check the references below.

Now a question arises that why one should use Kalman filters if the same job can be done by the kinematics equation? The reason is pretty onvious.

The Kalman filter puts correct weight to observations and the predicted state. The Kalman filter takes noise into account while kinematics equations do not. It puts more weight on that set which can be relied upon.

Covariance matrix:

$P_K = A*P_{K-1}*T + Q$

$$K_k = \frac{P_K * H_T}{H * P_K * H^T + R}$$

P = state covariance matrix (error in estimate)

Q = process noise covariance matrix (it keeps state covariance matrix from becoming too small or going to zero. It represents the uncertainty or error in predicted state or estimates noise in process)

R = measurement covariance matrix (error in measurement)

K = Kalman gain

1. If R tends to zero then k tends to 1 which means adjust primarily with adjustment update.
2. If R is large the k tends to zero which means adjust primarily with predicted state.
3. If P tends to zero then measurement updates are usually ignored.

The covariance matrix in 2D is given as:

$$\begin{bmatrix} \frac{\sum(\overline{x} - x_i)}{N} & \frac{\sum(\overline{x} - x_i)(\overline{y} - y_i)}{N} \\ \frac{\sum(\overline{y} - y_i)(\overline{x} - x_i)}{N} & \frac{\sum(\overline{y} - y_i)}{N} \end{bmatrix}$$

The format of 3D covariance matrix is same as 2D one. It is represented as:

$$\begin{bmatrix} variance\ of\ x & cov(x,y) & cov(x,z) \\ cov(y,x) & variance\ of\ y & cov(y,z) \\ cov(z,x) & cov(z,y) & variance\ of\ z \end{bmatrix}$$

The code used for predicting the flight of the ball directly implements what we have seen above. The function Kalman starts off with the calculation of the new predicted state. Then we calculate the Kalman gain k which is used to assign weights to the measurement and prediction values.

To use Kalman filter for prediction, we first pass the tracked center of our ball to the Kalman function to get the next updated state. After that we need to operate Kalman filter without measurement values. For that purpose, we again pass the predicted state to Kalman function which predicts a new state for this predicted state. In this way, we loop it again and again for a single frame.

When we get the next frame, the whole process is again repeated. By the virtue of Kalman filter, the predicted paths become more and more accurate as we keep on getting more frames. However, it is possible that the Kalman filter may break under some conditions which are out of its control. It will be discussed shortly.

So Kalman filter is indeed a powerful tool. No wonder why rocket scientists, astronomers and aeronautical engineers use it so extensively. If Kalman filter is designed properly, then one can track anything even if no initial seed value is provided. This is one of the major factor which makes Kalman filters really useful.

In this particular application, Kalman filter can break down if the ball centers are not tracked properly. Tracking ball centers can be both easy and difficult depending on the recording environment and the method used for tracking. So for successful application utmost attention should be given to tracking algorithms.

1. If it is known that the background and the object to be tracked form a color contrast with each other, then image can be transferred to hsv space. There the color of the object can be used to tack the object.

2. If the above technique fails, one can use background subtraction. Opencv provides just 3 methods. Check out the bgs library at github. This library has more than 30 background subtraction algorithms. You can also see there that how these algorithms perform in practical situations.

3. Since videos are composed of large number of images or frames, the haar cascade classifier that is popularly used for detecting features in an image, can be used to track a ball in a video. However, one needs to train the classifier by supplying thousands of positive and negative images to it. Also, haar cascade classifiers may become slow for a video. So it is recommended that one should not use it.

The code used that implements Kalman filter in python is:

```
def kalman(mu,P,F,Q,B,u,z,H,R):
    # mu, P : current state and its uncertainty
    # F, Q  : Dynamic system and its noise
    # B, u  : control model and the entrance
    # z     : observation
    # H, R  : Observation model and its noise

    mup = F @ mu + B @ u
    pp  = F @ P @ F.T + Q

    zp = H @ mup

    # if there is no observation we only do prediction

    if z is None:
        return mup, pp, zp

    epsilon = z - zp

    k = pp @ H.T @ la.inv(H @ pp @ H.T +R)

    new_mu = mup + k @ epsilon
    #print(new_mu)
    new_P  = (np.eye(len(P))-k @ H) @ pp
    return new_mu, new_P, zp
```

Please note that the function uses numPy library in python.So before using it, run a pip install command to install the library. You can use other methods to install it.

