Université de Bourgogne

VISUAL PERCEPTION

MSCV Semester II (Feb 2019 - Jun 2019)

IMAGE PROCESSING AND COMPUTER VISION TOOLBOX

Authors:
Nahid Nazifi
Vamshi Kodipaka
Inder Pal

Supervisor:
Dr. David Fofi

MSc Vibot/Computer Vision/MAIA

Contents

1 Introduction

Computer vision is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos. From engineering perspective, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.

Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

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.
Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, and image restoration.

2 Objective of This Project

The Main Objective of this project is to create a Computer vision Toolbox by using MATLAB. Computer Vision Toolbox provides algorithms, functions, and apps for designing and testing computer vision, 3D vision, and video processing systems. We can perform object detection and tracking, as well as feature detection, extraction, and matching. For 3D vision, the toolbox supports camera calibration, SFM and 3D reconstruction.

This Toolbox consist of 6 main tabs belonging to different category of operations. The first three tabs are for basic image processing. Fourth tab includes some basic features of computer vision like Camera Calibration and Structure from Motion. Last 2 tabs are dedicated to advanced computer vision includes moving object tracking, SLAM and Face Detection in videos etc.

3 Methodology and Implementation in MATLAB

We have used inbuilt toolboxes available in MATLAB in order to achieve this objective. Two main toolboxes which were the platform for us to use functions from and adapt them in our GUI to achieve the goal were :

1. Image Processing & Computer Vision Toolbox
2. MATLAB Support Package for USB WebCams

Our most of the reference material lies within *mathworks's* user guide for the toolbox which we have used. Every function of the toolbox has a capability to perform certain action within itself, which we have used, shuffled and adapted in order to build this toolbox. In the next section we will introduce all the functionalities of our toolbox along with the results.

4 Basic Image Processing tools

As we mentioned above, this section contains basic image processing tools in 4 tabs. The first tabs is **Basic Tab** which consist of **ColorSpace**, **Histogram**, and **Morphological Operations**.
https://www.overleaf.com/project/5cf7e3a3e2b1e030d9426e63 The Load Image in the panel is used mostly for this tab, once you load the image you can navigate through different Image Processing tools and obtaining different results.

4.1 ColorSpace

We can select different color spaces to convert images from one to another, like BGR to Gray,HSV and Ybcbr.

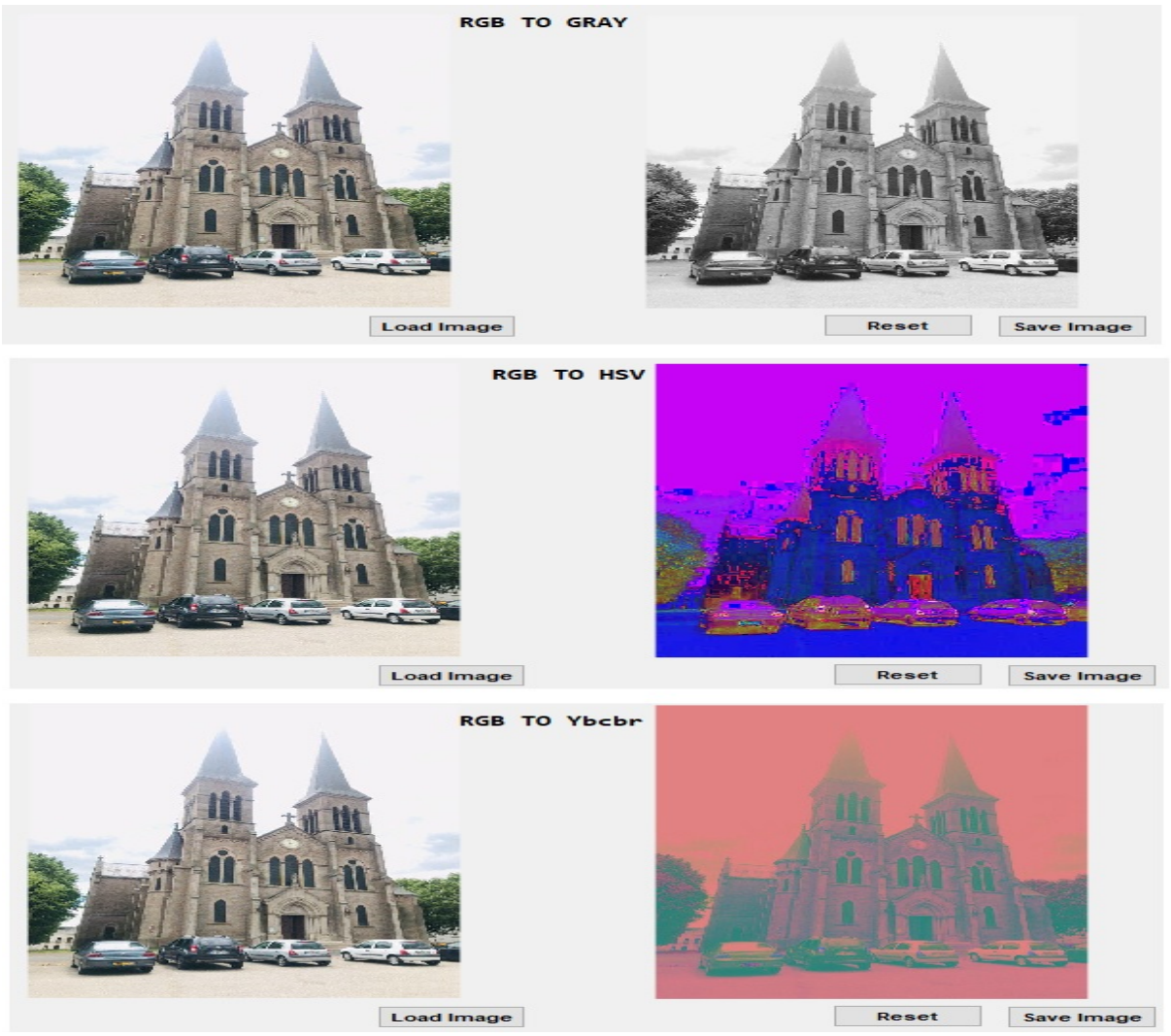

Figure 1: Different Color Spaces

4.2 Histogram

In our toolbox, we have provided **plotting Histogram** and **Equalizing Histogram**.

1. **Plotting Histogram** Shows pixel values (normally ranging from 0 to 255) and another way of understanding the image. shows intuition about contrast, brightness, intensity distribution etc of that image. Usually all image processing tools give histogram features.
2. **Equalizing Histogram** is a method in image processing of contrast adjustment using the image's histogram. It increases the global contrast of images, especially those with low contrast values. Through this adjustment, the intensities can be better distributed on the histogram. During this procedure, areas of lower local contrast gain a higher contrast. It works well for images which have both bright or both dark backgrounds and foregrounds.

