A Project Report on

CAMERA POSE ESTIMATION FOR AUGMENTED REALITY SYSTEMS

Submitted in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

BY

S.MOUNICA (1005-10-735025)

V.S.V. RAMA MADHURI (1005-10-735033)

S. SNEHA (1005-10-735038)

UNDER THE GUIDANCE OF

Mr. P. BALAKRISHNA REDDY
COMPUTER VISION ARCHITECT
THINC INNOVATIONS
HYDERABAD
&

Dr. P. ANANTH RAJ
Former Professor
DEPARTMENT OF ECE,
UNIVERSITY COLLEGE OF ENGINEERING (A)
OSMANIA UNIVERSITY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
UNIVERSITY COLLEGE OF ENGINEERING (AUTONOMOUS)
OSMANIA UNIVERSITY, HYDERABAD, AP-500007

2010-2014

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING (A)

OSMANIA UNIVERSITY, 500007

CERTIFICATE

This is to certify that the dissertation work entitled **"Camera Pose Estimation for Augmented Reality Systems"** is a bonafide work done and submitted by **S. MOUNICA, V.S.V. RAMA MADHURI, S. SNEHA** bearing roll numbers **1005-10-735025, 1005-10-735033, 1005-10-735038**, students of Department of Electronics and Communication Engineering, University College of Engineering (A), Osmania University, Hyderabad, in partial fulfillment of the academic requirements for the award of degree of Bachelor of Engineering in Electronics and Communication during the academic year 2010 – 2014.

Internal Guide

Dr. P. Ananth Raj,
Former Professor,
Department of ECE,
UCE (A),
Osmania University.

Head of Department

Dr. P. Chandra Sekhar,
Head,
Department of ECE,
UCE (A),
Osmania University.

DECLARATION

We hereby declare that the Project work entitled **"Camera Pose Estimation for Augmented Reality Systems"** is the bonafide work done and submitted by us under the esteemed guidance of **Mr. P. Balakrishna Reddy**, Computer Vision Architect, Thinc Innovations, Hyderabad, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering in Electronics and Communication.**

Date

Place

S. MOUNICA	**V.S.V. RAMA MADHURI**	**S. SNEHA**
(1005-09-735025)	**(1005-10-735033)**	**(1005-10-735038)**

ACKNOWLEDGEMENT

The completion of the project gives me an opportunity to convey our gratitude to all those who have helped us to reach a stage where we have confidence to launch our career in the competitive world.

We are extremely thankful to the Guide **Mr. P. Balakrishna Reddy**, Computer Vision Architect, Thinc Innovation, Hyderabad for his excellent guidance and constant encouragement throughout the project work. It was a pleasure and challenging experience to work under his supervision. His constant monitoring and the valuable advices ensured the completion of this thesis. We thank him for setting high standards and motivating us to do the best.

We express our deep sense of gratitude to **Dr. P. Ananth Raj**, Former Professor, Department of Electronics and Communication Engineering and **Dr. P. Chandrasekhar**, Head of Department and **Dr. R. Hemalatha**, Assistant Professor, Department of Electronics and Communication Engineering for providing necessary facilities in the department to complete our project successfully.

We would also like to thank our family for their support throughout the project.

S. MOUNICA	**V.S.V. RAMA MADHURI**	**S. SNEHA**
(1005-09-735025)	**(1005-10-735033)**	**(1005-10-735038)**

ABSTRACT

Augmented reality is a cutting edge technology that is budding up in the recent times. This is indeed an attractive and efficient system that allows for a digitally enhanced view of the real world, connecting you with more meaningful content in everyday life. In order to implement this, a real time pose estimation of the camera and information about the virtual object are required. This project presents a real-time camera-pose estimation method, based on multiple maps and local bundle adjustment which enables the registration to work without prior knowledge of natural scenes. This method can significantly enhance AR systems' usability. An iterative method is used to estimate the pose i.e. determine the translational and rotational parameters (also known as six degrees of freedom) of the camera. The camera coordinates are obtained from feature extraction and matching using Binary Robust Invariable Scalable Keypoints (BRISK).

Simultaneous localization and mapping (SLAM) is another technique which deals with the problem of building a map while at the same time localizing within that map. A good map is needed for localization while an accurate pose estimate is needed to build a map. The SLAM algorithm includes EKF (Extended Kalman Filter), state of the art methods like Inverse Depth Parameterization, 1-point RANSAC algorithm and quaternions.

By combining the results of BRISK and SLAM the matched features and the map in an unknown environment is obtained from which the camera pose estimation can be determined from the frames of a real video sequence, which is the main objective of our project.

In this project, BRISK and SLAM techniques are implemented using a single real time camera and softwares like Microsoft Visual Studio, Matlab and OpenCV.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 AUGMENTED REALITY

Since the inception of video games in the early 1970s, computer graphics have been entertaining us. Now, they have become much more sophisticated and are pushing the barriers of photorealism. Researchers and engineers are pulling graphics out of our television screen or computer display and integrating them into real-world environments. This new technology, called augmented reality, blurs the line between what's real and what's computer-generated. On the spectrum between virtual world and the real world, augmented reality is closer to the real world.

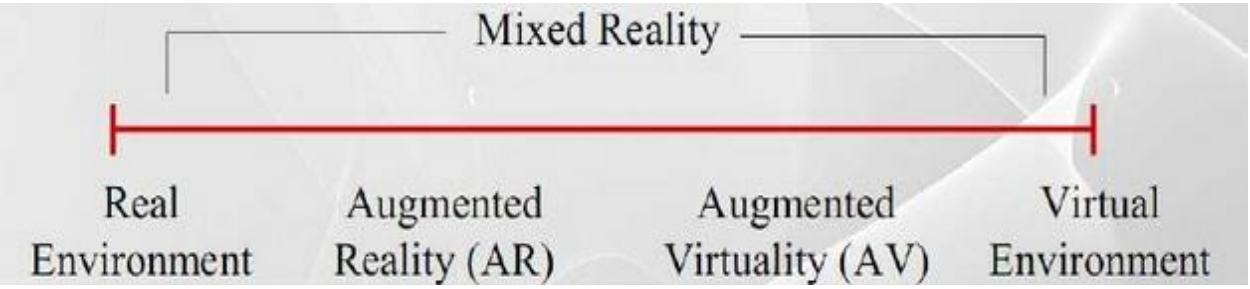


Figure 1: Spectrum of real and virtual environment

Figure 1 illustrates that with AR, users continue to be in touch with the real world while interacting with virtual objects around them. With VR, the user is isolated from the real world while immersed in a world that is completely fabricated.

Augmented reality adds graphics, sounds, haptic feedback to the natural world as it exists. As a result, the technology functions by enhancing one's perception of reality. Augmentation is conventionally in real-time and in semantic context with environmental elements, such as sports scores on TV during a match. With the help of advanced AR Technology the information about the surrounding real world of the user becomes interactive and digitally manipulable. Everyone from tourists, to soldiers, to someone looking for the closest subway stop can now benefit from the ability to place computer-generated graphics in their field of vision.

While virtual object insertion has been demonstrated before, and is even part of commercial video editing products nowadays, but accurate registration into a real environment with less computational efforts is an outspoken problem in Augmented Reality (AR). This problem needs to be solved regardless of the complexity of the virtual objects one wishes to enhance the real environment with. Both simple text annotations and complex virtual mimics of real-life objects need to be placed rigidly into the real environment. *Augmented Reality Systems* that lack this requirement will demonstrate serious 'jittering' of virtual objects in the real environment and will therefore fail to give the user a real-life impression of the augmented outcome. The registration problem has already been tackled by several researchers in the AR domain.

Other techniques have been devised to make the calibration of the video camera obsolete by using affine object representations. These techniques are simple and fast but fail to provide a real impression when projective skew is dominant in the video images. Therefore virtual objects can be viewed correctly only from large distances where the affine projection model is almost valid. So it seems that the most flexible registration solutions are those that don't depend on any a priori knowledge of the real environment and use the full perspective projection model. The modules of AR systems proposed in this project work belong to this class of flexible solutions. The importance of camera pose estimation in Augmented Reality is explained in next topic.

1.2 CAMERA-POSE ESTIMATION

Augmented reality is an emerging technology which superimposes the view of the real world by additional virtual computer generated objects. By augmenting the experience of the real world with computer generated sensations, virtual information can behave like a real object and users can leverage their knowledge of the real world to interact with that information. Achieving accurate registration between real and synthetic worlds is one of augmented reality's biggest challenges.

The cores of Augmented Reality applications are often based on recognition and pose estimation to allow the appropriate virtual content to be registered and augmented onto the real

world. So, the real time camera pose estimation plays a very crucial role for augmented reality systems which is a cutting-edge technology that allows for a digitally enhanced view of the real world.

A real-time camera-pose estimation method enables the registration to work without prior knowledge of natural scenes. It is an important part of many applications as for example structure-from-motion, Augmented Reality and other applications that involve 3D object or camera tracking. This method can significantly enhance AR systems' usability and also useful for motion analysis, robotic control tasks, automatic navigation, etc.

During the last years, several approaches based on natural features (points, planes, edges, etc.) extracted from the image scene have been developed. The mean idea of these techniques is to locate the camera in three dimensions given a sequence of 2D intensity images of the scene containing the 3D features whose positions and orientations are known relative to a reference frame. Often these applications require short processing time per image frame or even real-time constraints. In that case pose estimation algorithms are of interest, which are accurate and fast. SLAM technique is one of them which provide accurate results with less computational efforts. The objective of this project is stated in next topic.

1.3 OBJECTIVE

In order to estimate the camera-pose for an augmented reality system, the following steps are to be followed:

- Develop a code in Microsoft Visual Studio for extracting features and building descriptors from images of the video sequence through BRISK Algorithm. Obtain the best matches of the features.
- Develop a matlab code and implement SLAM technique on a video and obtain the trajectory of camera.
- Estimating the pose of the camera using BRISK features in SLAM i.e., obtain the translational and rotational parameters (also known as 6 degrees of freedom).

In this project, we have selected only the first two steps. Hence, explanation and simulation results on the below two topics are presented in this thesis.

- Development of code in Microsoft Visual Studio for detecting, extracting and matching features using BRISK algorithm.
- Development and testing of matlab code of SLAM and obtain the trajectory of camera.

The above two steps are explained in further chapters.

1.4 BRISK

Decomposing an image into local regions of interest or 'features (keypoints)' is a widely applied technique in Computer Vision and is used to alleviate complexity while exploiting local appearance properties. Many computer vision applications and automatic collimation procedures in photogrammetry, such as image orientation, Digital Surface Model (DSM) generation, 3D reconstruction and motion tracking, model based recognition, texture recognition, robot localization, etc., rely on the presence of stable, representative features in the image, driving research and yielding a plethora of approaches to this problem.

The ideal keypoint detector finds salient image regions such that they are repeatedly detected despite change of viewpoint; more generally it is robust to all possible image transformations. Similarly, the ideal keypoint descriptor captures the most important and distinctive information content enclosed in the detected salient regions, such that the same structure can be recognized if encountered. Moreover, on top of fulfilling these properties to achieve the desired quality of keypoints, the speed of detection and description also needs to be optimized to fit within the time constraints of the task at hand.

The inherent difficulty in extracting suitable features from an image lies in balancing two competing goals: high quality description and low computational requirements. The techniques developed over the last 25 years allow the automatic extraction of homologous points in normal stereoscopic conditions. On the other hand, these algorithms are unable to give good results if the convergent taking geometry, strong affinity transformations and 3D illumination changes affect

the image acquisition. These methods are not sensitive to image scale changes, rotations, affine distortions, or illumination changes.

This is where we aim to set a new milestone with the BRISK methodology. It achieves comparable quality of matching with respect to state-of-the-art methods (like SURF, SIFT, etc.) at much less computation time.

1.5 SLAM

Estimation of relative 3D camera position and orientation (pose) is one of the most challenging problems in computer vision. Indeed, the knowledge of the camera pose is useful for numerous applications, including motion analysis, robotic control tasks, and Augmented Reality (AR). During the last years, several approaches based on natural features (points, planes, edges, etc.) extracted from the image scene have been developed. The mean idea of these techniques is to locate the camera in three dimensions given a sequence of 2D intensity images of the scene containing the 3D features whose positions and orientations are known relative to a reference frame.

When the correspondences between 2D features extracted from the image and 3D features defined in the world frame are established, the problem is then solved using 2D-3D registration techniques. The pose recovery of the camera is obtained from essential matrix decomposition. Line-based methods are more robust and more efficient than point-based techniques. The camera pose is computed by minimizing the distances between the projection of the model lines and the most likely matches found in the image.

Many techniques like Structure from motion are being used in the automatic reconstruction of the trajectory of video camera. But SLAM is a technique where the sequential estimation proceeds by updating the state estimate. The processing time per image is less and the real-time constraint is not breached. It is an accurate technique that operates with minimum computational effort.

1.6 OPENCV LIBRARY

Open CV stands for Open Source Computer Vision Library. It is essentially a library which contains various functions and header files for Image Processing techniques. Microsoft Visual Studio, being a platform is used to write the C code using the function of OPEN CV.
The basic features of OPEN CV can be broadly classified into:

Core module: The Core Functionality:
It describes about the basic building blocks of the library. It helps when there is a need to manipulate the images on a pixel level. This describes various operations on image like adding to images, changing brightness and contrast of the image.

Imgproc module. Image Processing:
This module deals with the various image processing techniques (manipulate the images on a pixel level). It describes various operations on image like erosion, dilation, smoothing images, sobel derivatives, canny edge detector etc.

highgui module. High Level GUI and Media:
This file deals how to read/save your image/video files and how to use the built-in graphical user interface of the library. It describes various video operations like creation of videos.

calib3d module. Camera calibration and 3D reconstruction:
Although most of the times the desired images are of 2D format they does come a need for a 3Dworld. It deals as to how to find out from the 2D images information about the 3D world.

Feature2d module. 2D Features framework:
Learn about how to use the feature point detectors, descriptors and matching framework found inside Open CV. It describes various algorithms related to Harris corner detector, Shi-Tomasi corner detector, detection of planar objects.

video module. Video analysis:

Look here in order to find use on your video stream algorithms like: motion extraction, feature tracking and foreground extractions.

objdetect module. Object Detection:

Here the algorithms used by digital camera to detect peoples and faces are described. It deals with Cascade Classifier to detect objects in a video stream.

ml module. Machine Learning

It uses the powerful machine learning classes for statistical classification, regression and clustering of data.

gpu module. GPU-Accelerated Computer Vision

It squeezes out every little computation power from the system by using the power of the video card to run the Open CV algorithms.