Euler's method:

Euler's method is a numerical procedure for solving differential equations if an initial condition is provided. It is the most basic and easiest one to implement out of the other numerical methods. This method is named after Leonhard Euler.

The euler method is the first-order method which means that the local error is proportional to the square of the step size, and the global error is proportional to the step size. The euler's method often serves as the basis to construct more complex methods.

The euler's method can be derived by various methods and the most basic one is discussed here.

Consider the Taylor expansion of the function y around t_0.:

$$y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2y''(t_0) + O(h^3)$$

The differential equation states that $y' = f(t, y)$. If this is substituted in the taylor expansion and the quadratic and higher order terms are ignored, the Euler method arises.

Therefore, $Y(t_0 + h) = y(t_0 + h) + hy'(t_0)$

A closely related derivation is to substitute the forward finite difference formula for the derivative,

$$y'(t_0) = \frac{y(t_0 + h) - y(t_0)}{h}$$

Integrating the above equation,

$$y(t_0 + h) - y(t_0) = \int_{t_0}^{t_0+h} f\big(t, y(t)\big)dt$$

Or,

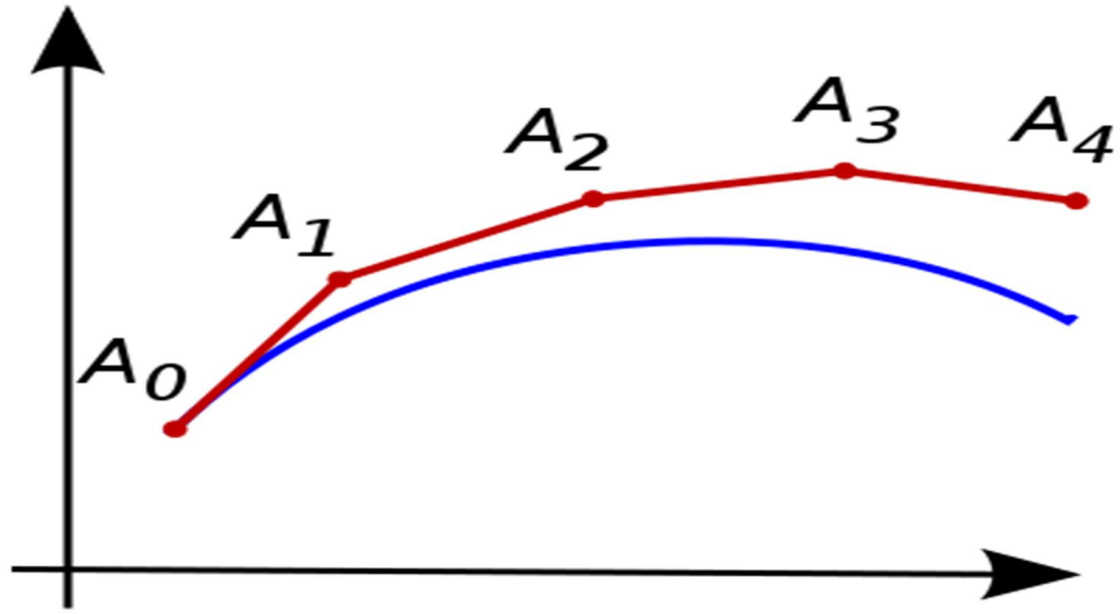
$$\int_{t_0}^{t_0+h} f\big(t, y(t)\big)dt = hf(t_0, y(t_0))$$

Therefore,

$$y(t_0 + h) - y(t_0) = hf(t_0, y(t_0))$$

The above equation represents the euler's equation for a step size of h.

The Euler's equation is simple to understand and easy to implement. However, most of the practical applications to do not ever use euler's method; they use fourth method Runge-kutta method. The primary reason is that the local truncation error of the Euler method made in a single step is equal to the square of the step size. As a result, Euler's method breaks apart when the step size is high, or even moderate.



The above picture is the illustration of the Euler's method. The unknown curve is blue, and its polygonal approximation is in red.

Intuitively, the working of euler's method can be understood in the following manner. Consider the problem of calculating the shape of an unknown curve which starts at a given point and satisfies a given differential equation. Here, a differential equation can be thought of as a formula by which the slope of the tangent line to the curve can be computed at any point on the curve, once the position of that point has been calculated.