Below we can see the plotted histogram of an image and the equalized histogram:

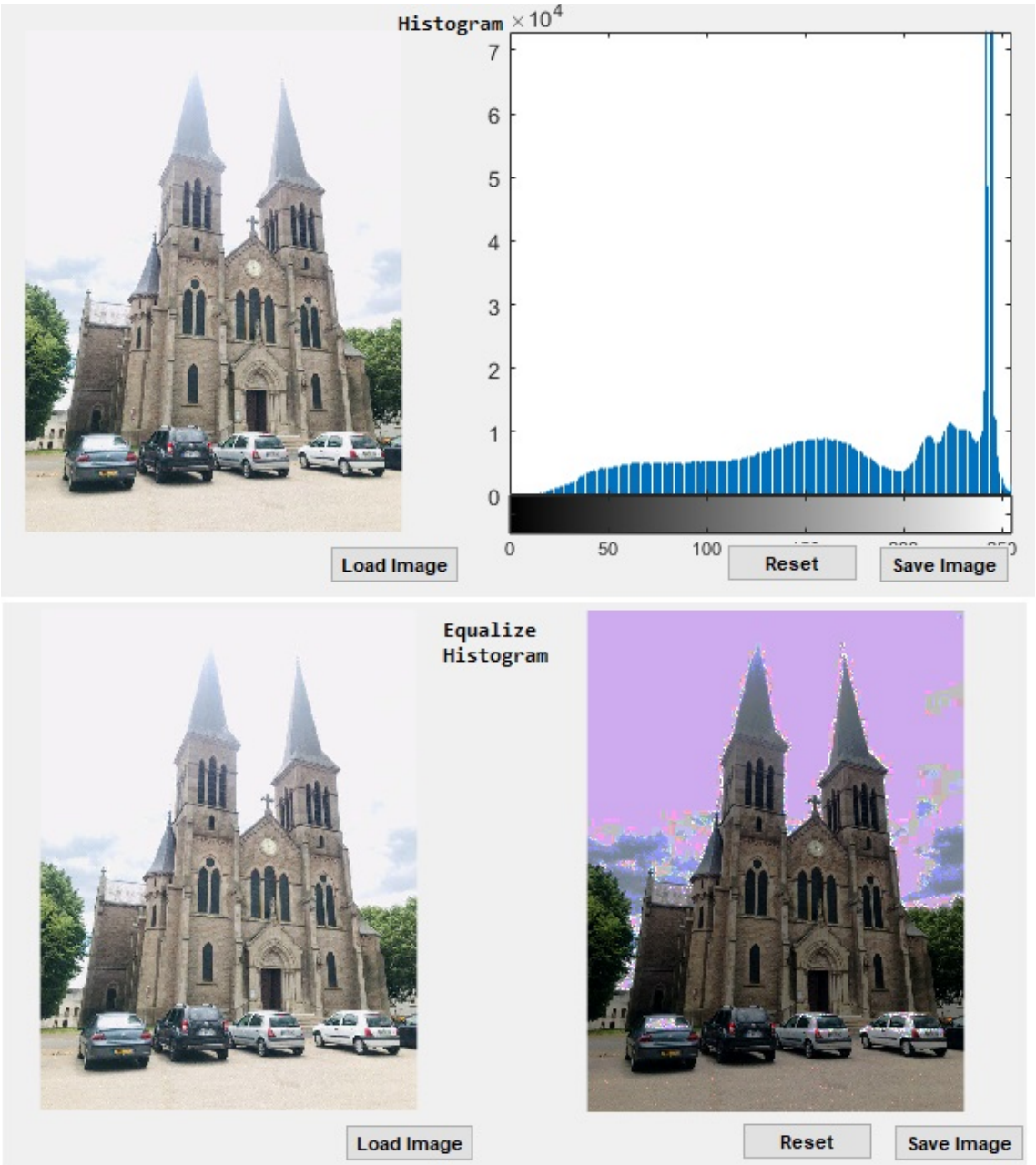

Figure 2: Histogram it's Equalization

4.3 Morphological transformations:

These operations are based on image shape. Normally they will apply on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. In this part, four morphological operators are shown.

- Dilate: It increases the white region in the image or size of foreground object increases.
- Erode: it erodes away the boundaries of foreground object
- Open: Opening is just another name of erosion followed by dilation. It is useful in removing noise
- Close: It is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object.

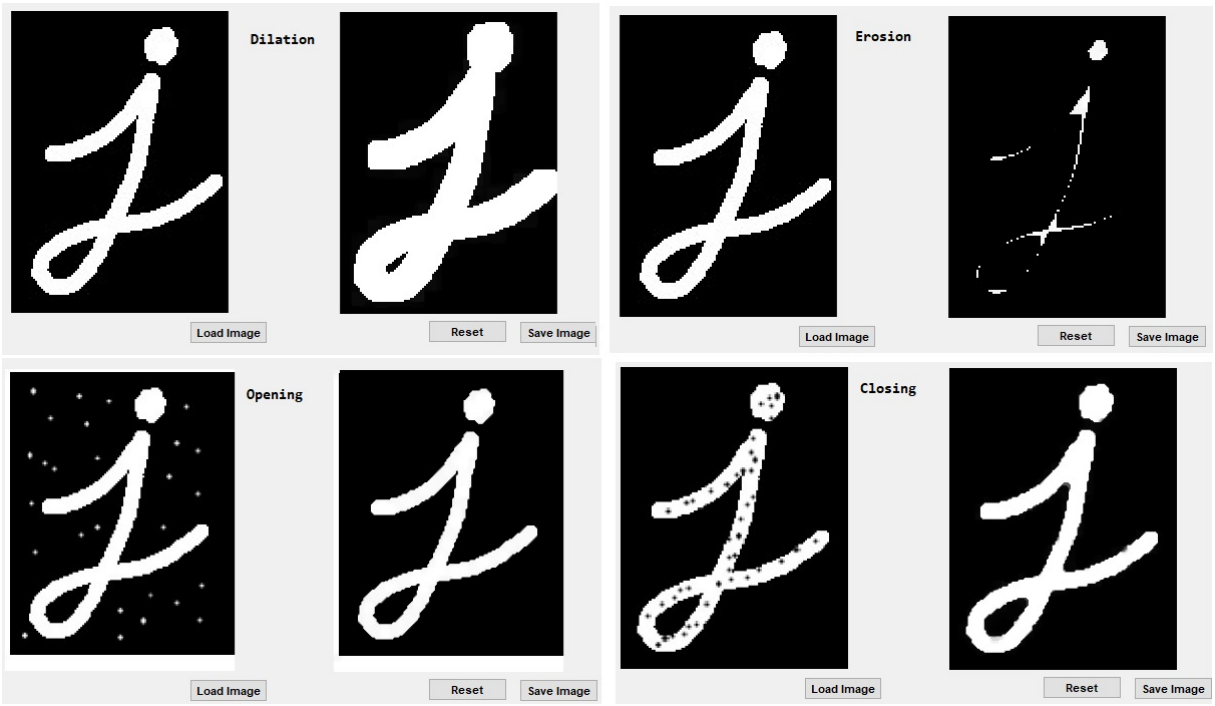

Figure 3: Morphological Operations

4.4 Edges and Lines

The second tab of the GUI is about Edges and lines. It consist of 3 sections: **Detect Edges**, **Detect Lines**, **Detect Circles:**.

4.4.1 Detect Edges:

This toolbox has 6 different algorithms for detecting edges:

1. **Canny** is a popular edge detection algorithm. It is a multi-stage algorithm which do Noise Reduction, Finding Intensity Gradient of the Image, Non-maximum Suppression and Hysteresis Thresholding.
2. **Sobel** operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

3. **Log:** Laplacian filters are derivative filters used to find areas of rapid change (edges) in images. Since derivative filters are very sensitive to noise, it is common to smooth the image (e.g., using a Gaussian filter) before applying the Laplacian. This two-step process is call the Laplacian of Gaussian (LoG) operation.

4. **Prewitt** operator is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function.

5. **Robert** is a differential operator which the idea behind the Roberts cross operator is to approximate the gradient of an image through discrete differentiation which is achieved by computing the sum of the squares of the differences between diagonally adjacent pixels.

6. **Zero-cross** It looks for places in the Laplacian of an image where the value of the Laplacian passes through zero, i.e. points where the Laplacian changes sign. Such points often occur at 'edges' in images, i.e. points where the intensity of the image changes rapidly, but they also occur at places that are not as easy to associate with edges. It is best to think of the zero crossing detector as some sort of feature detector rather than as a specific edge detector. Zero crossings always lie on closed contours, and so the output from the zero crossing detector is usually a binary image with single pixel thickness lines showing the positions of the zero crossing points.