The various features stated above can be selected by using the various header files from the following:

opencv_calib3d243d.lib
opencv_contrib243d.lib
opencv_core243d.lib
opencv_features2d243d.lib
opencv_flann243d.lib
opencv_gpu243d.lib
opencv_haartraining_engined.lib
opencv_highgui243d.lib
opencv_imgproc243d.lib
opencv_legacy243d.lib
opencv_ml243d.lib
opencv_objdetect243d.lib
opencv_ts243d.llib
opencv_video243d.lib

The next chapter gives the literature survey of BRISK and SLAM algorithms.

2. LITERATURE SURVEY

2.1 BRISK

Identifying local interest points to be used for image matching can be traced a long way back in the literature, with Harris and Stephens proposing one of the earliest and probably most well-known corner detectors in “A combined corner and edge detector”. The seminal work of Mikolajzyk et al. in “A comparison of affine region detectors” presented a comprehensive evaluation of the most competent detection methods at the time, which revealed no single all-purpose detector but rather the complementary properties of the different approaches depending on the context of the application. The more recent FAST criterion in “Machine learning for high speed corner detection” for keypoint detection has become increasingly popular in state-of-the-art methods with hard real-time constraints, with AGAST in “Adaptive and generic corner detection based on the accelerated segment test” extending this work for improved performance.

Amongst the best quality features currently in the literature is the SIFT in “Distinctive image features from scale-invariant keypoints”. The high descriptive power and robustness to illumination and viewpoint changes has rated the SIFT descriptor at the top of the rankings list in the survey in “A performance evaluation of local descriptors”. However, the high dimensionality of this descriptor makes SIFT prohibitively slow. PCA-SIFT in “PCA-SIFT: A More Distinctive Representation for Local Image Descriptors” reduced the descriptor from 128 to 36 dimensions, compromising however its distinctiveness and increasing the time for descriptor formation which almost annihilates the increased speed of matching. The GLOH descriptor “A performance evaluation of local descriptors” is also worth noting here, as it belongs to the family of SIFT-like methods and has been shown to be more distinctive but also more expensive to compute than SIFT.

The growing demand for high-quality and high-speed features has led to more research towards algorithms able to process richer data at higher rates. Notable is the work of Agrawal et al. in “CenSurE: Center surround extremas for real-time feature detection and matching” who apply a center-symmetric local binary pattern as an alternative to SIFT’s orientation histograms

approach. The most recent BRIEF in “BRIEF: Binary Robust Independent Elementary Features” is designed for super-fast description and matching and consists of a binary string containing the results of simple image intensity comparisons at random pre-determined pixel locations. Despite the simplicity and efficiency of this approach, the method is very sensitive to image rotation and scale changes restricting its application to general tasks.

Probably the most appealing features at the moment are the SURF in “SURF: Speeded up robust features”, which have been demonstrated to be significantly faster than SIFT. SURF detection uses the determinant of the Hessian matrix (blob detector), while the description is done by summing Haar wavelet responses at the region of interest. While demonstrating impressive timings with respect to the state-of-the-art (SURF) in terms of speed, still orders of magnitude away from the fastest, yet limited quality features currently available.

In this paper, we present a novel methodology dubbed ‘BRISK’ for high-quality, fast keypoint detection, description and matching. As suggested by the name, the method is rotation as well as scale invariant to a significant extent, achieving performance comparable to the state-of-the-art while dramatically reducing computational cost.

2.2 SLAM

Structure from motion is a widely used technique in camera-pose estimation. Structure from motion research in computer vision has reached the point where fully automated reconstruction of the trajectory of a video camera moving through a previously unknown scene is becoming routine but these and other successful approaches seen to date have been formulated as off-line algorithms and required batch, simultaneous processing of all the images acquired in the sequence. Of course batch processing provides the most accurate and robust solution to any estimation problem in applications where off-line operation is satisfactory. Real-time operation, however, enforces hard constraints on the processing permissible: specifically, in the common case in which data arrives at a constant rate (e.g. from a camera at 30Hz) the estimation must operate in constant time, requiring an amount of processing bounded by a constant to take account of the pertinent information available from each image. The value of this constant will

depend on the details of the processor available but significantly the processing time per image cannot increase indefinitely with time otherwise a point will always be reached at which the real-time constraint is breached.

Rather than storing a history of measurements we are led towards a time-independent state- based representation, in which everything of interest about the system in question is captured in a snapshot of the current instant. Sequential estimation then proceeds at each time step by updating the state estimate due to the effect of the passage of time and any new data obtained. And in sequential estimation and updating the state estimate, SLAM technique through the use of Extended Kalman Filter is more efficient and accurate.

The work of Harris and Pike in "3D Positional Integration from Image Sequences", whose DROID system built visual maps sequentially using input from a single camera, is perhaps the grandfather of our research and was far ahead of its time. Impressive results showed 3D maps of features from long image sequences, and a later real-time implementation was achieved. A serious oversight of this work, however, was the treatment of the locations of each of the mapped visual features as uncoupled estimation problems, neglecting the strong correlations introduced by the common camera motion. Closely-related approaches were presented by Ayache in "Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception" and later Beardsley et al. in "Active Visual Navigation Using Non-Metric Structure" in an un-calibrated geometrical framework, but these approaches also neglected correlations, the result being overconfident mapping and localization estimates and an inability to close loops and correct drift.

Smith et al. in "A Stochastic Map for Uncertain Spatial Relationships" and, at a similar time, Moutarlier and Chatila in "Stochastic Multisensory Data Fusion for Mobile Robot Location and Environment Modeling", had proposed taking account of all correlations in general robot localization and mapping problems within a single state vector and covariance matrix updated by the Extended Kalman Filter (EKF). Work by Leonard in "Directed Sonar Sensing for Mobile Robot Navigation", Manyika in "An Information-Theoretic Approach to Data Fusion and Sensor Management", and others demonstrated increasingly sophisticated robot mapping and

localization using related EKF techniques, but the single state vector and "full covariance" approach of Smith et al. did not receive widespread attention until the mid to late 1990s, perhaps when computing power reached the point where it could be practically tested. Several early implementations proved the single EKF approach for building modest-sized maps in real robot systems and demonstrated convincingly the importance of maintaining estimate correlations. These successes gradually saw very widespread adoption of the EKF as the core estimation technique in SLAM and its generality as a Bayesian solution was understood across a variety of different platforms and sensors.

In the intervening years, SLAM systems based on the EKF and related probabilistic filters have demonstrated impressive results in varied domains. The methods deviating from the standard EKF have mainly aimed at building large scale maps, where the EKF suffers problems of computational complexity and inaccuracy due to linearization, and have included sub mapping strategies and factorized particle filtering. The most impressive results in terms of mapping accuracy and scale have come for camera pose estimation for Augmented Reality systems. The first step of project i.e., BRISK is explained in next chapter.

3. BINARY ROBUST INVARIANT SCALABLE KEYPOINTS

3.1 INTRODUCTION

The main steps involved in BRISK (Binary Robust Invariant Scalable Keypoints) are: Feature detection, extraction and matching. A feature is defined as an "interesting" part of an image. It can be a point, edge, corner, blob, etc.

Feature Detectors:

- Feature detection is a low-level image processing operation. That is, it is usually performed as the first operation on an image, and examines every pixel to see if there is a feature present at that pixel. If this is part of a larger algorithm, then the algorithm will typically only examine the image in the region of the features.
- Many computer vision algorithms use feature detection as the initial step, so as a result, a very large number of feature detectors have been developed. The common detectors like Canny, Sobel, Harris, etc. detect the edges as features and certain algorithms like FAST, Shi & Tomasi, Laplacian of Gaussian, etc. detect the corners as features whereas Difference of Gaussian and Determinant of Hessian detect the blobs as features.

Feature Extraction:

- In Image Processing, feature extraction is a process of reducing the number of random variables under construction i.e., selecting only robust features.
- The feature vectors or feature descriptors are a set of characteristic attributes of the features, and each of which is invariant to image translation, scaling, and rotation, partially invariant to illumination changes and robust to local geometric distortion.
- Many feature extraction techniques like SIFT, SURF, etc. are being used to build the descriptors. But BRISK is adaptive and gives high quality performance at lower computational cost when compared to other methods.

Feature Matching:

- Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images. The approach we take depends partially on the application, e.g., different strategies may be preferable for matching images.
- The Hamming distance method is used to match the descriptors.
- Feature descriptors have been designed so that Euclidean (vector magnitude) distances in feature space can be used for ranking potential matches.
- The strategy is to set a threshold (maximum distance) and to return all matches from other images within this threshold.

The basic goal of these algorithms is the detection of image regions that are invariant under certain transformations feature extraction methods, called region detectors, must be invariant under a generic camera movement, such as scale change, rotation, and 3D viewpoint changes. The first step of BRISK i.e. feature detection is explained in next topic.

3.2 FEATURE DETECTION

3.2.1 Introduction:

Feature detection is a process in Computer Vision that aims to find visual features within the image with particular desirable properties. Detected features are some subsection of the image and can be one of the following:

- Points (Ex: Harris corners)
- Connected image regions
- Continuous curves in the image

The important information about visual features can include co-ordinates in the image, radius of the visual feature, scale (octave of the image). After detecting the feature with some feature detection algorithm, it is usually described by a descriptor vector for the purposes of

visual feature matching, while the feature's position in the image can be used directly for application in tracking.

In this detection process, we need to detect the keypoints with robust corners and scale invariant points. Many algorithms, such as FAST, AGAST, FREAK, BRIEF, DAISY, SURF, SIFT are used. But FAST detector is more efficient than the others.

3.2.2 FAST:

Features from Accelerated Segment Test (FAST) is an algorithm defined for identifying interest points in an image. An interest point in an image is a pixel which has a well-defined position and can be robustly detected. Interest points have high local information content.

There are several well established algorithms like Harris & Stephens corner detection algorithm, SUSAN corner detector. The reason behind the work of the FAST algorithm was to develop an interest point detector for use in real time frame rate applications which have limited computational resources. FAST corner detector is very suitable for real-time video processing application because of high-speed performance. Keypoints are detected based on the intensity at each pixel using Scale Space Pyramid.

Feature detection using FAST:

The FAST corner detection algorithm is given below:

- Given an image, select a pixel 'p' in the image. Assume the intensity of this pixel to be Ip. This is the pixel which is to be identified as an interest point or not.
- Set a threshold intensity value T, (say 20% of the pixel under test).
- Consider a circle of 16 pixels surrounding the pixel p.
- 'N' contiguous pixels out of the 16 need to be either above or below Ip by the value T, if the pixel needs to be detected as an interest point.

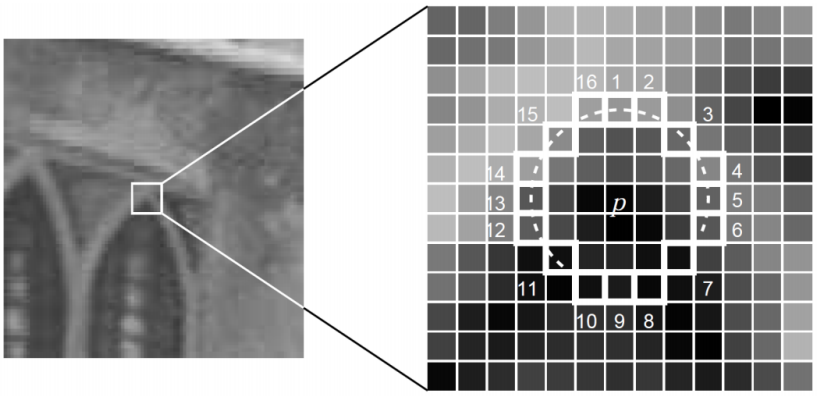


Figure 2: Interest point under test and the 16 pixels on the circle.

Figure 2 illustrates 12 point segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection. The pixel at p is the centre of a candidate corner. The arc is indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold.

In order to make the algorithm fast, first compare the intensity of pixels 1, 5, 9,13of the circle with Ip. As evident from the figure above, at least three of these four pixels should satisfy the threshold criterion so that the interest point will exist.

- If at least three of the four pixel values - 1, 5,9,13 are not above or below Ip + T, then p is not an interest point (corner). In this case reject the pixel p as a possible interest point. Else if atleast three of the pixels are above or below Ip + T, then check for all 16 pixels and check if 12 contiguous pixels fall in the criterion.
- Repeat the procedure for all the pixels in the image.

Scale Space Keypoint Detection:

In order to achieve invariance to scale which is crucial for high-quality keypoints, we implement this technique by searching for maxima not only in the image plane, but also in scale-space using the FAST scores as a measure for saliency. Despite discretizing the scale axis at coarser intervals than in alternative high-performance detectors, the BRISK detector estimates the true scale of each keypoint in the continuous scale-space.

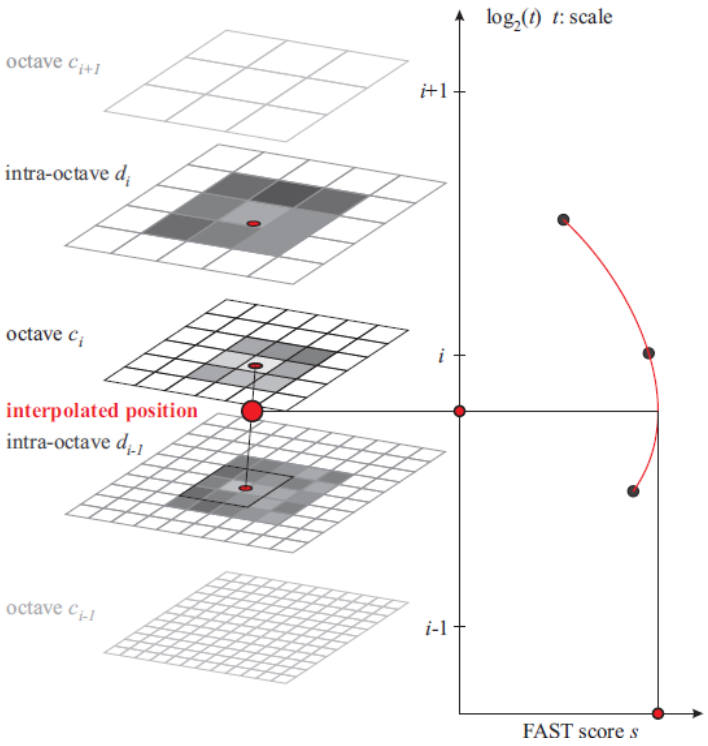


Figure 3: Scale-space pyramid layers

Figure 3 explains the Scale-space interest point detection: a keypoint (i.e. saliency maximum) is identified at octave c_i by analyzing the 8 neighboring saliency scores in ci as well as in the corresponding scores-patches in the immediately-neighboring layers above and below. In all three layers of interest, the local saliency maximum is sub-pixel refined before a 1D parabola is fitted along the scale-axis to determine the true scale of the keypoint. The location of the keypoint is then also re-interpolated between the patch maxima closest to the determined scale.