The idea is that while the curve is initially unknown, its starting point, which we denote by A_0, is known (see the picture). Then, from the differential equation, the slope to the curve at A_0can be computed, and so, the tangent line.

Take a small step along that tangent line up to a point A_1. Along this small step, the slope does not change too much, so A_1 will be close to the curve. If we pretend A_1 is still on the curve, the same reasoning as for the point A_0 above can be used. After several steps, a polygonal curve $A_0,A_1,A_2\ldots$ is computed. In

general, this curve does not diverge too far from the original unknown curve, and the error between the two curves can be made small if the step size is small enough and the interval of computation is finite.

RESULT

The complete code for the basic state estimator to track a basketball in video data is given below:

```
import cv2
import numpy as np
import numpy.linalg as la
#from umucv.kalman import kalman

def kalman(mu,P,F,Q,B,u,z,H,R):
    # mu, P : current state and its uncertainty
    # F, Q  : Dynamic system and its noise
    # B, u  : control model and the entrance
    # z     : observation
    # H, R  : Observation model and its noise

    mup = F @ mu + B @ u
    pp  = F @ P @ F.T + Q

    zp = H @ mup

    # if there is no observation we only do prediction

    if z is None:
        return mup, pp, zp

    epsilon = z - zp

    k = pp @ H.T @ la.inv(H @ pp @ H.T +R)

    new_mu = mup + k @ epsilon
    #print(new_mu)
    new_P  = (np.eye(len(P))-k @ H) @ pp
    return new_mu, new_P, zp
```

```
cap = cv2.VideoCapture('2 (2).mp4')
loHue = 0
loSaturation = 50
loValue = 50
high_hue = 0
high_saturation = 255
high_value = 255

def low_hue(x):
      global loHue
      loHue = x


def upper_hue (x):
      global high_hue
      high_hue = x



cv2.namedWindow('Trackbars', flags=cv2.WINDOW_OPENGL)
cv2.resizeWindow('Trackbars', 500, 30)
cv2.moveWindow('Trackbars', 500, 600)
cv2.createTrackbar('loHue', 'Trackbars', 0, 180, low_hue)

cv2.createTrackbar('upperHue', 'Trackbars', 0, 180, upper_hue)

cv2.setTrackbarPos('loHue', 'Trackbars', 3)
cv2.setTrackbarPos('upperHue', 'Trackbars', 9)



fps = 30
dt = 1 / fps
#t = np.arange(0, 2.01, dt)
noise = 3
```