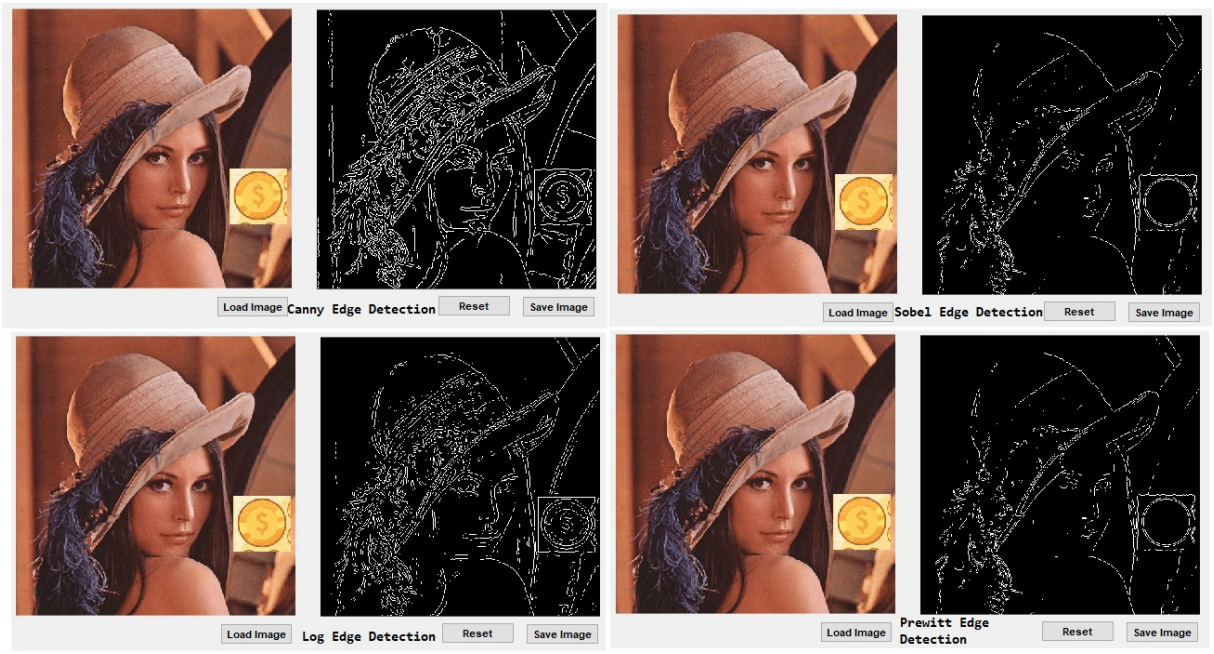

Figure 4: Edge Detection

4.5 Image Filtering

We divided the third tab which is related to image filters into two parts: **Basic Filters** and **Edge Preserving Filters**
In Basic Filters we put **Gaussian**, **Median**, **Box** and **Zero-Padding** filters.

1. **Gaussian:** imgaussfilt(A) filters image A with a 2-D Gaussian smoothing kernel with standard deviation of 0.5,

2. **Median:** medfilt2(I) performs median filtering of the image I in two dimensions. Each output pixel contains the median value in a 3-by-3 neighborhood around the corresponding pixel in the input image.

3. **Box:** imboxfilt(A) filters image A with a 2-D, 3-by-3 box filter. A box filter is also called a mean filter.

4. **Zero-Padding:**B = padarray(A,padsize) pads array A with 0s (zeros). padsize is a vector of nonnegative integers that specifies both the amount of padding to add and the dimension along which to add it.

On the other hand, in **Edge Preserving Filters** we put **Bilateral**, **Diffusion**, **Guided** and **Non-Local Means:**.

1. **Bilateral** is highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters.

2. **Diffusion:** In image processing and computer vision, anisotropic diffusion, also called Perona–Malik diffusion, is a technique aiming at reducing image noise without removing significant parts of the image content, typically edges, lines or other details that are important for the interpretation of the image.

3. **Guided:** The imguidedfilter function performs edge-preserving smoothing on an image, using the content of a second image, called a guidance image, to influence the filtering. The guidance image can be the image itself, a different version of the image, or a completely different image. Guided image filtering is a neighborhood operation, like other filtering operations, but takes into account the statistics of a region in the corresponding spatial neighborhood in the guidance image when calculating the value of the output pixel.

4. **Non-Local Means:** imnlmfilt(I) applies a non-local means-based filter to the grayscale or color image I

Figure 5: Image Filtering

4.6 Feature Detection

The Fourth tab in our GUI is dedicated to **Feature Detection** which is divided into 4 part:

- **Detect Features**
- **Object Detection**
- **Match Features**
- **Panorama**

4.6.1 Detect Features

In feature detection, we are looking for specific patterns or specific features which are unique, can be easily tracked and can be easily compared. Therefore, we check for the regions in images which have maximum variation when moved (by a small amount) in all regions around it. Finding these image features is called Feature Detection.
In the toolbox we implemented **BRISK**, **FAST**, **HARRIS**, **Min Eigen**,**MSER**, **SURF**, **KAZE** and **HOG** feature detections.

1. **SIFT:** A corner may not be a corner if the image is scaled. A corner in a small image within a small window is flat when it is zoomed in the same window. So Harris corner is not scale invariant. for keypoint detection and drawing them, first we have to construct a SIFT object. We can pass different parameters to it which are optional. SIFT function finds the keypoint in the images. Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighbourhood, angle which specifies its orientation, response that specifies strength of keypoints etc.

2. **SURF**: In SIFT, Lowe approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space. SURF goes a little further and approximates LoG with Box Filter. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral images. And it can be done in parallel for different scales. Also the SURF rely on determinant of Hessian matrix for both scale and location.

3. **FAST:** We showed some feature detectors which are good but not fast enough for a real-time application. For example, for SLAM (Simultaneous Localization and Mapping) mobile robot which have limited computational resources. So, FAST (Features from Accelerated Segment Test) algorithm was proposed.
   It is several times faster than other existing corner detectors. But it is not robust to high levels of noise. It is dependant on a threshold.

4. **BRISK**: The object contains information about BRISK features detected in a 2-D grayscale input image, I . The detectBRISKFeatures function uses a Binary Robust Invariant Scalable Keypoints (BRISK) algorithm to detect multiscale corner features.

5. **Min Eigen:** detectMinEigenFeatures(I) returns a cornerPoints object, points. The object contains information about the feature points detected in a 2-D grayscale input image, I. The detectMinEigenFeatures function uses the minimum eigenvalue algorithm developed by Shi and Tomasi to find feature points.

6. **MSER**: detectMSERFeatures(I) returns an MSERRegions object, regions, containing information about MSER features detected in the 2-D grayscale input image, I. This object uses Maximally Stable Extremal Regions (MSER) algorithm to find regions.

7. **KAZE**: detectKAZEFeatures(I) returns a KAZEPoints object containing information about KAZE keypoints detected in a 2-D grayscale image. The function uses nonlinear diffusion to construct a scale space for the given image. It then detects multiscale corner features from the scale space.

8. **Hog**: detectKAZEFeatures(I) returns a KAZEPoints object containing information about KAZE keypoints detected in a 2-D grayscale image. The function uses nonlinear diffusion to construct a scale space for the given image. It then detects multiscale corner features from the scale space.

9. **Harris Feature Detection**: It is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. Compared to the previous one, Harris' corner detector takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45 degree angles, and has been proved to be more accurate in distinguishing between edges and corners. Since then, it has been improved and adopted in many algorithms to preprocess images for subsequent applications.

Below we can see the results of different feature detection algorithms.