In the BRISK framework, the scale-space pyramid layers consist of n octaves c_i and n intra octaves d_i, for i = {0, 1... n−1} and typically n=4. The octaves are formed by progressively half sampling the original image (corresponding to c_0). Each intra-octave d_i is located between layers c_i and c_{i+1}.

The first intra octave d_o is obtained by down sampling the original image Co by a factor of 1.5, while the rest of intra octave layers are derived by successive half sampling. Therefore, if t denotes scale then $t(c_i) = 2^i$ and $t(d_i) = 2^i \cdot 1.5$.

In BRISK, we mostly use the 9-16 mask, which requires atleast 9 consecutive pixelsinthe16- pixel circle to either be sufficiently brighter or darker than the central pixel for the FAST criterion to be fulfilled.

Initially, the FAST 9-16 detector is applied on each octave and intra octave separately using the same threshold T to identify potential regions of interest. Next, the points belonging to these regions are subjected to non-maxima suppression in scale-space: firstly, the point in question needs to fulfill the maximum condition with respect to its 8 neighboring FAST scores s in the same layer. The score s is defined as the maximum threshold still considering an image point a corner. Secondly, the scores in the layer above and below will need to be lower as well. We check inside equally sized square patches: the side-length is chosen to be 2 pixels in the layer with the suspected maximum. Since the neighboring layers (and therefore its FAST scores) are represented with a different discretization, some interpolation is applied at the boundaries of the patch. Figure 3 depicts an example of this sampling and the maxima search.

Considering the image saliency as a continuous quantity across the image and also along the scale dimension, we perform a sub pixel and continuous scale refinement for each detected maximum. In order to limit complexity of the refinement process, we first fit a 2D quadratic function in the least squares sense to each of the three score patches (as obtained in the layer of the keypoint, the one above and the one below) resulting in the three sub-pixel refined saliency maxima. In order to avoid re-sampling, we consider a 3x3 score patch on each layer. Next, these refined scores are used to fit a 1D parabola along the scale axis yielding the final score estimate and scale estimate at its maximum. As a final step, we re-interpolate the image coordinates

between the patches in the layers next to the determined scale. The features detected are to be extracted now. The next step i.e. feature extraction is explained in next topic.

3.3 FEATURE EXTRACTION

3.3.1 Introduction:

Feature Extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy. It is a method in which the desired portions or shapes (features) of a digitalized image or video stream are detected using various algorithms. One of the most efficient algorithms is the BRISK.

3.3.2 BRISK:

Binary Robust Invariant Scalable Keypoints (BRISK) has been recently developed. Here, the interest points are detected by computing FAST score for each pixel in the image. If the pixel is above a pre-defined score threshold, then it is detected as an interest point.

The BRISK descriptor consists of a binary string by concatenating the results of simple brightness comparison tests.

In this process, we identify the characteristic direction of each keypoint to allow for orientation-normalized descriptors and hence achieve rotation invariance which is the key to general robustness. Also, we carefully select the brightness comparisons with the focus on maximizing descriptiveness.

Sampling pattern and rotation estimation:

The key concept of BRISK descriptor makes use of a pattern, used for sampling the neighborhood of the keypoint. The sampling pattern with N location, equally spaced on circles concentric with keypoints is shown below.

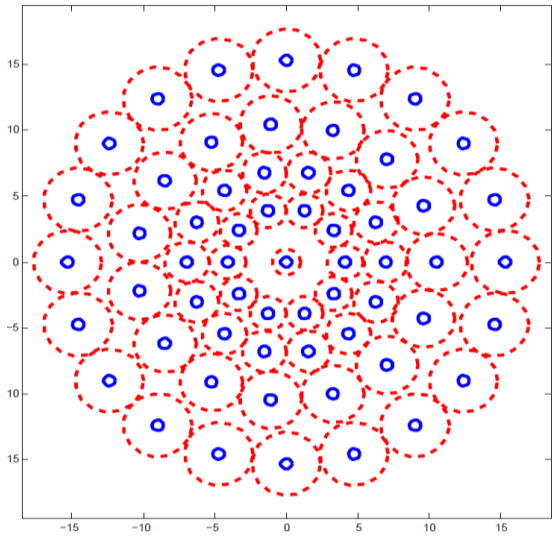


Figure 4: Sampling pattern

Figure 4 shows the BRISK sampling pattern with N=60 points. The small blue circles denote the sampling locations, the bigger red dashed circles are drawn at a radius σ corresponding to the standard deviation of the Gaussian kernel used to smooth the intensity values ay the sampling points. The pattern shown applies to a scale of t=1.

Consider a small patch centered about a keypoint, which is described as a binary string. First step is to take a sampling pattern around the keypoint. For example, the points spread on the concentric circles.

Next, choose say 512 pairs of points on this sampling pattern. Now for all the pairs of points, compare the intensity value of the first point in the pair with the intensity value of the second value in the pair. If the first value is greater than the second, write '1' in the string otherwise '0'. After going over all the 512 pairs, we have 512 characters string composed of '1' and '0' that encoded the local information around the keypoint.

In order to avoid aliasing effects when sampling the image intensity of a point $\mathbf{p_i}$ in the pattern, we apply Gaussian smoothing with standard deviation $\boldsymbol{\sigma_i}$ which is proportional to the distance between the points on the respective circle. The Gaussian smoothing operator is a 2-D

convolution operator that is used to 'blur' images and remove detail and noise. In 2-D, an isotropic (i.e. circularly symmetric) Gaussian has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

There are N points on the circular pattern. The total number of sampling point pairs will be N·(N−1)/2. For positioning and scaling the pattern accordingly for a particular key point k in the image, let us consider one of the sampling point pairs, say $(\mathbf{p_i},\mathbf{p_j})$. The smoothed intensity values at these points are $\mathbf{I}\,(\mathbf{p_i},\sigma_\mathbf{i})$ and $\mathbf{I}(\mathbf{p_j},\sigma_\mathbf{j})$ respectively. They are used to estimate the local gradient.

$$g(p_i,p_j) = (p_j - p_i) \cdot \frac{I(p_j,\sigma_j) - I(p_i,\sigma_i)}{\|p_j - p_i\|^2}$$

Considering the set A with all sampling pattern pairs,

$$\mathcal{A} = \left\{(\mathbf{p}_i,\mathbf{p}_j) \in \mathbb{R}^2 \times \mathbb{R}^2 \mid i < N \wedge j < i \wedge i,j \in \mathbb{N}\right\}$$

We define a set of short distance pairings 'S' and another subset of long distance pairings 'L'

$$S = \{(p_i,p_j) \in A \mid \|p_j - p_i\| < \delta_{max}\} \subseteq A$$

$$L = \{(p_i,p_j) \in A \mid \|p_j - p_i\| > \delta_{min}\} \subseteq A$$

The threshold distances are set to $\delta_{\mathbf{max}} = 9.75\mathbf{t}$ and $\delta_{\mathbf{min}} = 13.67\mathbf{t}$ (t is the scale of k). The long distance pairs are used for computing overall characteristic pattern direction. This is done based on the assumption that local gradients annihilate each other and are thus not necessary in the global gradient determination. Iterating the point pairs in L, we estimate the overall characteristic pattern direction of the keypoint k to be:

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{L}} \mathbf{g}(\mathbf{p}_i, \mathbf{p}_j).$$

This gradient value is used to find the angle by which the sampling pattern is rotated around the keypoint 'k'. Let it be α.

$$\boldsymbol{\alpha = \text{arctan2}\,(g_y, g_x)}$$

Building descriptor:

The rotated short distance pairings are used to build the binary descriptor 'd_k'.The bit vector 'd_k' is assembled by performing all short distance intensity comparisons of point pairs, $(p^\alpha, p^\alpha) \in \mathbb{S}$ such that each bit 'b' corresponds to:

$$b = \begin{cases} 1, & I(\mathbf{p}_j^\alpha, \sigma_j) > I(\mathbf{p}_i^\alpha, \sigma_i) \\ 0, & \text{otherwise} \end{cases}$$
$$\forall (\mathbf{p}_i^\alpha, \mathbf{p}_j^\alpha) \in \mathcal{S}$$

Firstly, BRISK uses a deterministic sampling pattern resulting in a uniform sampling-point density at a given radius around the keypoint. Furthermore, BRISK uses dramatically fewer sampling-points than pair wise comparisons (i.e. a single point participates in more comparisons), limiting the complexity of looking-up intensity values. Finally, the comparisons here are restricted spatially such that the brightness variations are only required to be locally consistent. With the sampling pattern and the distance thresholds as shown above, we obtain a bit-string of length 512. The extracted features are to be matched which is explained in next topic.

3.4 FEATURE MATCHING

3.4.1 Introduction:

The approach for matching depends partially on the application, i.e., different strategies may be preferable for matching images that are known to overlap (e.g., in image stitching) vs. images that may have no correspondence whatsoever (e.g., when trying to recognize objects from a database).

Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images. Feature matching is the comparison of two sets of feature descriptors obtained from different images to provide point correspondences between images. In the following we will describe some feature point matching algorithms that have been proposed so far. We will take a closer look at the application of each algorithm.

3.4.2 Why Match Features?

- To ignore irrelevant data
    - Backgrounds
    - Meaningless variation among instances
        - E.g. color of car, whether pickup is covered
    - Variations in illumination
- To accommodate changes in viewpoint
    - Insensitivity to classes of transformations
        - Depends on the feature extraction and matching techniques
    - Defining the class of transformation is critical to the success of any feature matching system

3.4.3 Types of Feature Matchers

- Brute Force
- FlannBased
- KnnMatch
- Hamming distance for binary features

Among the four methods, hamming distance is used in this project. The other methods are briefly explained below.

Brute force matcher:

Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one. This descriptor matcher supports masking permissible matches of descriptor sets. The following command is used in OpenCV.

*class***BFMatcher**: *public***DescriptorMatcher**

The **BFMatcher** will only return consistent pairs. Such technique usually produces best results with minimal number of outliers when there are enough matches. This brute force matcher rarely yields efficient algorithms. Some brute-force algorithms are unacceptably slow.

Flann Based Matcher:

FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset. FLANN stands for Fast Library for Approximate Nearest Neighbors. This matcher trains **flann::Index_** on a train descriptor collection and calls its nearest search methods to find the best matches. So, this matcher may be faster when matching a large collection than the brute force matcher. This matcher does not support masking permissible matches of descriptor sets. The following command is used in OpenCV.

class **FlannBasedMatcher** : *public* **DescriptorMatcher**

K-NN MATCHER:

It finds the k best matches for each descriptor from a query set. *K*-nearest neighbor algorithm (*k*-NN) matches objects based on the k closest training examples in the feature space. The function is only approximated locally and all computation is deferred until classification. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors. Basically **KNN** matcher works by finding for each SIFT or SURF descriptor (each keypoint has got a descriptor attached to it) the first **k-neighbors**. k-neighbors are found using a distance function such as Euclidean or Manhattan between 64 or 128 dimensional feature vector. So if you have n keypoints, you have n descriptors and if you want to find out k-neighbors for keypoint **x_i**, you compute distance between descriptor **x_i** and all other descriptors **x_j (j=1…..n, i! =j)**. The following command is used in OpenCV.

DescriptorMatcher: :knnMatch

Hamming Distance for binary features:

Hamming distance between two strings of equal length is the number of positions at which corresponding signals are different. The function hamming_distance() computes the Hamming distance between two strings of equal length, by creating a sequence of Boolean values indicating mismatches and matches between corresponding positions in the two inputs, and then summing the sequence with False and True values being interpreted as zero and one. For binary strings *a* and *b* the Hamming distance is equal to the number of ones (population count) in *a* XOR *b*.

Furthermore, comparing strings by computing the Hamming distance is extremely fast on modern CPUs that often provide a specific instruction to perform a XOR or bit count operation, as is the case in the latest SSE instruction set. Since establishing a match can be understood as classifying pairs of points as being a match or not, a classifier that relies on these Hamming distances will work best when their distributions are most separated. Therefore, matcher based on hamming distance is used along with BRISK descriptor in this project.