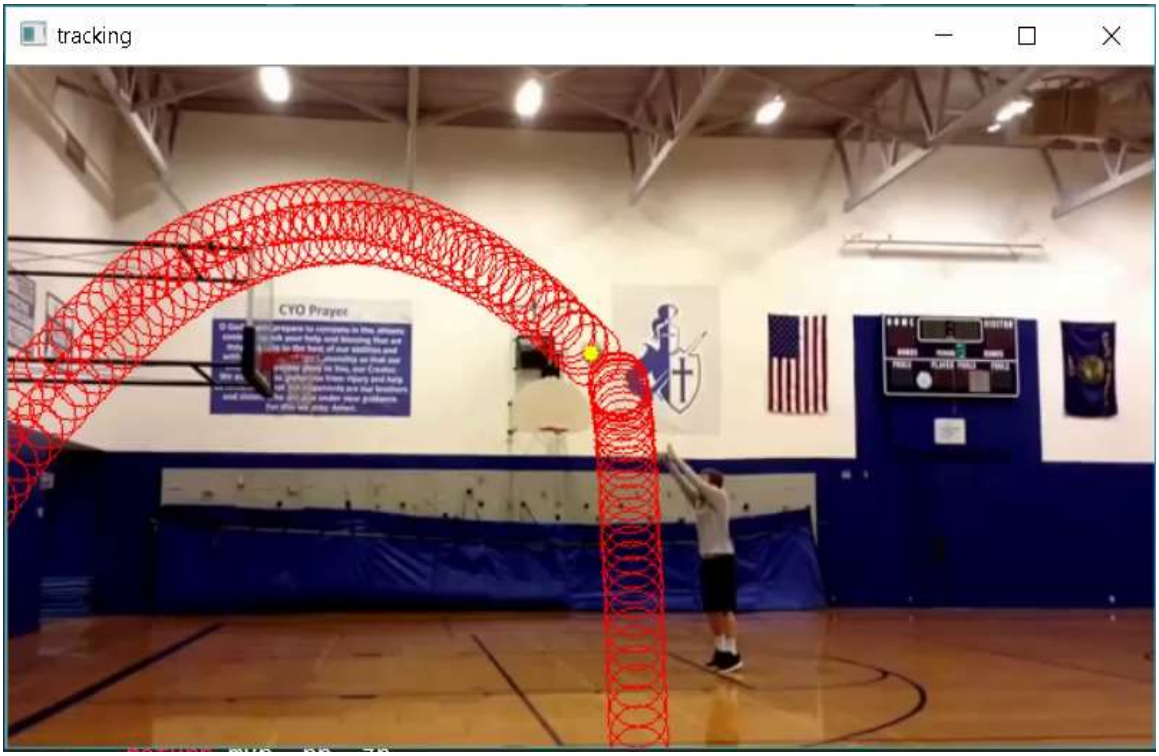
```
a = np.array([0, 300])
F = np.array(
      [1, 0, dt, 0,
      0, 1, 0, dt,
      0, 0, 1, 0,
      0, 0, 0, 1 ]).reshape(4,4)


B = np.array(
      [dt**2/2, 0,
      0, dt**2/2,
      dt, 0,
      0, dt ]).reshape(4,2)


H = np.array(
      [1,0,0,0,
      0,1,0,0]).reshape(2,4)


mu = np.array([0,0,0,0])
P = np.diag([1000, 1000, 1000, 1000])**2


sigmaM = 0.0001
sigmaZ = 3*noise

Q = sigmaM**2 * np.eye(4)
R = sigmaZ**2 * np.eye(2)

listCenterX=[]
listCenterY=[]
xe = []
xu = []
```

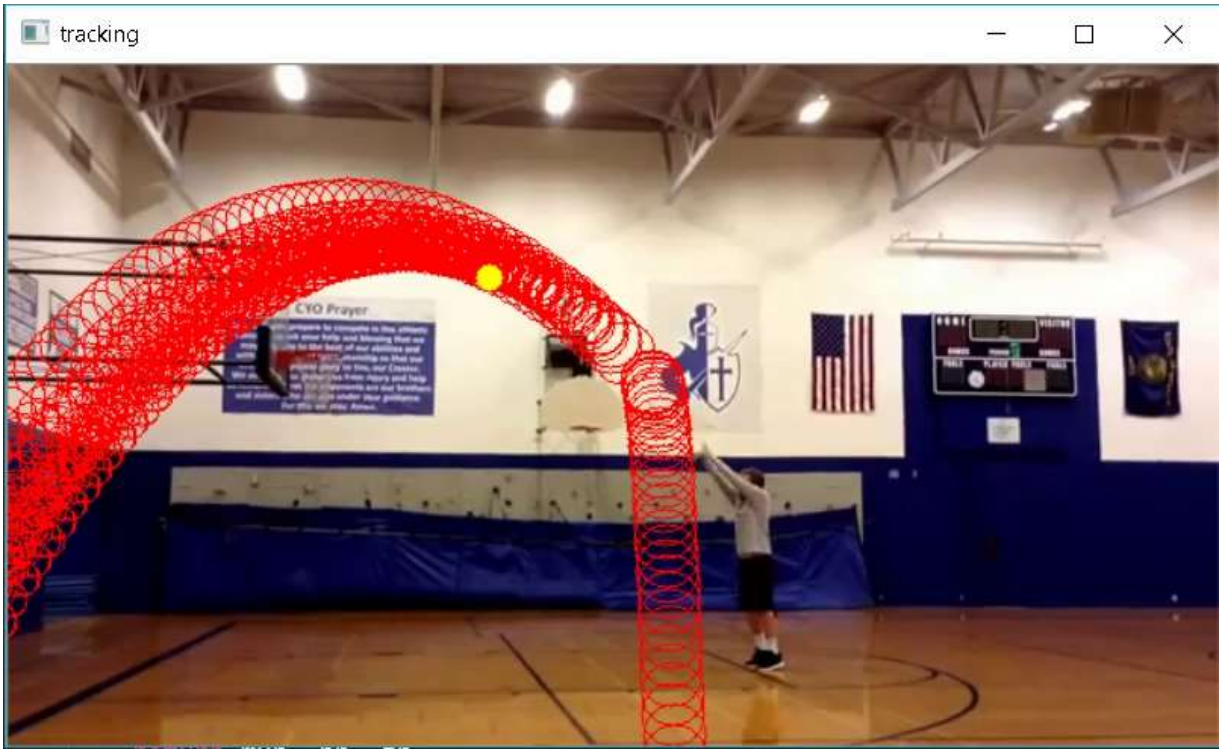
```
ye = []
yu = []
xp = []
yp = []
xpu = []
ypu = []


while(True):
      _, image = cap.read()
      if _ == False:
            break
      height, width, __ = image.shape
      print(image.shape)
      roi_vertices = [
      (width / 4, height / 4.7),
      (width / 3.8, height / 1.8),
      (width / 1.64, height / 1.52),
      (width / 1.8, height / 4)

      ]
      vertices = np.array([roi_vertices], np.int32)

      mask1 = np.zeros_like(image)
      mask_colour = (255,255,255)
      cv2.fillPoly(mask1, vertices, mask_colour)
      masked_image = cv2.bitwise_and(image, mask1)



      blur_masked_image = cv2.GaussianBlur(masked_image, (3, 3), 2)
      hsv = cv2.cvtColor(blur_masked_image, cv2.COLOR_BGR2HSV)
      lower_limit = np.array([loHue,loSaturation,loValue])
      upper_limit = np.array([high_hue,high_saturation,high_value])
      mask2 = cv2.inRange(hsv, lower_limit, upper_limit)
```