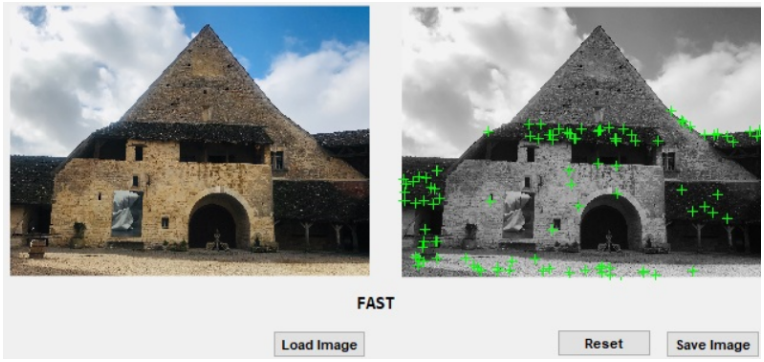

Figure 6: FAST Feature Detection

Figure 7: SURF Feature Detection

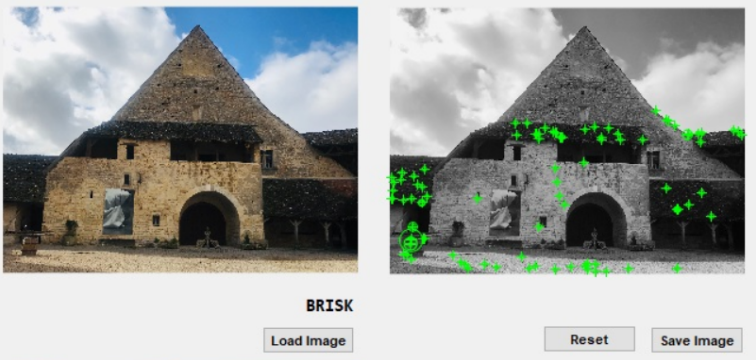

Figure 8: BRISK Feature Detection

Figure 9: Harris Feature Detection

4.7 Feature Matching

Feature matching is performed on two images that are rotated and scaled with respect to each other. Before the two images can be matched, feature points for each image must be detected and extracted. So detected features are described mathematically. The results are feature descriptors which find matches between two images. For two images we may get a set of pairs $(x_i, y_i) \leftrightarrow (x_i', y_i')$, where (x_i, y_i) is is a feature in one image and (x_i', y_i') is its matching feature in the other image.
As mentioned, feature detection and matching form an essential component of many computer vision applications. Examples include:

- Image alignment (e.g. for panoramic stitching)
- Video stabilization
- 3D model reconstruction from two or more views
- camera motion estimation (e.g. in robotic navigation or augmented reality)
- Object tracking
- Object recognition
- Large-scale image retrieval

Below we can see the results of different algorithms for feature matching:

Figure 10: FAST Feature Matching

Figure 11: SURF Feature Matching

Figure 12: Harris Feature Matching

Figure 13: Min Eigen Feature Matching

4.8 Object Detection

The goal of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image.

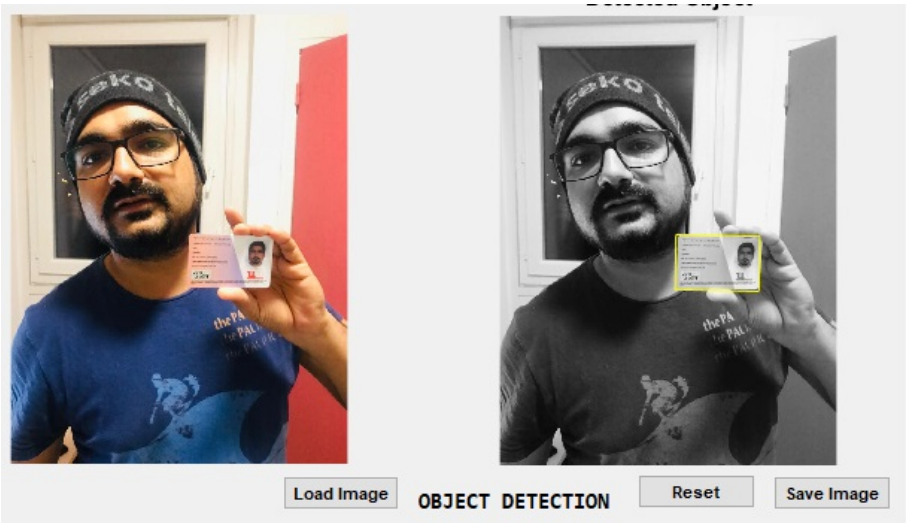

Figure 14: Object Detection

4.9 Panorama

Sometimes a single image can not capture a wide scene and we need to wider frame to record the image. By stitching, we can add more images together to create a large panorama. Below is the result of a panorama image.

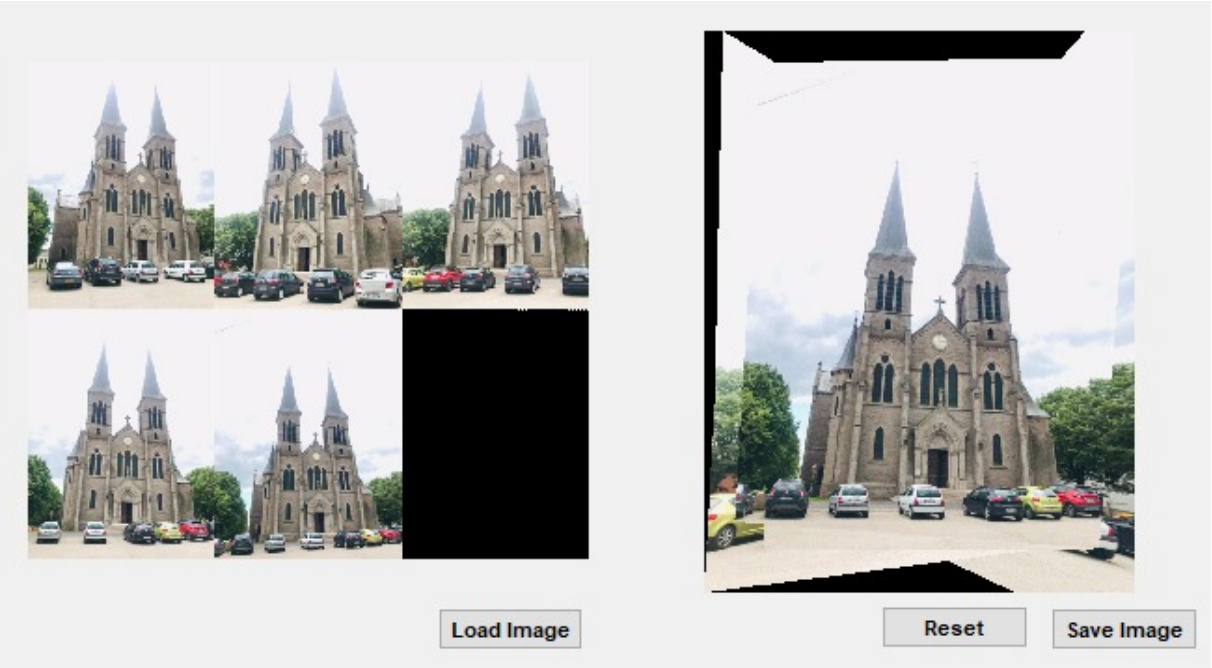

Figure 15: Panorama Image

5 Basic Computer Vision Tools

5.1 Camera Calibration

Camera calibration estimates the parameters of a lens and image sensor of an image or video camera. These parameters are used for lens distortion correction, measuring the size of an object in world units, or determining camera location in the scene. The applications are in machine vision to detect and measure objects, in navigation systems of robots, and 3-D scene reconstruction. Camera consist parameters such as intrinsics, extrinsics, and distortion coefficients. 3-D world points and their corresponding 2-D image points help for estimating camera parameters. You can get these correspondences using multiple images of a calibration pattern, such as a checkerboard. Using the correspondences, you can solve for the camera parameters.