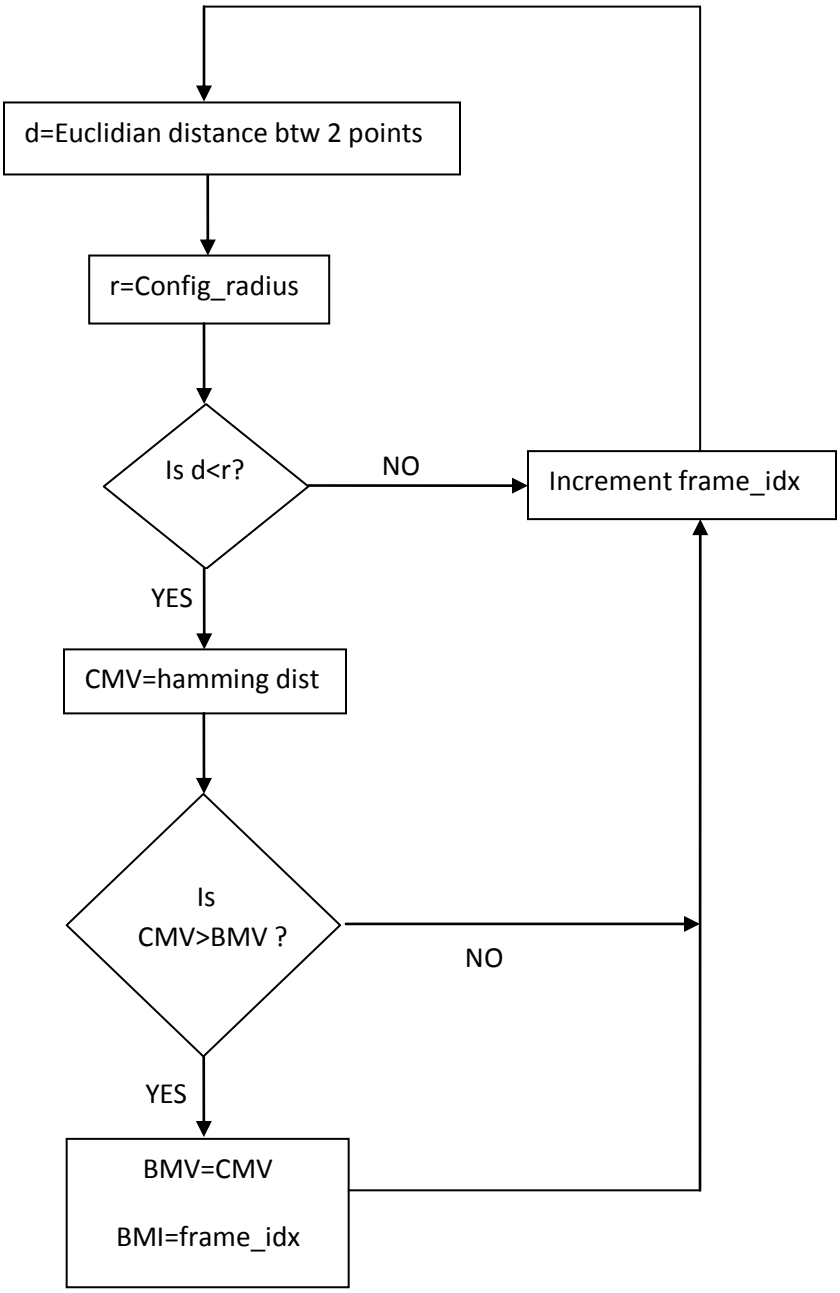
3.4.4 History and applications of Hamming distance:

The Hamming distance is named after Richard Hamming, who introduced it in his fundamental paper on Hamming codes *Error detecting and error correcting codes* in 1950. It is used in telecommunication to count the number of flipped bits in a fixed-length binary word as an estimate of error, and therefore is sometimes called the **signal distance**. The Hamming distance is an important measurement for the detection of errors in information transmission (Hamming 1950). Beyond this application its usage is valuable for the investigation of e.g. ranked variables of entities coded by numerical tokens. Hamming weight analysis of bits is used in several disciplines including information theory, coding theory, and cryptography. However, for comparing strings of different lengths or strings where not just substitutions but also insertions or deletions have to be expected, a more sophisticated metric like the Levenshtein distance is more appropriate. For q-ary strings over an alphabet of size $q \geq 2$ the Hamming distance is applied in case of orthogonal modulation, while the Lee distance is used for phase modulation. If $q = 2$ or $q = 3$ both distances coincide.

The Hamming distance is also used in systematics as a measure of genetic distance.

On a grid such as a chessboard, the Hamming distance is the minimum number of moves it would take a rook to move from one cell to the other.

3.4.5 Flowchart for matching process:



Explanation:

Given a feature in I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

In feature matching, keypoints from frame1 are matched with those in frame2.

Create a file keypoints.doc and store all the keypoints from feature extraction step into it. Bestmatchindex (BMI), currentmatchvalue (CMV) and bestmatchvalue (BMV) are initialized where bestmatchindex indicates the best matching keypoint index of frame1 in frame2, currentmatchvalue indicates hamming distance between keypoints and bestmatchvalue indicates the maximum hamming distance.

Euclidean distance (d) between keypoints of frame 1 and 2 is calculated. In mathematics, the **Euclidean distance** or **Euclidean metric** is the "ordinary" distance between two points that one would measure with a ruler, and is given by the Pythagorean formula

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This Euclidean distance is compared with configuration radius (r) to ensure that matched keypoints lie within the locality. Here configuration radius is 30 and is constant during the whole process. This method can give good scores to very ambiguous (bad) matches.

If the condition is satisfied, then hamming distance between the keypoints is found out which will be explained later. Since the aim of this step is to find out the best match, this hamming distance i.e. the current match value will be compared with best match value and based on that condition, the step will be repeated and best match of keypoint of frame1 is identified in frame2.

Now, this process will be repeated for all the frames and corresponding match indices will be identified and stored in keypoints.doc.

Hamming Distance:

To find the matching between two images, the keypoint descriptors of 1st and 2nd images are XORed and the number of similar bits is obtained. This hamming distance helps in identifying the best match for each keypoint. When initial conditions are satisfied, maximum hamming distance between two keypoints tells that they are the best matches.

Figure 5: Hamming distance example

For example, in figure 5, let us consider a keypoint in frame1 and find its best match in frame2. In the figure, the 2 keypoints in frame2 satisfy the initial conditions of Euclidean distance with keypoint in frame1. But to find the best match, we will find the hamming distance and the set of keypoints of frame1 and frame2 which has the maximum hamming distance will be considered as the best match.

A summary of feature detection, extraction and matching using BRISK is given in next topic.

3.5 SUMMARY

Effective and efficient generation of keypoints from an image is a well-studied problem in the literature and forms the basis of numerous Computer Vision applications. In this paper, we have presented a novel method named BRISK, which tackles the classic Computer Vision problem of detecting, describing and matching image keypoints for cases without sufficient a

priori knowledge on the scene and camera poses. In contrast to well-established algorithms with proven high performance, such as SIFT and SURF, the method at hand offers a dramatically faster alternative at comparable matching performance – a statement which we base on an extensive evaluation using an established framework.

The features are detected using FAST algorithm. The extracted features are matched using Hamming distance method and the best possible matches are shortlisted. BRISK relies on an easily configurable circular sampling pattern from which it computes brightness comparisons to form a binary descriptor string. The unique properties of BRISK can be useful for a wide spectrum of applications, in particular for tasks with hard real-time constraints or limited computation power: BRISK finally offers the quality of high-end features in such time-demanding applications. The results of BRISK will be combined with that of SLAM and camera pose of real world video sequence will be estimated.

The next part of the project i.e., SLAM is explained in next chapter.

4. SIMULTANEOUS LOCALIZATION AND MAPPING

4.1 INTRODUCTION

Simultaneous Location and Mapping (SLAM) is a technique used by robots and autonomous vehicles to build up a map within an unknown environment (without a priori knowledge) or to update a map within a known environment (with a priori knowledge from a given map) while at the same time keeping track of the current location. It can be used to track the camera pose while building a 3D map of the unknown scene.

This project deals with single-camera SLAM system. EKF (Extended Kalman Filter) is used to estimate the 3D trajectory of a monocular camera moving through an unknown scene. EKF allows also to achieve noise and disturbance rejection and to enhance estimation accuracy. In order to identify visual landmarks to be used in the SLAM algorithm, highly distinctive visual features, invariant to scale, rotation and linear illumination variations, are extracted from the images using the BRISK algorithm. To each feature is assigned at least one descriptor that embodies the image properties in the features' neighborhood. The descriptors are used to establish the frame-to-frame feature matches. This system combines another advanced state-of-the-art methods such as Inverse Depth Parameterization, and the 1-Point RANSAC (Random Sample Consensus) algorithm, for outlier rejection.

Algorithm steps of SLAM:

- Map management
- EKF prediction
- Search for individually compatible matches
- 1-Point RANSAC hypothesis and selection of low-innovation inliers
- Partial update using low-innovation inliers
- "Rescue" high-innovation inliers
- Partial update using high-innovation inliers

All these steps are explained in further topics.

4.2 MAP MANAGEMENT

- Delete features, if necessary
- Update features info
- Convert features from inverse depth to Cartesian, if necessary
- Initialize features randomly

4.2.1 Probabilistic 3D Map

The key concept of our approach is a probabilistic feature-based map, representing at any instant a snapshot of the current estimates of the state of the camera and all features of interest and, crucially, also the uncertainty in these estimates. The map is initialized at system start-up and persists until operation ends, but evolves continuously and dynamically as it is updated by the Extended Kalman Filter. The probabilistic state estimates of the camera and features are updated during camera motion and feature observation. When new features are observed the map is enlarged with new states and, if necessary, features can also be deleted. Mathematically, the map is represented by a state vector $\hat{x}$ and covariance matrix P.

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{\mathbf{x}}_v \\ \hat{\mathbf{y}}_1 \\ \hat{\mathbf{y}}_2 \\ \vdots \end{pmatrix}, \quad \mathrm{P} = \begin{bmatrix} \mathrm{P}_{xx} & \mathrm{P}_{xy_1} & \mathrm{P}_{xy_2} & \cdots \\ \mathrm{P}_{y_1x} & \mathrm{P}_{y_1y_1} & \mathrm{P}_{y_1y_2} & \cdots \\ \mathrm{P}_{y_2x} & \mathrm{P}_{y_2y_1} & \mathrm{P}_{y_2y_2} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}.$$

Explicitly, the camera's state vector x_v comprises a metric 3D position vector r^W, orientation quaternion q^{WR}, velocity vector v^W, and angular velocity vector ω relative to a fixed world frame W and "robot" frame R carried by the camera (13 parameters):

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^W \\ \mathbf{q}^{WR} \\ \mathbf{v}^W \\ \omega^R \end{pmatrix}$$

4.2.2 Map Management (Adding and deleting features):

An important part of the overall algorithm is sensible management of the number of features in the map, and on-the-fly decisions need to be made about when new features should be identified and initialized, as well as when it might be necessary to delete a feature. Our map-maintenance criterion aims to keep the number of reliable features visible from any camera location close to a predetermined value determined by the specifics of the measurement process, the required localization accuracy and the computing power available: we have found that with a wide-angle camera a number in the region of 12 gives accurate localization without overburdening the processor.

Feature "visibility" is calculated based on the relative position of the camera and feature and the saved position of the camera from which the feature was initialized. The feature must be predicted to lie within the image, but also the camera must not have translated too far from its initialization viewpoint of the feature or we would expect correlation to fail.

Adding features:

Features are added to the map only if the number visible in the area the camera is passing through is less than this threshold—it is undesirable to increase the number of features and add to the computational complexity of filtering without good reason. Features are detected by running the image interest operator of Shi and Tomasi to locate the best candidate within a box of limited size (around 80 X 60 pixels) placed within the image. The position of the search box is currently chosen randomly, with the constraints only that it should not overlap with any existing features and that based on the current estimates of camera velocity and angular velocity any detected features are not expected to disappear from the field of view immediately.

Deleting features:

A feature is deleted from the map if, after a predetermined number of detection and matching attempts when the feature should be visible, more than a fixed proportion (in our work, 50 percent) are failures. This criterion prunes features which are "bad" for a number of possible reasons: They are not true 3D points (lying at occlusion boundaries such as T-junctions), lie on

moving objects, are caused by specular highlights on a curved surface, or importantly are just often occluded.

Over a period of time, a "natural selection" of features takes place through these map management criteria which leads to a map of stable, static, widely-observable point features. Clutter in the scene can be dealt with even if it sometimes occludes these landmarks since attempted measurements of the occluded landmarks simply fail and do not lead to a filter update. Problems only arise if mismatches occur due to a similarity in appearance between clutter and landmarks and this can potentially lead to catastrophic failure. Note, however, that mismatches of any kind are extremely rare during periods of good tracking since the large feature templates give a high degree of uniqueness and the active search method means that matching is usually only attempted within very small image regions (typically, 15-20 pixels across).

In map management, after adding or deleting features and updating their information, the next step is to convert features from inverse depth to Cartesian, if necessary. This is explained in next topic.

4.2.3 Inverse Depth Parameterization:

For point features within monocular SLAM, a new parameterization is required which permits efficient and accurate representation of uncertainty during undelayed initialization. Feature initialization is undelayed in the sense that even distant features are immediately used to improve camera motion estimates, acting initially as bearing references but not labeled as such. The key concept is direct parameterization of inverse depth of features relative to the camera from which they are first viewed, which produces measurement equations with a high degree of linearity.

The probabilistic SLAM approach of explicit uncertainty propagation is successful in permitting repeatable 3D real-time localization and mapping even in the pure vision domain of a single camera with no extra sensing. But an issue has risen in the initialization of features, since information from multiple images acquired during motion must be combined to achieve accurate depth estimates.

To infer the depth of a feature, the camera must observe it repeatedly as it translates through the scene, each time capturing a ray of light from the feature to its optic center. The angle between the captured rays is the feature's parallax-this is what allows its depth to be estimated.

In computer vision, the well-known concept of a point at infinity is a feature which exhibits no parallax during camera motion due to its extreme depth. The homogeneous coordinate systems of visual projective geometry allow explicit representation of points at infinity, and they have proven to play an important role during off-line optimization-based structure and motion estimation from image sequences. The features at infinity must be differentiated in the beginning itself to avoid conundrum. A point at infinity is simply far enough relative to the camera motion since it has been observed that no parallax has been observed.

From the estimation point of view, we have all the features starting at infinity and 'coming in' as the camera moves far enough to measure sufficient parallax.

For nearby indoor features, only a few centimeters of movement will be sufficient. Distant features may require many meters or even kilometers of motion before parallax is observed. It is important that these features are not permanently labeled as infinite. A feature that seems to be at infinity should always have the chance to prove its finite depth given enough motion, or there will be the serious risk of systematic errors in the scene map.

With the unified representation, the algorithm requires no special initialization process for features. They are simply tracked right from the start, immediately contribute to improved camera estimates and have their correlations with all other features in the map correctly modeled. The unified inverse depth parameterization is able to code in the map distant points, in which the inverse depth coding never collapses and cannot be coded with standard Euclidean representation.