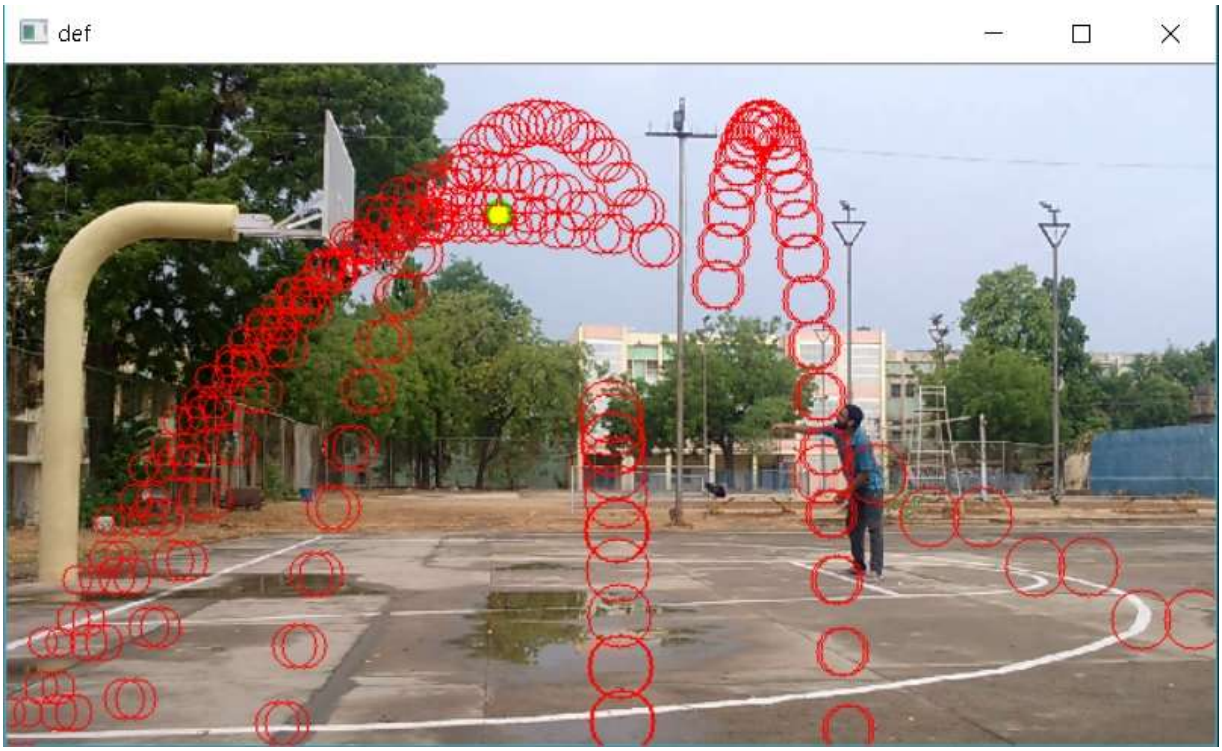
```
      res = cv2.bitwise_and(image, blur_masked_image, mask = mask2)
      erode_element = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
      dilate_element  =  cv2.getStructuringElement(cv2.MORPH_RECT,  (3,
3))
      erosion = cv2.erode(mask2, erode_element, iterations = 1)

      erosion = cv2.medianBlur(erosion,3)
      dilation = cv2.dilate(erosion, dilate_element, iterations = 2)
      copy_dilation = dilation.copy()

      _,   contours,   hierarchy   =   cv2.findContours(copy_dilation,
cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
      center = None

      if len(contours) > 0:
            c = max(contours, key = cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center   =   (int(M["m10"]   /   M["m00"]),   int(M["m01"]   /
M["m00"]))
            mu,P,pred = kalman(mu,P,F,Q,B,a,np.array([x,y]),H,R)
            xe.append(mu[0])
            ye.append(mu[1])
            xu.append(2 * np.sqrt(P[0, 0]))
            yu.append(2 * np.sqrt(P[1, 1]))
            P2 = P
            mu2 = mu
            res2 = []
            for __ in range (fps * 2):
                  mu2, P2, pred2 = kalman(mu2, P2, F, Q, B, a, None, H,
R)
                  xp.append(mu2[0])
                  yp.append(mu2[1])
                  xpu.append(2 * np.sqrt(P[0, 0]))
                  ypu.append(2 * np.sqrt(P[1, 1]))
```

```
            for n in range(len(xp)):
                  uncertainity_in_state = (xpu[n] + ypu[n]) / 2
                  cv2.circle(image,       (int(xp[n]),       int(yp[n])),
int(uncertainity_in_state), (0,0,255))
            cv2.circle(image,  (int(x),  int(y)),  int(radius),  (0,  255,
255), -2)
            cv2.imshow('tracking', image)


      k = cv2.waitKey(500) & 0xFF
      if k ==27:
            break
cap.release()
cv2.destroyAllWindows()
```