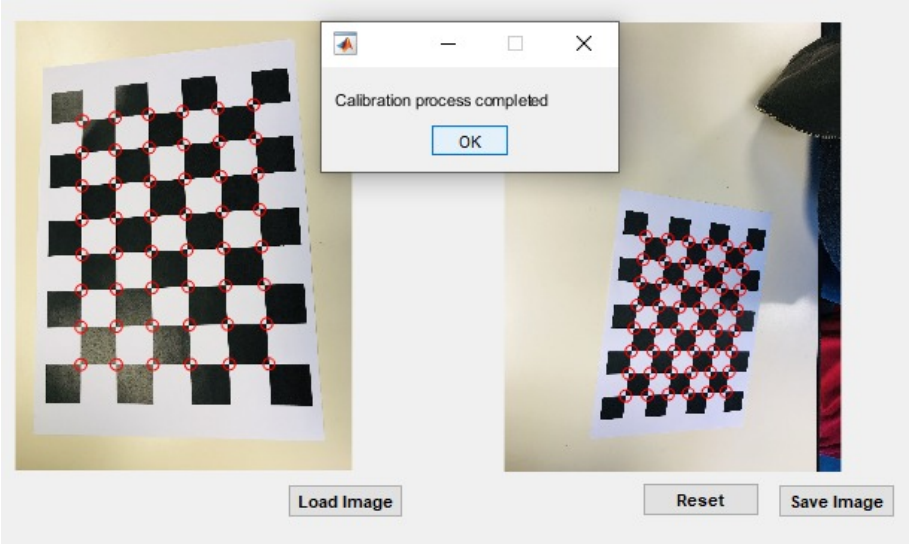

Figure 16: Camera Calibration

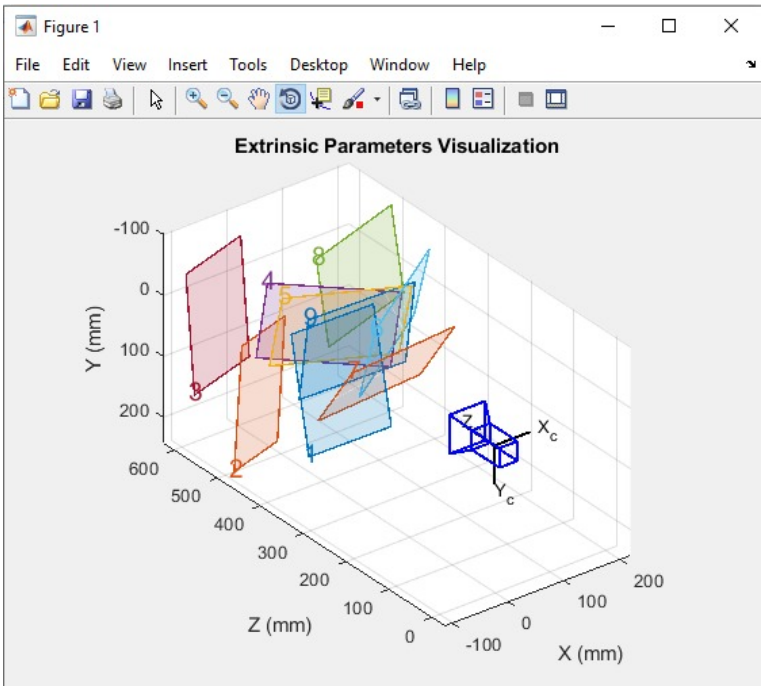

Figure 17: Extrinsic Parameter Result

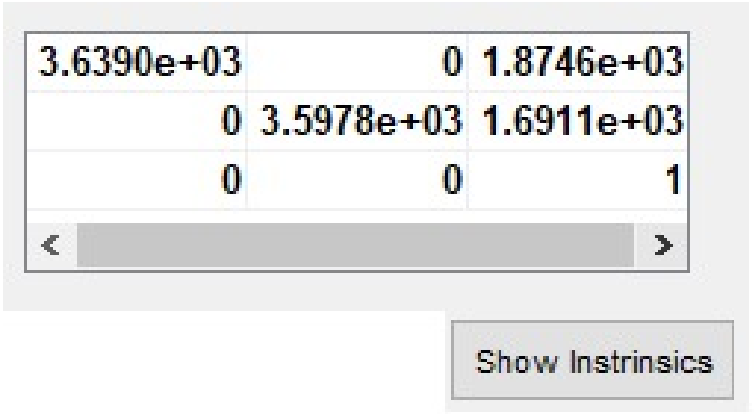

Figure 18: Intrinsic Parameter Result

After camera calibration, the accuracy of the estimated parameters can be evaluated by:

- Plotting the relative locations of the camera and the calibration pattern
- Calculating the re-projection errors
- Calculating the parameter estimation errors.

5.2 Structure From Motion

The process in which the 3-D structure of a scene from a set of 2-D images is estimated is called Structure from motion (SfM). It has many applications, such as 3-D scanning and augmented reality.

Computing SFM can be done in many ways. The way in which you approach the problem depends on different factors, such as the number and type of cameras used, and whether the images are ordered. In case of taking the images with one calibrated camera, recovering of 3-D structure and camera motion can be done up to scale. An illustration of 3-D reconstruction using MinEigen technique for feature detection and 2-D point evaluation is given below:

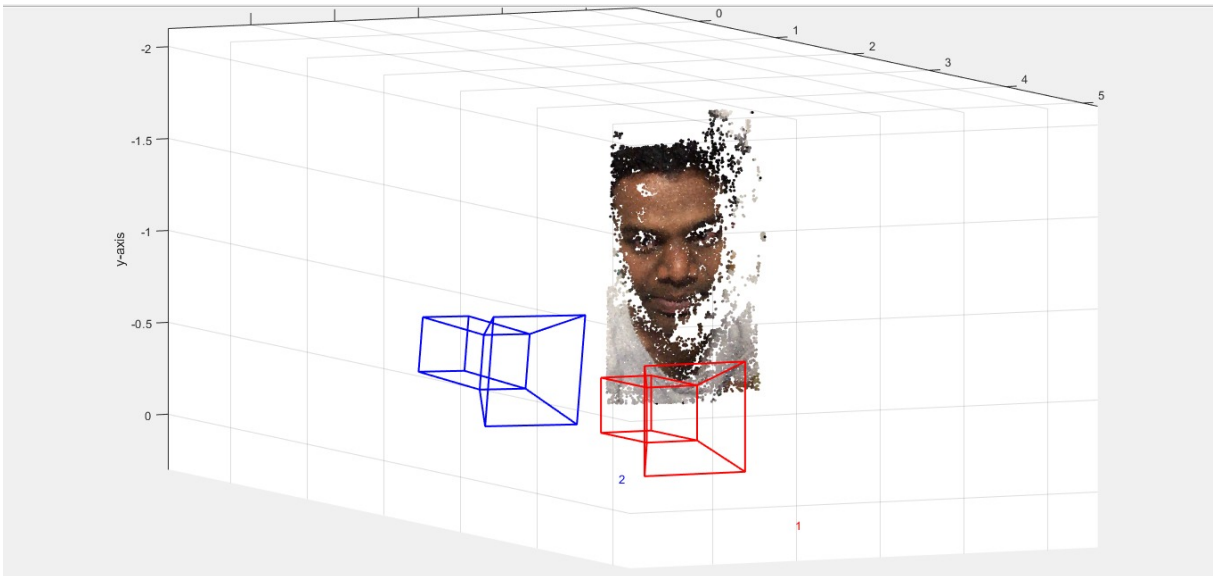

Figure 19: Structure from Motion

5.3 Epipolar Lines

When we take an image using pin-hole camera, we loose an important information, i.e depth of the image. Or how far is each point in the image from the camera because it is a 3D-to-2D conversion. We can find the depth information using more than one camera. Our eyes works in similar way where we use two cameras (two eyes) which is called stereo vision. image below which shows a basic setup with two cameras taking the image of same scene.
If we are using only the left camera, we can't find the 3D point corresponding to the point x in image because every point on the line OX projects to the same point on the image plane. But consider the right image also. Now different points on the line OX projects to different points (x') in right plane. So with these two images, we can triangulate the correct 3D point. This is the whole idea.