Camera motion model:

State vector definition:

We assume a constant angular velocity and linear model to define the movement of the camera. So the camera state Xv is composed the following locations:

r^{WC} → camera optical center

q^{WC} → quaternion defining orientation

v^{W} → velocity

w^{W} → angular velocity

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^{WC} \\ \mathbf{q}^{WC} \\ \mathbf{v}^{W} \\ \omega^{W} \end{pmatrix}.$$

It is assumed that there are a linear and angular accelerations a and α. These affect the camera, producing at each step, an impulse of linear velocity V = a Δt and angular velocity Ω = αΔt, with zero mean and known Gaussian distribution. A diagonal covariance matrix for the unknown input linear and angular accelerations are defined. The state update equation for the camera is:

$$\mathrm{f}_v = \begin{pmatrix} \mathrm{r}^{WC}_{k+1} \\ \mathrm{q}^{WC}_{k+1} \\ \mathrm{v}^{W}_{k+1} \\ \omega^{W}_{k+1} \end{pmatrix} = \begin{pmatrix} \mathrm{r}^{WC}_{k} + \left(\mathrm{v}^{W}_{k} + \mathrm{V}^{W}_{k}\right)\Delta t \\ \mathrm{q}^{WC}_{k} \times \mathrm{q}\left(\left(\omega^{W}_{k} + \Omega^{W}\right)\Delta t\right) \\ \mathrm{v}^{W}_{k} + \mathrm{V}^{W} \\ \omega^{W}_{k} + \Omega^{W} \end{pmatrix}$$

The quaternion defined by the rotation vector $(\omega^{W}_{k} + \Omega^{W})\,\Delta t$ is $\mathrm{q}\,(\omega^{W}_{k} + \Omega^{W})\,\Delta t$)

Euclidean XYZ point parameterization:

The representation of the position of features in Euclidean coordinates is as follows:

$$\mathrm{x} = (\mathrm{x}_v^{\mathrm{T}}, \mathrm{y}_1^{\mathrm{T}}, \mathrm{y}_2^{\mathrm{T}}, \ldots\ldots \mathrm{y}_n^{\mathrm{T}})^{\mathrm{T}}$$

Inverse depth point parameterization:

A scene 3D point 'I' is defined by the 6 dimension state vector

$$\mathbf{y_i} = (\, x_i \;\; y_i \;\; z_i \;\; \theta_i \;\; \emptyset_i \;\; \rho_i \,)^{\mathrm{T}}$$

Here, the state codes the ray for the first point observation as x_i, y_i, z_i, the camera optical center where the 3D point was first observed and θ_i, $\emptyset_i$ azimuth and elevation (coded in the absolute reference) for the ray directional vector $\mathbf{m}\,(\theta_i, \emptyset_i)$. The point depth along the ray d_i, is coded by its inverse $\rho_i = 1 / d_i$

This expression explains how to transform a 6-D point in 3-D:

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \emptyset_i)$$

$$\mathrm{m} = (\cos\emptyset_i \ \sin\theta_i, -\sin\emptyset_i, \cos\emptyset_i \cos\theta_i)^{\mathrm{T}}$$

Full state vector:

As in other visual SLAM approaches working in a EKF framework, we use a only a state vector. Therefore the full state vector is composed of Xv and the 6D codification estimated features.

$$\mathrm{X} = (\mathrm{x}_v^{\mathrm{T}}, \mathrm{y}_1^{\mathrm{T}}, \mathrm{y}_2^{\mathrm{T}}, \ldots\ldots \mathrm{y}_n^{\mathrm{T}})^{\mathrm{T}}$$

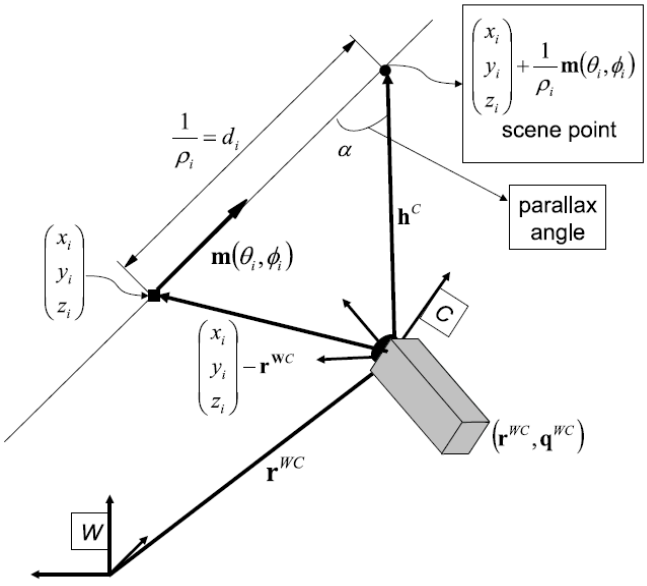


Figure 6 illustrates the ray coded in IDP (Inverse Depth Parameterization)

Measurement model:

The measurements are related to the state by camera parameters and perspective projection. The measurement model predicts hi values based on:

• The absolute position of each landmark

• The absolute position of the camera

• The rotation matrix of the camera

For points in the Euclidean representation,

$$\mathbf{h}^C = \mathbf{R}^{CW}\left(\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}\left(\theta_i, \phi_i\right) - \mathbf{r}^{WC}\right)$$

For inverse depth parameterization,

$$\mathbf{h}^C = \mathbf{h}_\rho^C = \mathbf{R}^{CW}\left(\rho_i\left(\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \mathbf{r}^{WC}\right) + \mathbf{m}\left(\theta_i, \phi_i\right)\right)$$

In the next expression we transform the measurement to pixel units by the camera parameters

$$\mathbf{h} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 - \frac{f}{d_x}\upsilon \\ v_0 - \frac{f}{d_y}\nu \end{pmatrix}$$

Where, u_0 , v_0 are the camera center in pixels, f is the focal length and d_x and d_y the pixel size.

Finally, a radial distortion model has to be applied in order to deal with real camera lenses. In this work we have used the standard photogrammetry two parameters distortion model. It is worth noting, that the measurement equation has a sensitive dependency on the parallax angle α. In the calibrated camera context, the parallax is the angle defined by the two rays defined by the same scene point when observed from two different viewpoints. At low parallax, both rays are almost parallel and:

$$\rho_i\left(\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \mathbf{r}^{WC}\right) + \mathbf{m}\left(\theta_i, \phi_i\right) \approx \mathbf{m}\left(\theta_i, \phi_i\right)$$

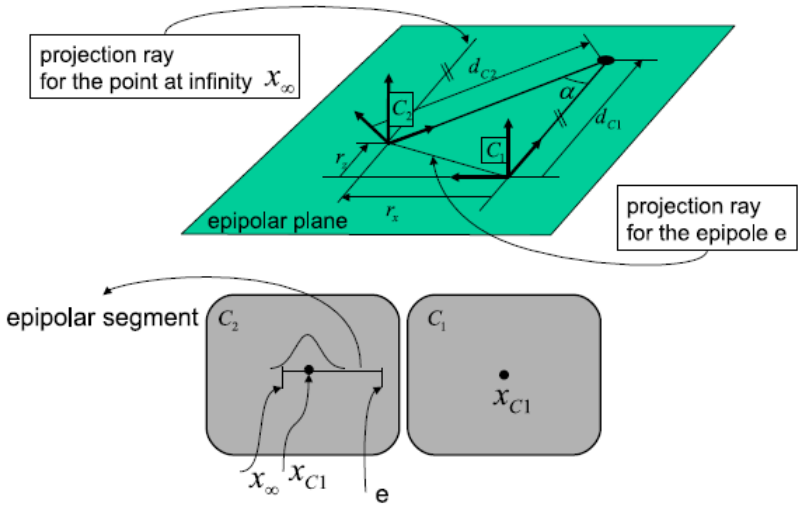


Figure 7: Observation of a point by two cameras

Figure 7 shows the observation of a point by two cameras. The geometry has been defined with respect to the epipolar plane. Bottom subfigure shows the same geometry as observed by the cameras.

The measurement equation only provides information about the camera orientation and about the directional vector $\mathbf{m}(\theta_i, \emptyset_i)$. This particular case has been exploited to build a visual compass based on SLAM.

Measurement equation linearity:

We are using the EKF to estimate the state. The more linear the measurement equation, the better performance is expected from the Kalman filter. Next, we show how at low parallax angles, coded in d, improves the linearization .Because of that we parameterize on the inverse depth.

We focus on the observation of a point from two camera locationsC1 (absolute frame) and C2. The references are aligned with respect to the epipolar plane (defined by the scene point and the two cameras optical centers, to simplify the measurement equation. The Z axis is aligned with the ray defined by the optical center and the observed point. The Y axis is normal to the epipolar plane. Given a point imaged in C1 as xC1 its image on C2, xC2 is constrained to be (if in front of the cameras) on the epipolar segment defined by the epipole (the image of C1 on C2)

and x1 (the image on xC2 if the scene point where at infinity). Hence the measurement equation is defined by:

$$\begin{aligned} \mathbf{y} &= \left(0,0,0,0,0,\frac{1}{d_{c_1}}\right)^T \\ \mathbf{R}_{C_2C_1} &= \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix} \\ \mathbf{r}^{CW} &= (r_x, 0, r_z)\,. \end{aligned}$$

We propose to compare the two parameterizations in terms of their linearity, first we focus on v (p) then the analysis is extended to v (d) and finally a comparison is made.

A first order approximation for the first derivative in the interval $[\rho_0 - 2\sigma_\rho, \rho_0 - 2\sigma_\rho]$ is given by the first order Taylor expansion around ρo.

$$\frac{\partial v}{\partial \rho}(\rho_0 + \Delta\rho) \approx \left.\frac{\partial v}{\partial \rho}\right|_{\rho_0} + \left.\frac{\partial^2 v}{\partial \rho^2}\right|_{\rho_0} \Delta\rho.$$

The dimensionless ratio between the derivative increments at the interval extreme $\left.\frac{\partial v}{\partial \rho}\right|_{\rho_0}$

And the derivative in the linearization point $\left.\frac{\partial^2 v}{\partial \rho^2}\right|_{\rho_0} 2\sigma_\rho$ as a linearity measurement. So,

$$\frac{\frac{\partial^2 v}{\partial \rho^2} 2\sigma_\rho}{\frac{\partial v}{\partial \rho}} \approx 0$$

By computing the dimensionless ratio for the ρ parameterization:

$$\frac{2\sigma_\rho}{\rho_0} 2\left(1 - \frac{d_{C_1}}{d_{C_2}}\cos\alpha\right) \approx 0$$

That is, at low parallax and $\frac{d_{C_1}}{d_{C_2}} \approx 1$, when the term $\left(1 - \frac{d_{C_1}}{d_{C_2}} \cos\alpha\right) \approx 0$

The low linearization error can be achieved even if $\frac{2\sigma_\rho}{\rho_0} \gg 0$.

So huge initial uncertainty regions can be coded Gaussianly. After converting features from inverse depth to cartesian, initialize the features as explained in next topic.

4.2.4 Feature initialization:

The new features are initialized using only one image, the image where the feature is first observed; the initialization includes both the feature state initial values and the covariance assignment. Despite the initial uncertainty region covers a huge range depth till infinity because of the low linearization errors the uncertainty is successfully coded as Gaussian; once initialized, the feature is processed with the standard EKF prediction-update loop.

At low parallax, the feature will be used mainly to determine the camera orientation but the feature depth will be kept quite uncertain, including in its uncertainty region the even infinity; if the camera translation is able to produce a parallax big enough then the feature depth estimation will be improved.

The initial location for the observed feature is defined as:

$$\hat{\mathbf{y}}\left(\hat{\mathbf{r}}^{WC}, \hat{\mathbf{q}}^{WC}, \mathbf{h}, \rho_0\right) = \begin{pmatrix} \hat{x}_i & \hat{y}_i & \hat{z}_i & \hat{\theta}_i & \hat{\phi}_i & \hat{\rho}_i \end{pmatrix}$$

From the camera location estimate at step k (the k indexes have been dropped for simplicity), and the observation of a new feature: h = $(u\ v)^T$ and initial ρ_0.

The projection ray initial point is directly taken from the current camera location estimate:

$$\begin{pmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{z}_i \end{pmatrix} = \hat{\mathbf{r}}^{WC}_{k|k}$$

The projection ray directional vector is computed from the observed point, expressed in the absolute frame:

$$\mathbf{h}^W = \mathbf{R}_{WC}\left(\mathbf{q}_{k|k}^{WC}\right)\mathbf{h}^C \begin{pmatrix} \upsilon \\ \nu \\ 1 \end{pmatrix}$$

The angles can be derived as:

$$\begin{pmatrix} \theta_i \\ \phi_i \end{pmatrix} = \begin{pmatrix} \arctan\left(-\mathbf{h}_y^W, \sqrt{\mathbf{h}_x^{W\,2} + \mathbf{h}_z^{W\,2}}\right) \\ \arctan\left(\mathbf{h}_x^W, \mathbf{h}_z^W\right) \end{pmatrix}$$

The initial value for ρo is derived heuristically to cover inits 95% acceptance region a working space from infinity to a predefined close distance, dmin expressed as inverse depth: [1/dmin, 0]

$$\hat{\rho}_0 = \frac{\rho_{\min}}{2} \quad \sigma_\rho = \frac{\rho_{\min}}{4} \quad \rho_{\min} = \frac{1}{d_{\min}}.$$

The state covariance after feature initialization is:

$$\hat{\mathbf{P}}_{k|k}^{\text{new}} = \mathbf{J}\begin{pmatrix} \hat{\mathbf{P}}_{k|k} & 0 & 0 \\ 0 & \mathbf{R}_j & 0 \\ 0 & 0 & \sigma_\rho^2 \end{pmatrix}\mathbf{J}^\top$$

$$\mathbf{J} = \left(\begin{array}{c|c} I & 0 \\ \hline \frac{\partial \mathbf{y}}{\partial \mathbf{r}^{WC}}, \frac{\partial \mathbf{y}}{\partial \mathbf{q}^{WC}}, 0, \ldots, 0, & \frac{\partial \mathbf{y}}{\partial \mathbf{h}}, \frac{\partial \mathbf{y}}{\partial \rho} \end{array}\right)$$

Example:

This figure shows the image where the analyzed features are initialized and the last image in the sequence; the top view of the map with the feature covariance is plotted as well. To display a map that contains features at very different depths, two top views at different scales are plotted. The top view plotted at bottom left subfigure displays the close features; the top view plotted at the bottom right subfigure displays the distant features. Both top views compare our inverse depth Gaussian parameterization with the standard XYX Gaussian parameterization by the comparison of their uncertainty regions.