The resultant predicted paths are:




tracking
CYO Prayer


def

CONCLUSION

This project demonstrated how the application of a number of computer vision techniques, combined with state estimation techniques, can result in a robust (and visually pleasing) system for tracking the movement of balls in a video scene. There is much room for improvement: it would be of great interest to remove the need to manually specify color ranges for balls, and many improvements could be made (as discussed earlier) to obtain more accurate observations for ball centers. Additionally, proper quantification of error covariances to allow varying Kalman gains could be very beneficial to tracking accuracy. However, the results as obtained are still satisfactory and seem promising for future applications.

The author gratefully acknowledges the help and the tutelage of the CEO of Sanskrut Corporation, Mr. Sandeep Koduri. The author is also thankful to Dr. Navneet Agrawal, asst. professor in the department of Electronics and Communication Engineering, CTAE, Udaipur for providing internship opportunities.

References

1. David Forsyth and Jean Ponce(2002). Computer Vision: A Modern Approach. Pearson Education publishers. ISBN 9780136085928.
2. Richard Szeliski (2010). Computer Vision: Algorithms and Applications. Springer-Verlag. ISBN 978-1848829343.
3. OpenCV documentation, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
4. Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing, 2nd edition, Pearson Education. ISBN 8131726959
5. Wikipedia, HSV colour model, https://en.wikipedia.org/wiki/HSL_and_HSV
6. Wikipedia, Euler method, https://en.wikipedia.org/wiki/Euler_method
7. Colour spaces and Colour models, https://www.drycreekphoto.com/Learn/color_spaces.htm
8. Morphological Image Processing, https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm
9. Kalman Filter tutorial, http://www.tina-vision.net/docs/memos/1996-002.pdf
10. Paul Zarchan and Howard Musoff, Frank K. Lu, Fundamentals of Kalman Filtering: A Practical Approach. American Institute of Aeronautics & Ast; 3rd edition (2009). ISBN 978-1600867187.
11. Python (programming language), https://en.wikipedia.org/wiki/Python_(programming_language)