The projection of the different points on OX form a line on right plane (line l'). We call it epiline corresponding to the point x. It means, to find the point x on the right image, search along this epiline. It should be somewhere on this line. This is called Epipolar Constraint. Similarly all points will have its corresponding epilines in the other image. The plane XOO' is called Epipolar Plane.

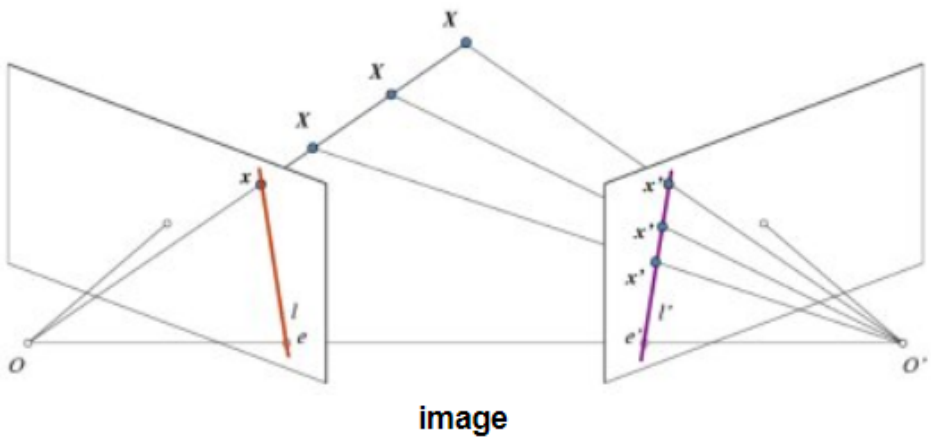

Figure 20: Epipolar Lines

6 Advanced Computer Vision Tools

This section consist of **Object and Face Detection In Videos**, **Moving Object Tracking** and **SLAM**.

6.1 Object Detection In Videos

Object detection is a branch of Computer Vision, in which visually observable objects that are in images of videos can be detected, localized, and recognized by computers. An image is a single frame that captures a single-static instance of a naturally occurring event . On the other hand, a video contains many instances of static images displayed in one second, inducing the effect of viewing a naturally occurring event.
Technically, a single static image in a video is called a video frame. In most videos, the number of frames in one second of the video ranges between 20 to 32, and this value is called the frames-per-second (fps).

Figure 21: Object Detection in Videos

6.2 Face Detection In Videos

Face Detection is the first and essential step for face recognition, and it is used to detect faces in the images and videos. It is a part of object detection and can use in many areas such as security, bio-metrics, law enforcement, entertainment, personal safety, etc.

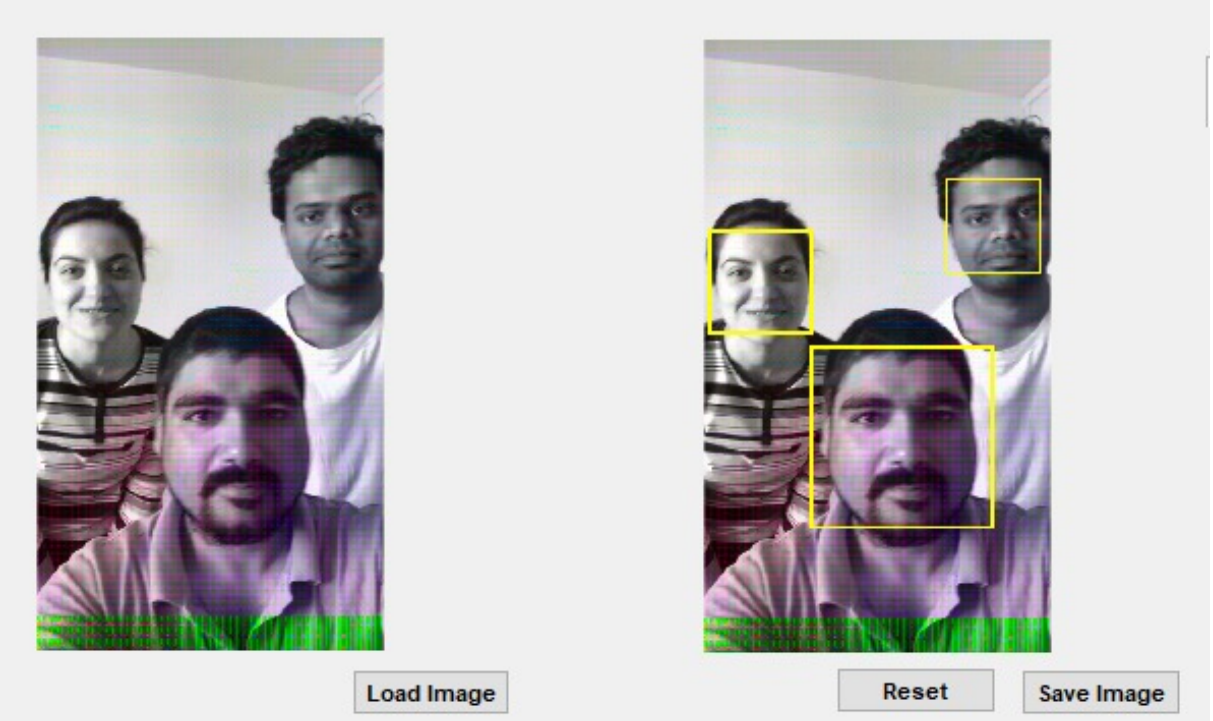

Figure 22: Face Detection in Videos

6.3 Moving Object Tracking

Moving Object Tracking in Video Sequence:

1. Choose an Video and choose an Object to track in that particular video
2. Do some basic filtering operations and Perform Segmentation of the Object
3. Localize the object. Here we have taken a yellow ball for experimentation.
4. Find the Centroid of the Object in each frame wise
5. Connect centroid frame-wise and track the path of the rolling ball.

Here in our experimentation, we use used a robust algorithm which can track the object path through the obstacles.

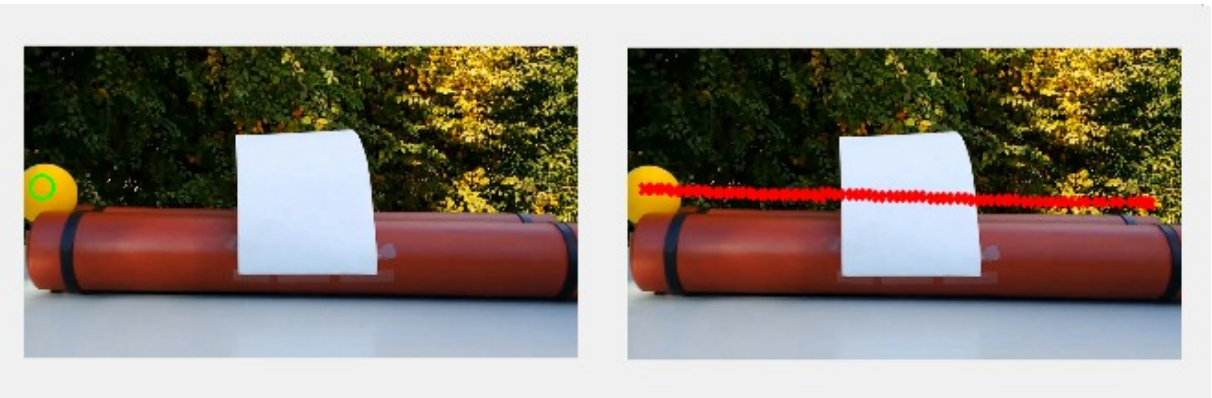

Figure 23: Tracking a moving object

6.4 SLAM

1. Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.
2. While this initially appears to be a chicken-and-egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments.
3. Popular approximate solution methods include the particle filter, extended Kalman filter, Covariance intersection, and GraphSLAM.