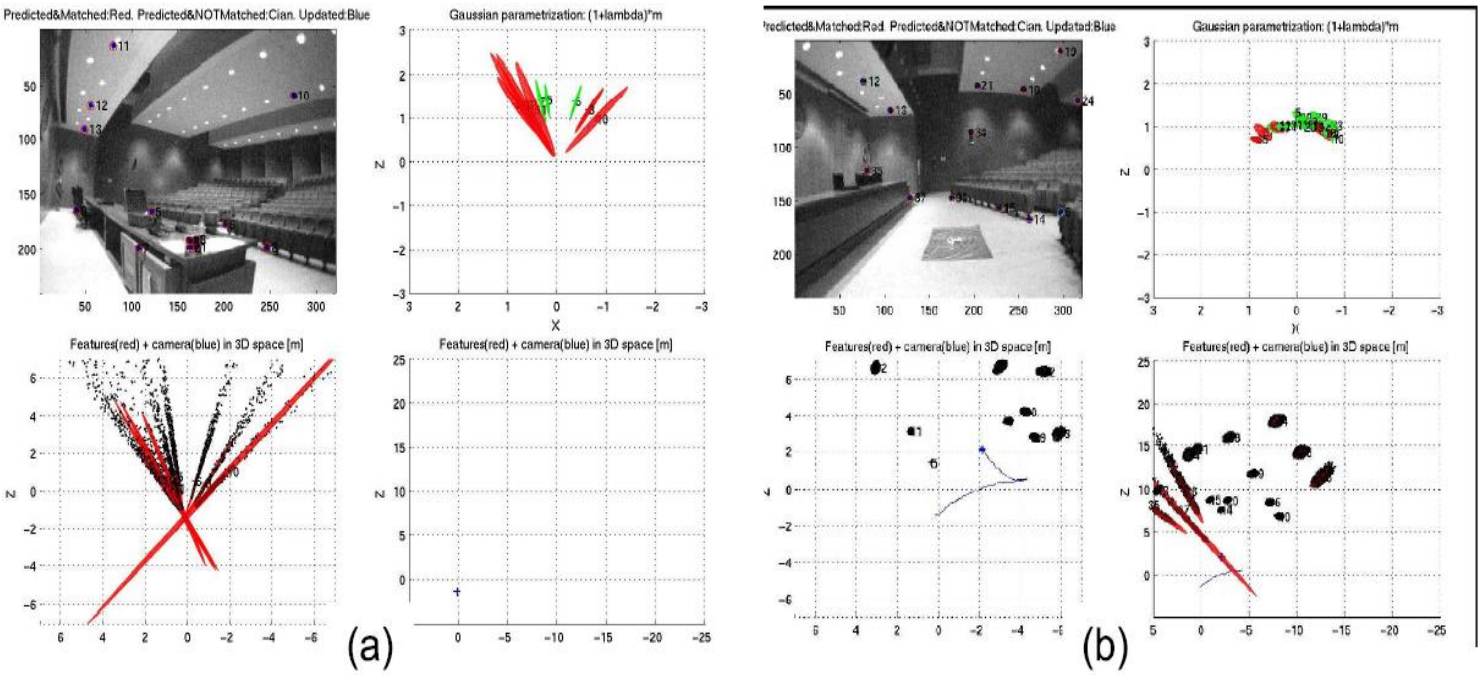


Figure 8: Example- A sample video sequence

- Figure (a) and (b) are the images of a video sequence.
- To display a map that contains features at very different depths, two top views at different scales are plotted.
- The top view plotted at bottom left subfigure displays the close features; the top view plotted at the bottom right subfigure displays the distant features.
- Both top views compare our inverse depth Gaussian parameterization with the standard XYX Gaussian parameterization by the comparison of their uncertainty regions.
- The Gaussian inverse depth acceptance regions are plotted in XYZ as a cloud of black dots numerically propagated from the Gaussian6 dimensional super ellipsoidal acceptance region coded in inverse depth.
- The standard Gaussian XYZ acceptance ellipsoids are linearly propagated from the6 dimensional Gaussian coded in inverse depth by means of the Jacobian.
- The camera trajectory and its uncertainty is shown in blue.
- At the initial step (a), most the features are at low parallax.
- At the final step (b), parallax enough has been gathered for the majority of the features and the feature uncertainty is low.

As the camera moves, the translation produces parallax; the features depth estimate improves, so in the last image, most of the map features have reduced their uncertainty. As a result the both the uncertainty in XYZ and in inverse depth are Gaussian and the black and the red uncertainty regions become coincident.

Example of feature initialization:

This is the evolution of the estimate corresponding to features 11, 12 and 13 at frames 1, 10, 25, 50, 100 and 200 counted since feature initialization.

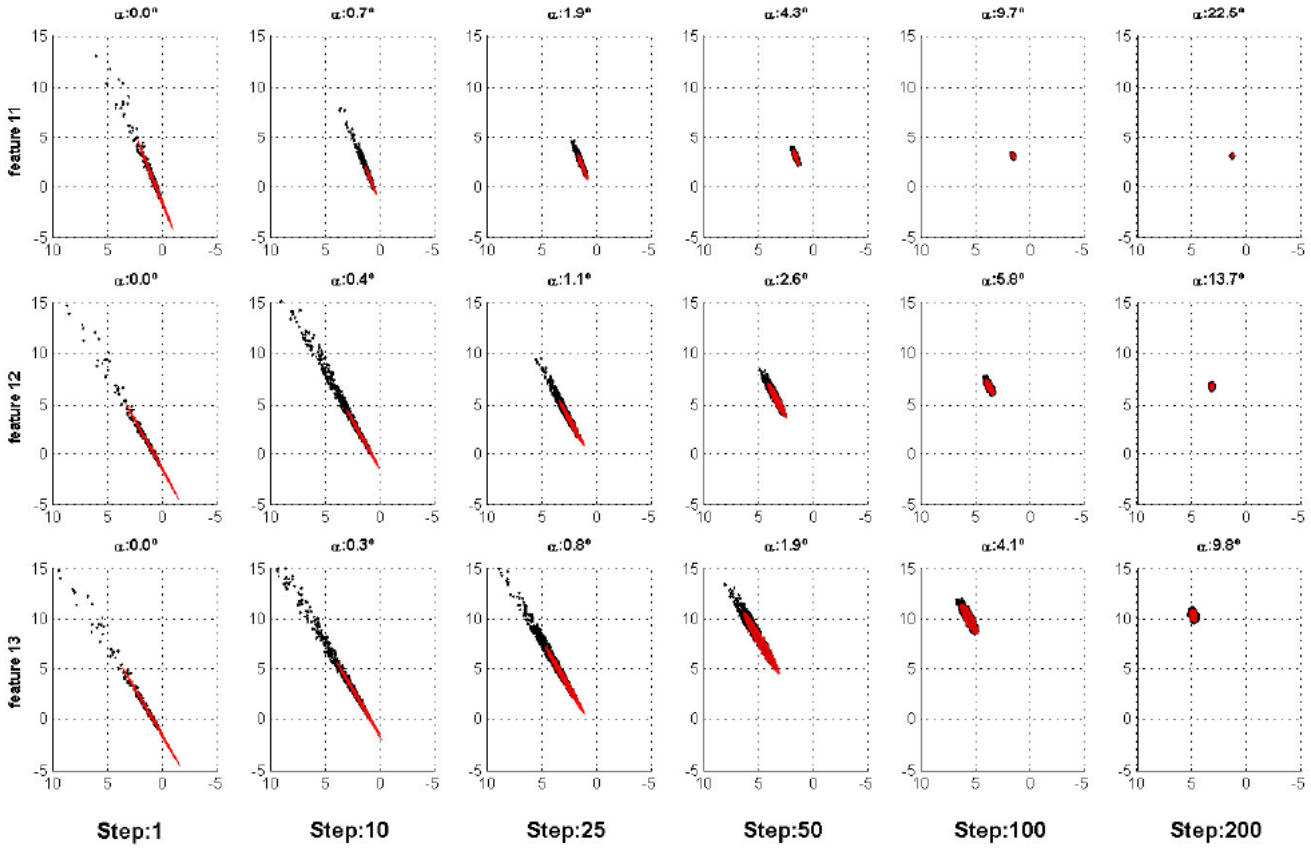


Figure 9: Feature Initialization

Here, every row shows the evolution of feature estimation in top view. Per each feature, the estimation after 1, 10, 25, 50, 100 and 200 frames since initialization are plotted; the parallax between the initial observation and the current frame is detailed on top of every subplot. Black dots are a numerical representation for the 95% uncertainty region Gaussian in the inverse depth. The red ellipsoid is the uncertainty region coded as Gaussian in XYZ.

In the unified inverse depth parameterization process, operation on standard EKF prediction-update procedure at every step is done along with the tracking of known features. For any point in the scene, close or distant, or even at infinity, inverse depth parameterization can be done. This map management is followed by EKF prediction, which is explained in next topic.

4.3 EXTENDED KALMAN FILTER

4.3.1 Introduction:

The Extended Kalman Filter is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance.

The Kalman filter is an algorithm that uses a series of measurements observed over time, containing noise and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone.

The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. Because of the algorithm's recursive nature, it can run in real time using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

Extensions and generalizations to the method have also been developed, such as the extended Kalman filter and the unscented Kalman filter which work on nonlinear systems. The underlying model is a Bayesian model similar to a hidden Markov model but where the state space of the latent variables is continuous and where all latent and observed variables have Gaussian distributions.

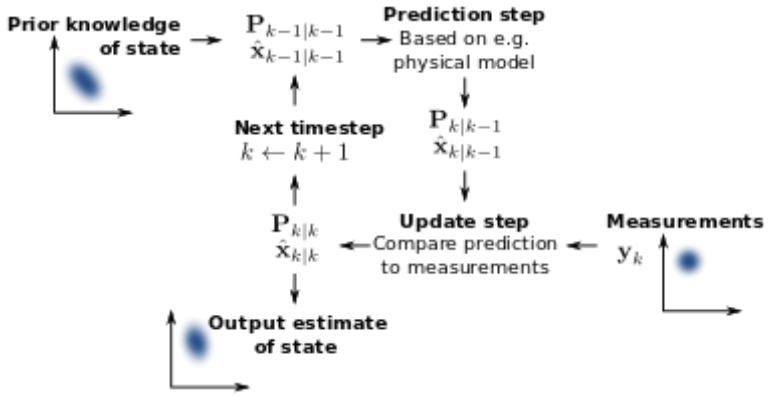


Figure 10

Figure 10 illustrates that the Kalman filter keeps track of the estimated state of the system and the variance or 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.

4.3.2 Historical Development:

The filter is named after Hungarian émigré Rudolf E. Kálmán, although Thorvald Nicolai Thiele and Peter Swerling developed a similar algorithm earlier. Richard S. Bucy of the University of Southern California contributed to the theory, leading to it often being called the Kalman–Bucy filter. Stanley F. Schmidt is generally credited with developing the first implementation of a Kalman filter. It was during a visit by Kalman to the NASA Ames Research Center that he saw the applicability of his ideas to the problem of trajectory estimation for the Apollo program, leading to its incorporation in the Apollo navigation computer. This Kalman filter was first described and partially developed in technical papers by Swerling (1958), Kalman (1960) and Kalman and Bucy (1961).

Kalman filters have been vital in the implementation of the navigation systems of U.S. Navy nuclear ballistic missile submarines, and in the guidance and navigation systems of cruise missiles such as the U.S. Navy's Tomahawk missile and the U.S. Air Force's Air Launched Cruise Missile. It is also used in the guidance and navigation systems of the NASA Space Shuttle and the attitude control and navigation systems of the International Space Station.

This digital filter is sometimes called the *Stratonovich–Kalman–Bucy filter* because it is a special case of a more general, non-linear filter developed somewhat earlier by the Soviet mathematician Ruslan L. Stratonovich. In fact, some of the special case linear filter's equations appeared in these papers by Stratonovich that were published before summer 1960, when Kalman met with Stratonovich during a conference in Moscow.

The papers establishing the mathematical foundations of Kalman type filters were published between 1959 and 1961. The primary drawback of the Kalman Filter is that it is the optimal estimate for linear system models with additive independent white noise in both the transition and the measurement systems. Unfortunately in engineering most systems are nonlinear, so some attempt was immediately made to apply this filtering method to nonlinear systems. Most of this work was done at NASA Ames. The EKF which adapted techniques, namely multivariate Taylor Series expansions, from calculus to linearize about a working point became the working solution. If the system model (as described below) is not well known or is inaccurate, then Monte Carlo methods, especially particle filters are employed for estimation. Monte Carlo techniques predate the existence of the EKF but are more computationally expensive for any moderately dimensioned state-space.

4.3.3 Applications:

- Tracking and Vertex Fitting of charged particles in Particle Detectors
- Tracking of objects in computer vision
- Dynamic positioning
- Satellite navigation systems
- Simultaneous localization and mapping
- 3D modeling

4.3.4 Overview of the Calculation:

The Kalman filter averages a prediction of a system's state with a new measurement using a weighted average. The purpose of the weights is that values with better (i.e., smaller) estimated uncertainty are trusted more. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state. The result of the weighted average is a new state estimate that lies between the predicted and measured state, and has a better estimated uncertainty than either alone. This process is repeated every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that the Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state.

The Kalman gain is a function of the relative certainty of the measurements and current state estimate, and can be "tuned" to achieve particular performance. With a high gain, the filter places more weight on the measurements, and thus follow them more closely. With a low gain, the filter follows the model predictions more closely, smoothing out noise but decreasing the responsiveness. At the extremes, a gain of one causes the filter to ignore the state estimate entirely, while a gain of zero causes the measurements to be ignored.

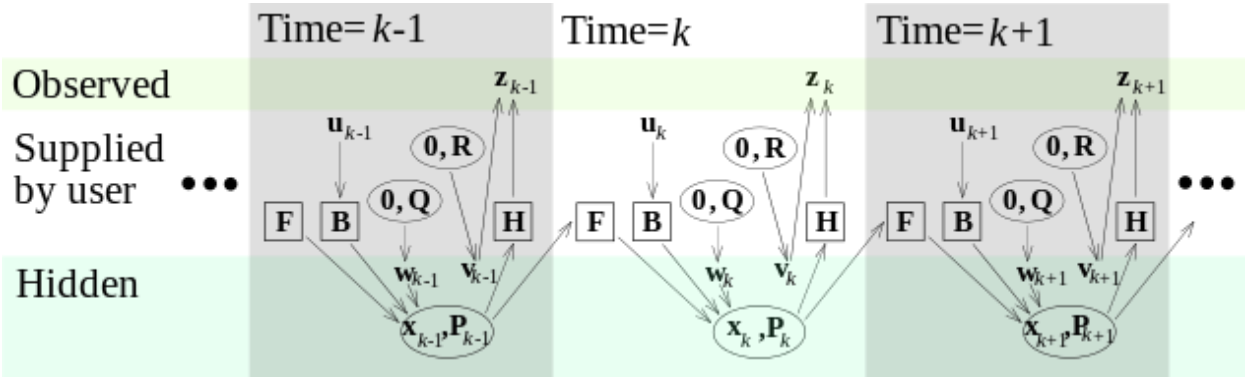


Figure 11: Underlying dynamic system model

Figure 11 gives the Model underlying the Kalman filter. Squares represent matrices. Ellipses represent multivariate normal distributions (with the mean and covariance matrix enclosed). Unenclosed values are vectors. In the simple case, the various matrices are constant

with time, and thus the subscripts are dropped, but the Kalman filter allows any of them to change each time step.

Here,

F_k → State-transition model

H_k →Observation model

Q_k →Covariance of the process noise

R_k →the covariance of the observation noise

B_k →the control-input model, for each time-step, k.

The Kalman filter model assumes the true state at time k is evolved from the state at (k−1) according to

$$\mathbf{x}_k = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k$$

where

- $\mathbf{F}_k$ is the state transition model which is applied to the previous state $\mathbf{x}_{k-1}$;
- $\mathbf{B}_k$ is the control-input model which is applied to the control vector $\mathbf{u}_k$;
- $\mathbf{w}_k$ is the process noise which is assumed to be drawn from a zero mean multivariate and normal distribution with covariance $\mathbf{Q}_k$.

$$\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$$

At time k an observation (or measurement) $\mathbf{z}_k$ of the true state $\mathbf{x}_k$ is made according to

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$$

where $\mathbf{H}_k$ is the observation model which maps the true state space into the observed space and $\mathbf{v}_k$ is the observation noise which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$.

$$\mathbf{v}_k \sim N(0, \mathbf{R}_k)$$

The initial state, and the noise vectors at each step $\{\mathbf{x}_0, \mathbf{w}_1, ..., \mathbf{w}_k, \mathbf{v}_1 ... \mathbf{v}_k\}$ are all assumed to be mutual independent.

4.3.5 Details:

The Kalman filter is a recursive estimator. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state.

The state of the filter is represented by two variables:

- $\hat{x}_{k|k}$ the *a posteriori* state estimate at time *k* given observations up to and including at time *k*;
- $P_{k|k}$, the *a posteriori* error covariance matrix.

The Kalman filter can be written as a single equation; however it is most often conceptualized as two distinct phases: "Predict" and "Update".

- The predict phase uses the state estimate from the previous time step to produce an estimate of the state at the current time step. This predicted state estimate is also known as the *a priori* state estimate because, although it is an estimate of the state at the current time step, it does not include observation information from the current time step.

In the update phase, the current *a priori* prediction is combined with current observation information to refine the state estimate. This improved estimate is termed the *a posteriori* state estimate.

4.3.6 Formulation:

In the extended Kalman filter, the state transition and observation models need not be linear functions of the state but may instead be differentiable functions.

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}) + \boldsymbol{w}_{k-1}$$
$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

Where $\mathbf{w}_k$ and $\mathbf{v}_k$ are the process and observation noises which are both assumed to be zero mean multivariate Gaussian noises with covariance $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.

The function *f* can be used to compute the predicted state from the previous estimate and similarly the function *h* can be used to compute the predicted measurement from the predicted state. However, *f* and *h* cannot be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed.

At each time step, the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear function around the current estimate.

4.3.7 Discrete-time predict and update equations:

Predict

Predicted state estimate $\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1})$

Predicted covariance estimate $\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^{\top} + \boldsymbol{Q}_{k-1}$

Update

Innovation or measurement residual $\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1})$

Innovation (or residual) covariance $\boldsymbol{S}_k = \boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^{\top} + \boldsymbol{R}_k$

Near-optimal Kalman gain $\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^{\top}\boldsymbol{S}_k^{-1}$

Updated state estimate $\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k$

Updated covariance estimate $\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1}$

where the state transition and observation matrices are defined to be the following Jacobians

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}}$$

Invariants

If the model is accurate, and the values for $\hat{\mathbf{x}}_{0|0}$ and $\mathbf{P}_{0|0}$ accurately reflect the distribution of the initial state values, then the following invariants are preserved: (all estimates have a mean error of zero)

- $\mathrm{E}[\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}] = \mathrm{E}[\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1}] = 0$
- $\mathrm{E}[\tilde{\mathbf{y}}_k] = 0$

where E[ξ] is the expected value of ξ, and covariance matrices accurately reflect the covariance of estimates

- $\mathbf{P}_{k|k} = \mathrm{cov}(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})$
- $\mathbf{P}_{k|k-1} = \mathrm{cov}(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1})$
- $\mathbf{S}_k = \mathrm{cov}(\tilde{\mathbf{y}}_k)$

4.3.8 Estimation of the noise covariances $\mathbf{Q}_k$ and $\mathbf{R}_k$

Practical implementation of the Kalman Filter is often difficult due to the inability in getting a good estimate of the noise covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$. Extensive research has been done in this field to estimate these covariances from data. One of the more promising approaches to do this is the **Autocovariance Least-Squares (ALS)** technique that uses auto covariances of routine operating data to estimate the covariances. The GNU Octave code used to calculate the noise covariance matrices using the **ALS**technique is available online under the GNU General Public License license.

4.3.9 Deriving the *a posteriori* estimate covariance matrix

Starting with our invariant on the error covariance $\mathbf{P}_{k\,|\,k}$ as above

$$\mathbf{P}_{k|k} = \mathrm{cov}(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})$$

substitute in the definition of $\hat{\mathbf{x}}_{k|k}$

$$\mathbf{P}_{k|k} = \mathrm{cov}(\mathbf{x}_k - (\hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k))$$

and substitute $\tilde{\mathbf{y}}_k$

$$\mathbf{P}_{k|k} = \operatorname{cov}(\mathbf{x}_k - (\hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1})))$$

and $\mathbf{z}_k$

$$\mathbf{P}_{k|k} = \operatorname{cov}(\mathbf{x}_k - (\hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1})))$$

and by collecting the error vectors we get

$$\mathbf{P}_{k|k} = \operatorname{cov}((I - \mathbf{K}_k\mathbf{H}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1}) - \mathbf{K}_k\mathbf{v}_k)$$

Since the measurement error $\mathbf{v}_k$ is uncorrelated with the other terms, this becomes

$$\mathbf{P}_{k|k} = \operatorname{cov}((I - \mathbf{K}_k\mathbf{H}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1})) + \operatorname{cov}(\mathbf{K}_k\mathbf{v}_k)$$

by the properties of vector covariance this becomes

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\operatorname{cov}(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1})(I - \mathbf{K}_k\mathbf{H}_k)^{\mathrm{T}} + \mathbf{K}_k\operatorname{cov}(\mathbf{v}_k)\mathbf{K}_k^{\mathrm{T}}$$

which, using our invariant on $\mathbf{P}_{k\,|\,k-1}$ and the definition of $\mathbf{R}_k$ becomes

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}(I - \mathbf{K}_k\mathbf{H}_k)^{\mathrm{T}} + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^{\mathrm{T}}$$

This formula (sometimes known as the "**Joseph form**" of the covariance update equation) is valid for any value of $\mathbf{K}_k$. It turns out that if $\mathbf{K}_k$ is the optimal Kalman gain, this can be simplified further as shown below.

4.3.10 Kalman gain derivation

The Kalman filter is a minimum mean-square error estimator. The error in the *a posteriori* state estimation is

$$\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}$$

We seek to minimize the expected value of the square of the magnitude of this vector, $\mathbf{E[\,||\,x_k - \hat{x}_{k|k}\,||^2]}$. This is equivalent to minimizing the trace of the *a posteriori* estimate covariance matrix $P_{k|k}$. By expanding out the terms in the equation above and collecting, we get:

$$\begin{aligned}\mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{K}_k^{\mathrm{T}} + \mathbf{K}_k(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k)\mathbf{K}_k^{\mathrm{T}} \\ &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{K}_k^{\mathrm{T}} + \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^{\mathrm{T}}\end{aligned}$$

The trace is minimized when its matrix derivative with respect to the gain matrix is zero. Using the gradient matrix rules and the symmetry of the matrices involved we find that

$$\frac{\partial\,\mathrm{tr}(\mathbf{P}_{k|k})}{\partial\,\mathbf{K}_k} = -2(\mathbf{H}_k\mathbf{P}_{k|k-1})^{\mathrm{T}} + 2\mathbf{K}_k\mathbf{S}_k = 0.$$

Solving this for $\mathbf{K}_k$ yields the Kalman gain:

$$\begin{aligned}\mathbf{K}_k\mathbf{S}_k &= (\mathbf{H}_k\mathbf{P}_{k|k-1})^{\mathrm{T}} = \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1}\end{aligned}$$

This gain, which is known as the *optimal Kalman gain*, is the one that yields MMSE estimates when used.

4.3.11 Simplification of the *a posteriori* error covariance formula:

The formula used to calculate the *a posteriori* error covariance can be simplified when the Kalman gain equals the optimal value derived above. Multiplying both sides of our Kalman gain formula on the right by $\mathbf{S}_k\mathbf{K}_k^{T}$, it follows that

$$\mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{K}_k^T$$

Referring back to our expanded formula for the *a posteriori* error covariance,

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{K}_k^T + \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^T$$

we find the last two terms cancel out, giving

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}.$$

This formula is computationally cheaper and thus nearly always used in practice, but is only correct for the optimal gain. If arithmetic precision is unusually low causing problems with numerical stability, or if a non-optimal Kalman gain is deliberately used, this simplification cannot be applied; the *a posteriori* error covariance formula as derived above must be used. To determine rotation of a feature, quaternion is used. It is explained in next topic.

4.4 QUATERNION

4.4.1 Introduction:

A quaternion is a tool which is as powerful as vector notation but which follows the representation of operations not directly representable with vectors, such as rotations. In mathematics, the quaternions are a number system that extends the complex numbers. In general, a quaternion is defined as the quotient of two directed lines in a three-dimensional space or equivalently as the quotient of two vectors. Quaternions find uses in both theoretical and applied mathematics, in particular for calculations involving three-dimensional rotations such as in three-dimensional computer graphics and computer vision.

The basic equation of quaternion is $i^2 = j^2 = k^2$ = ijk = -1. This formula is the rule for multiplying two quaternions.

In quaternions, all properties of real and complex numbers are preserved except for commutativity of multiplication. Quaternions can be used to represent many operations in three-dimensional space, including rotations, affine transformations, and projections.

The standard quadrinomial form is Q = {α+βi+γj+δk ; α,β,γ,δ real}

4.4.2 Multiplication of basis elements:

i. Closure: If P,Q € Q, then PQ € Q
ii. Associativity: (PQ)R = P(QR) for all P,Q,R € Q
iii. Identity: There is a 1€Q such that IP = PI = P
iv. Inverse: If P!=0, then there is a P^1 such that P^1P=1

Vectors as quaternions:

The symbols i, j, k are commonly used in vector analysis to represent elements of an orthonormal basis suggests that quaternions of the form βi + γj + δk might be interpreted as vectors.

U = u_x i + u_y j + u_z k

V = v_x i + v_y j + v_z k

When u and v are multiplied, the product is

UV $= (-u_x v_x - u_y v_y - u_z v_z) + (u_y v_z - u_z v_y)\,i + (u_z v_x - u_x v_z)\,j + (u_x v_y - u_y v_x)\,k$

$= -(u.v) + (u \mathbf{x} v)$

Where u.v and u **x** v are the dot product and cross products respectively.

4.4.3 Applications of quaternion in computer vision:

Most important computer vision functions can be represented simply using quaternions. The general rotations and affine transformations develops expressions for expressions that are used exclusively in computer vision.

Let Plane (v) be the image plane, the plane passing through the origin with surface normal v. The (parallel or orthogonal) projection of a point p onto Plane (v) is

$$Pr(p) = p+vpv/2$$

Similarly the (central or perspective) projection of point p as shown in figure 13 is

$$PR(p) = -(p+vpv)/(vp+pv)$$

$$= v \mathbf{x} (p \mathbf{x} v) / v \mathbf{.} p$$

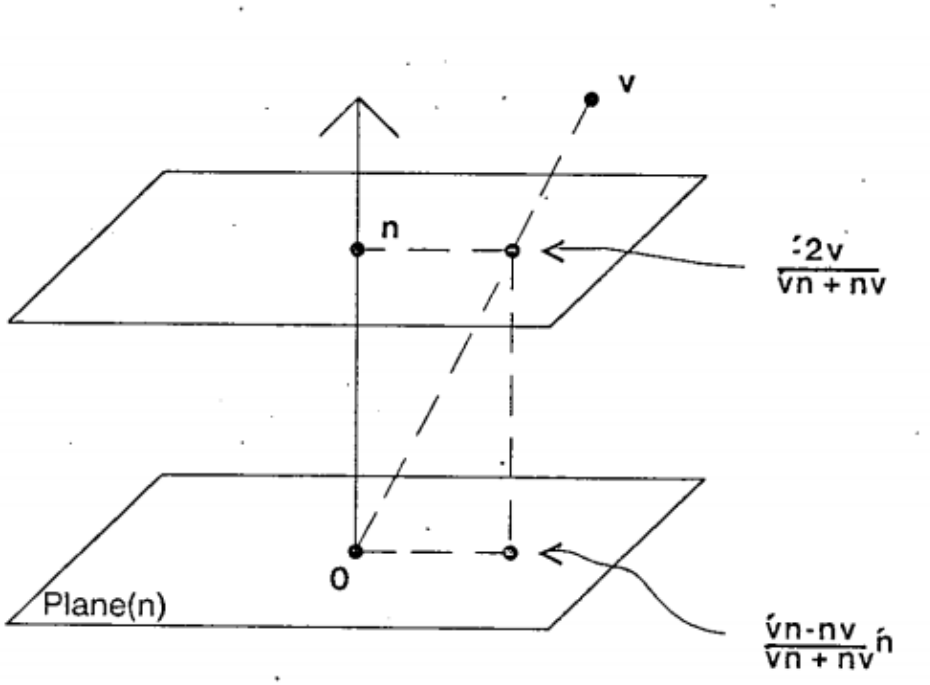


Figure 12: Parallel and centre projection

A general affine mapping can be represented as the composition of stretching's and rotations. However, if we are just studying a plane, all we need are compositions of rotations arid projections. In particular, consider the mapping