Here in our toolbox we have used Extended Kalman Filter for SLAM.

6.4.1 Basics of Kalman Filter

The Kalman Filter keeps track of the estimated state of the system and the variance or the uncertainty of the estimate. The estimate is updated using a state transition model and measurements. $\hat{X_{k|k+1}}$ denotes the estimate of the system's state at time step k before the k-th measurement y_k has been taken into account; $P_{k|k+1}$ is the corresponding uncertainty.

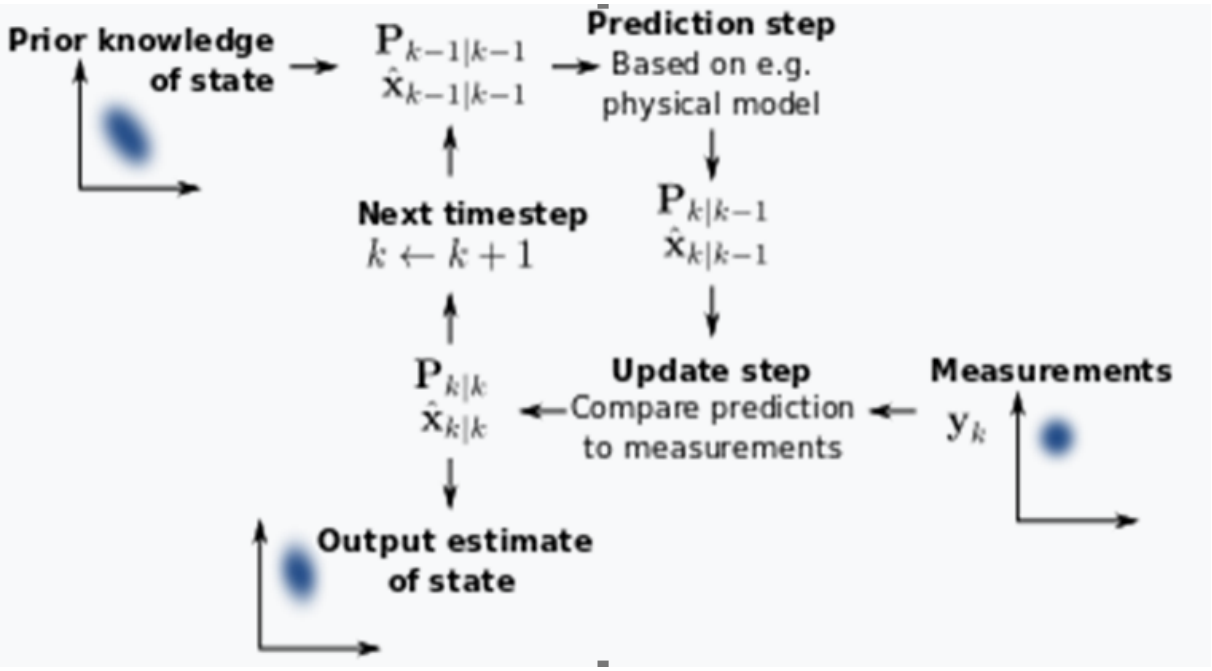

Figure 24

NOTE: Here, we have Sense and Move Operations in Prediction Step. State can be either Position(location) or Motion(velocity).

6.4.2 Extended Kalman Filter

It is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. In the case of well-defined transition models, the EKF has been considered the standardized in the theory of nonlinear state estimation, navigation systems and GPS. Extended Kalman filter uses Prediction and Estimation steps. For Formulation Refer[7]

6.4.3 SLAM Simulator

For Designing a SLAM Simulator the steps are as followed:

1. Declare Landmarks, Robot way-points, Obstacles.
2. Prediction by Sensing and Estimation with Precision in movement is what Extended Kalman Filter does.
3. We can make no. of iterations for a robots to move in the specified way-path
4. If robot is not given any loop, then one finds the Problem of Loop Closure

6.4.4 Loop Closure Problem:

It is state of robot where the robot cannot make the loop up to the starting point from the last way-point accurately.
However, a robust algorithms like SPM-SLAM and Uco-SLAM are developed which are 79% of Accuracy in Loop-Closure. But, for now we are just simulating the SLAM and SLAM is added as a Tools in our Project.

6.4.5 Highlight of SLAM Tool in our Toolbox:

1. Virtual Simulating of SLAM environment is possible.
2. Position errors, standard deviations, Scan Errors, Odometry errors, Robot Tuning Errors can be plotted by experimentation

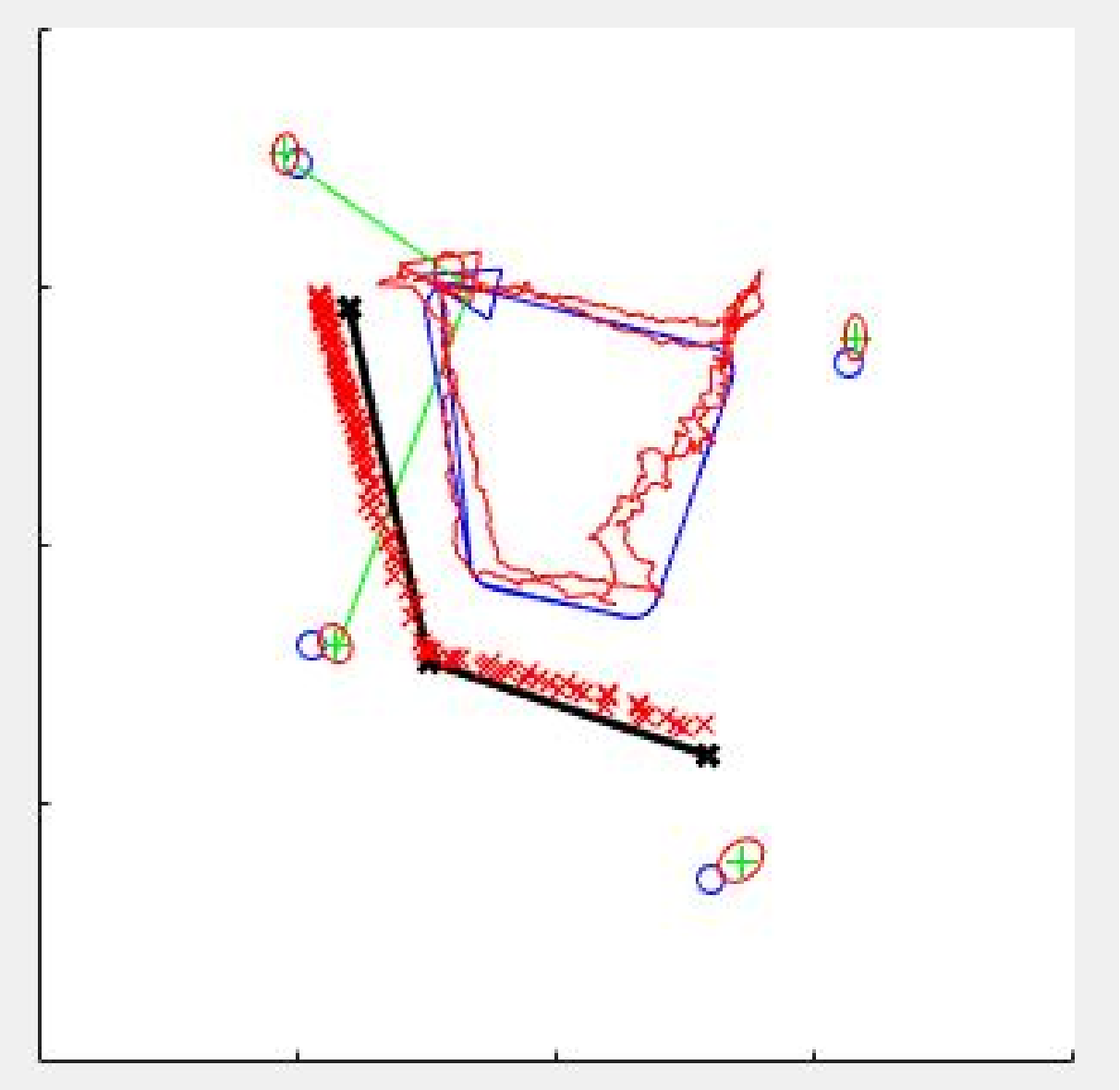

Figure 25: SLAM Results: a.) Blue line indicate – expected path b.) Red line indicate – real path c.) Black lines indicate – obstacles d.) Green lines indicate – neighboring landmarks e.) Red x marks indicates – the covariance/precision errors of the robot w.r.t the obstacles f.) Blue Circles are the landmark

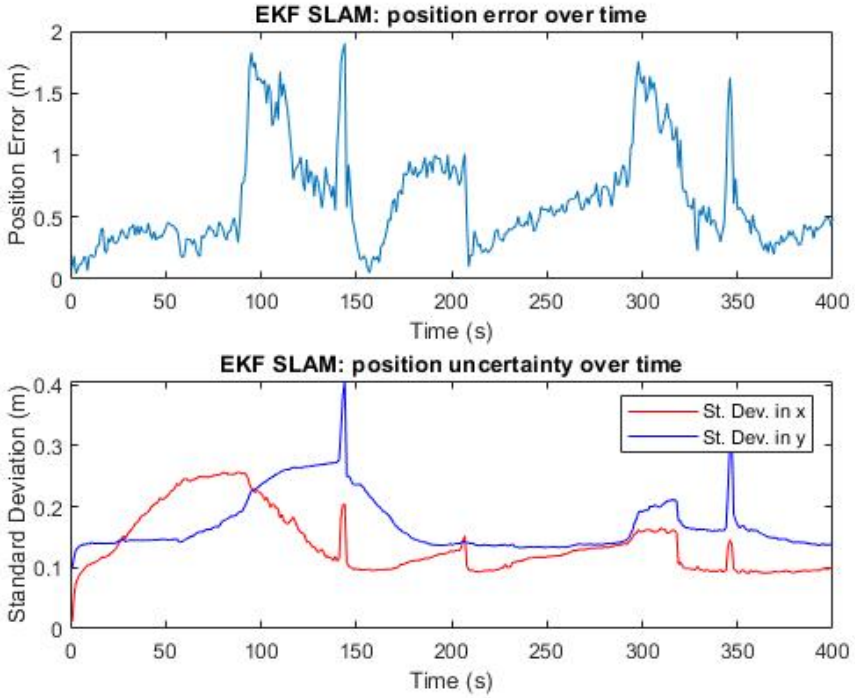

Figure 26: Plot-1: Position Error w.r.t Time; Plot-2: Standard Deviation w.r.t Time

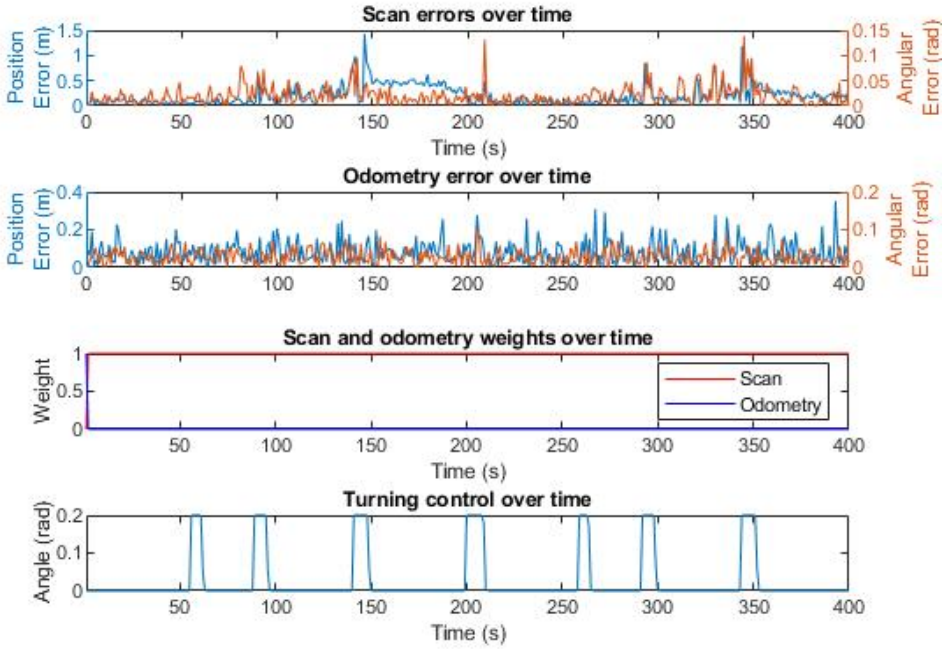

Figure 27: Plot-1: Scan errors w.r.t. to time; Plot-2: Odometry weights errors w.r.t. to time; Plot-3: Robot Turning i.e. Weighted control and Angle w,r,t to Time instances

7 Challenges and Conclusion

While achieving our objective of designing a computer vision toolbox using MATLAB, we have encountered several challenges. At the same time, as we were moving through each challenge, it was a turning into a learning for us.

First of all, it has given us an opportunity to make good GUIs. We have faced several challenges in making our GUI robust. For example: controlling the visibility of button groups or ui panels with selection of corresponding radio buttons. One major problem was to store the original image throughout the usage of toolbox.

We have tried to perform certain operations in the videos using the fact that a video is nothing but a combination of several frames in sequence. We have perform operations on individual frames to handle all video related operations.

We have also introduced the usage of inbuilt webcam to perform certain operations and as of now we are able to initiate the live video stream from webcam, capture images from the live video stream and save the captured image into the local drive.

For future works, we can perform all the toolbox operations on live video stream. As of now, it was not possible because the control over live video stream is not possible as we can not work on the frames (as we did in already loaded videos). It will be great achievement if these operations could be applied on live video streams as well.

At last, we would like to say that it was a great learning experience to see how theoretical things actually work in real practical world. We would like to thank our professor(s) who have made our basics clear and given an opportunity to make it happen.

8 References

[1]https://www.mathworks.com
[2]https://en.wikipedia.org/wiki/Homography_(computer_vision)
[3]https://en.wikipedia.org/wiki/Harris_Corner_Detector
[4]https://www.mathworks.com/help/vision/ug/structure-from-motion.html

https://heartbeat.fritz.ai/detecting-objects-in-videos-and-camera-feeds-using-keras-opencv
-and-imageai-c869fe1ebcdb
[6]https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping
[7]https://en.wikipedia.org/wiki/Extended_Kalman_filter
[8]Matlab online documentation