$$v \rightarrow \frac{RvR^{-1} + nRvR^{-1}n}{2}$$

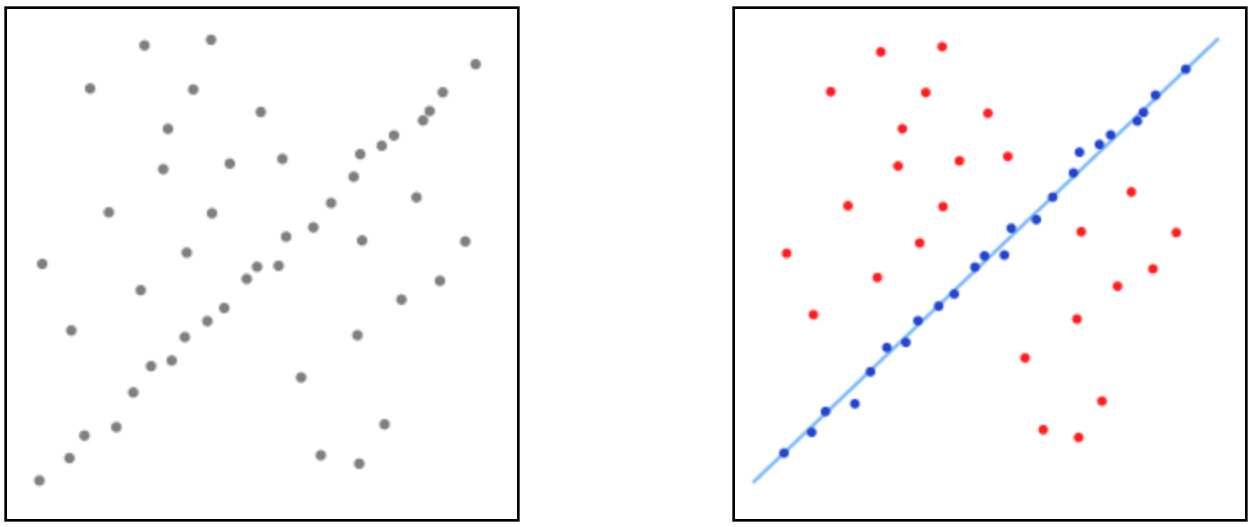
where R is some rotation $e^{\theta p}$. This mapping will have the effect of rotating v by an angle ɵ about the axis p, and then projecting it onto Plane (n). If we allow R to be any quaternion, and not just a unit quaternion (a rotation), we can represent any affine transformation in this way, and can think of R as representing the affine transformation. EKF prediction is followed by 1-point RANSAC algorithm. Before that, a brief explanation of RANSAC is given in next topic.

4.5 RANSAC

4.5.1 Introduction:

RANSAC is an abbreviation for "RANdom SAmple Consensus". It is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. It is a non-deterministic algorithm in the sense that it produces a reasonable result only with a certain probability, with this probability increasing as more iterations are allowed. The algorithm was first published by Fischler and Bolles at SRI International in 1981.

A basic assumption is that the data consists of "inliers", i.e., data whose distribution can be explained by some set of model parameters, though may be subject to noise, and "outliers" which are data that do not fit the model. The outliers can come, e.g., from extreme values of the noise or from erroneous measurements or incorrect hypotheses about the interpretation of data. RANSAC also assumes that, given a (usually small) set of inliers, there exists a procedure which can estimate the parameters of a model that optimally explains or fits this data.



(a) (b)

Figure 13: (a) A data set with many outliers for which a line has to be fitted.
(b) Fitted line with RANSAC; outliers have no influence on the result.

Figure 13 illustrates a simple example is fitting of a line in two dimensions to a set of observations. Assuming that this set contains both inliers, i.e., points which approximately can be fitted to a line, and outliers, points which cannot be fitted to this line, a simple least squares method for line fitting will in general produce a line with a bad fit to the inliers. The reason is that it is optimally fitted to all points, including the outliers. RANSAC, on the other hand, can produce a model which is only computed from the inliers, provided that the probability of choosing only inliers in the selection of data is sufficiently high. There is no guarantee for this situation, however, and there are a number of algorithm parameters which must be carefully chosen to keep the level of probability reasonably high.

4.5.2 Overview:

The input to the RANSAC algorithm is a set of observed data values, a parameterized model which can explain or be fitted to the observations, and some confidence parameters.

RANSAC achieves its goal by iteratively selecting a random subset of the original data. These data are *hypothetical inliers* and this hypothesis is then tested as follows:

1. A model is fitted to the sample of hypothetical inliers, i.e. all free parameters of the model are fitted from this sample.

2. All other data are then tested against the fitted model and, those points that fit the estimated model well are considered as part of the *consensus set*.
3. The estimated model is reasonably good if sufficiently many points have been classified as part of the consensus set.
4. Afterwards, the model may be improved by re-estimating it using all members of the consensus set.

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are part of the consensus set, or a refined model together with a corresponding consensus set size. In the latter case, we keep the refined model if its consensus set is larger than the previously saved model. The sequence of steps required for the algorithm are given below:

Algorithm

1: Select randomly the minimum number of points required to determine the model parameters.

2: Solve for the parameters of the model.

3: Determine how many points from the set of all points fit with a predefined tolerance ϵ

4: If the fraction of the number of inliers over the total number of points in the set exceeds a predefined threshold r, re-estimate the model parameters using all the identified inliers and terminate.

5: Otherwise, repeat steps 1 through 4 (maximum of N times).

The parameters

The values of parameters *t* and *d* have to be determined from specific requirements related to the application and the data set, possibly based on experimental evaluation. The parameter *k* (the number of iterations), however, can be determined from a theoretical result. Let *p* be the probability that the RANSAC algorithm in some iteration selects only inliers from the input data set when it chooses the *n* points from which the model parameters are estimated. When this happens, the resulting model is likely to be useful so *p* gives the probability that the algorithm produces a useful result. Let *w* be the probability of choosing an inlier each time a single point is selected, that is,

w = number of inliers in data / number of points in data

A common case is that w is not well known beforehand, but some rough value can be given. Assuming that the n points needed for estimating a model are selected independently, w^n is the probability that all n points are inliers and 1-w^n is the probability that at least one of the n points is an outlier, a case which implies that a bad model will be estimated from this point set. That probability to the power of k is the probability that the algorithm never selects a set of n points which all are inliers and this must be the same as 1 - p.

Consequently,

$$1 - p = (1 - w^n)^k$$

this, after taking the logarithm of both sides, leads to

$$k = \frac{\log(1 - p)}{\log(1 - w^n)}$$

This result assumes that the n data points are selected independently, that is, a point which has been selected once is replaced and can be selected again in the same iteration. This is often not a reasonable approach and the derived value for k should be taken as an upper limit in the case that the points are selected without replacement. For example, in the case of finding a line which fits the data set illustrated in the above figure, the RANSAC algorithm typically chooses two points in each iteration and computes maybe model as the line between the points and it is then critical that the two points are distinct.

To gain additional confidence, the standard deviation or multiples thereof can be added to k. The standard deviation of k is defined as

$$SD(k) = \frac{\sqrt{1 - w^n}}{w^n}$$

4.5.3 Advantages and disadvantages

An advantage of RANSAC is its ability to do robust estimation of the model parameters, i.e., it can estimate the parameters with a high degree of accuracy even when a significant number of outliers are present in the data set. A disadvantage of RANSAC is that there is no upper bound on the time it takes to compute these parameters. When the number of iterations computed is limited the solution obtained may not be optimal, and it may not even be one that fits the data in a good way. In this way RANSAC offers a trade-off; by computing a greater number of iterations the probability of a reasonable model being produced is increased. Moreover, RANSAC is not always able to find the optimal set even for moderately contaminated sets and it usually performs badly when the number of inliers is less than 50%. Optimal RANSAC was proposed to handle both these problems and is capable of finding the optimal set for heavily contaminated sets, even for an inlier ratio under 5%. Another disadvantage of RANSAC is that it requires the setting of problem-specific thresholds.

RANSAC can only estimate one model for a particular data set. As for any one-model approach when two (or more) model instances exist, RANSAC may fail to find either one. The Hough transform is one alternative robust estimation technique that may be useful when more than one model instance is present. Another approach for multi model fitting is known as PEARL, which combines model sampling from data points as in RANSAC with iterative re-estimation of inliers and the multi-model fitting being formulated as an optimization problem with a global energy functional describing the quality of the overall solution.

4.5.4 Applications:

The RANSAC algorithm is often used in computer vision, e.g., to simultaneously solve the correspondence problem and estimate the fundamental matrix related to a pair of stereo cameras.

Because of limitations of RANSAC, a new algorithm called 1-point RANSAC is used in this project, which is explained in next topic.

4.6 1-POINT RANSAC FOR EKF ESTIMATION

4.6.1 Introduction:

In the last few years, a very established method for removing outliers has been the "5-point RANSAC" algorithm which needs a minimum of 5 point correspondences to estimate the model hypotheses. Because of this, however, it can require up to several hundreds of iterations to find a set of points free of outliers. Using a single feature correspondence for motion estimation is the lowest model parameterization possible and results in the most efficient algorithms for removing outliers:1-point RANSAC.

While at least 5 points would be needed to compute monocular Structure from Motion for a calibrated camera if no prior knowledge is available, fewer are needed as more information is introduced: as few as 2 points for planar motion and 1 point in for planar and nonholonomic motion. As a clear limitation of both approaches, any motion performed out of the model will result in estimation error. In fact, it is shown in real-image experiments in that, although the most constrained model is enough for RANSAC hypotheses (reaching then 1-point RANSAC), a less restrictive one offers better results for motion estimation.

In the case of the new 1-point RANSAC presented here, extra information for the camera motion comes from the probability distribution function that the EKF naturally propagates over time. No camera motion constraints need to be added in order to successfully use a 1-point RANSAC hypothesis, and the full 6 degrees of freedom of camera motion can be dealt with using the proposed method.

Computational savings with respect to standard RANSAC can be easily derived. The number of RANSAC random hypothesis n_{hyp} necessary to guarantee that at least one of them is mismatch-free with probability p can be computed using the following formula:

$$n_{hyp} = \frac{log\,(1-p)}{log\,(1-(1-\epsilon)^{m})}$$

where ϵ is the assumed inlier ratio and m the number of measurements that instantiate the model. As a simple but illustrative example, if the inlier ratio ϵ is 0:5 and the probability p equals 0:99,

the number of random hypothesis would be reduced from 146 (m = 5; no prior information used) to only 7 for m = 1 (1-point RANSAC using prior information).

Efficiency in our 1-point RANSAC also derives from the fact that random RANSAC hypothesis evaluation does not imply an expensive EKF covariance update, whose complexity is cubic in the size of the estimated parameters. Support for each hypothesis is evaluated by comparing the innovation against a threshold; only a state update is necessary to compute it. Our matching algorithm is fully detailed in pseudocode in the algorithm given in the following page.

4.6.2 Algorithm 1-Point RANSAC for EKF estimation

INPUT: $\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}$ {EKF estimate at step $k-1$}
  th {Threshold for low-innovation points. In this paper, $th = 2\sigma\_pixels$}
OUTPUT: $\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}$ {EKF estimate at step k}

{A. EKF prediction and individually compatible matches}
$[\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}] = EKF\_prediction(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}, \mathbf{u})$
$[\hat{\mathbf{h}}_{k|k-1}, \mathbf{S}_{k|k-1}] = measurement\_prediction(\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$
$\mathbf{z}^{IC} = search\_IC\_matches(\hat{\mathbf{h}}_{k|k-1}, \mathbf{S}_{k|k-1})$

{B. 1-Point hypotheses generation and evaluation}
$\mathbf{z}^{li\_inliers} = [\,]$
$n_{hyp} = 1000$ {Initial value, will be updated in the loop}
for $i = 0$ to n_{hyp} **do**
  $\mathbf{z}_i = select\_random\_match(\mathbf{z}^{IC})$
  $\hat{\mathbf{x}}_i = EKF\_state\_update(\mathbf{z}_i, \hat{\mathbf{x}}_{k|k-1})$ {Notice: only state update; NO covariance update}
  $\hat{\mathbf{h}}_i = predict\_all\_measurements(\hat{\mathbf{x}}_i)$
  $\mathbf{z}_i^{th} = find\_matches\_below\_a\_threshold(\mathbf{z}^{IC}, \hat{\mathbf{h}}_i, th)$
  if $size(\mathbf{z}_i^{th}) > size(\mathbf{z}^{li\_inliers})$ **then**
    $\mathbf{z}^{li\_inliers} = \mathbf{z}_i^{th}$
    $\epsilon = 1 - \frac{size(\mathbf{z}^{li\_inliers})}{size(\mathbf{z}^{IC})}$
    $n_{hyp} = \frac{log(1-p)}{log(1-(1-\epsilon))}$
  end if
end for

{C. Partial EKF update using low-innovation inliers}
$[\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}] = EKF\_update(\mathbf{z}^{li\_inliers}, \hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$

{D. Partial EKF update using high-innovation inliers}
$\mathbf{z}^{hi\_inliers} = [\,]$
for every match $\mathbf{z}^j$ above a threshold th **do**
  $[\hat{\mathbf{h}}^j, \mathbf{S}^j] = point\_j\_prediction\_and\_covariance(\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}, j)$
  $\nu^j = \mathbf{z}^j - \hat{\mathbf{h}}^j$
  if $\nu^{j\top} \mathbf{S}^{j-1} \nu^j < \chi^2_{2,0.01}$ **then**
    $\mathbf{z}^{hi\_inliers} = add\_match\_j\_to\_inliers(\mathbf{z}^{hi\_inliers}, \mathbf{z}^j)$ {If individually compatible, add to inliers}
  end if
end for
if $size(\mathbf{z}^{hi\_inliers}) > 0$ **then**
  $[\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}] = EKF\_update(\mathbf{z}^{hi\_inliers}, \hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k})$
end if

These algorithm steps are explained in the following units.

4.7 Search for Individually Compatible Matches:

The algorithm begins with standard EKF prediction: the estimation for the state vector, $x_{k-1|k-1}$ at step k-1, modeled as a multidimensional Gaussian

$$\mathbf{x}_{k-1|k-1} \sim \mathcal{N}\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}\right)$$

This is propagated to step k through the known dynamic model f_k.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}_k\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\right) \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{G}_k \mathbf{Q}_k \mathbf{G}_k^\top \end{aligned}$$

Here,

U_k → Control inputs given to the system at step k

F_k → Jacobian of fk with respect to the state vector $x_{k|k-1}$ at step k

Q_k → Covariance of the zero-mean Gaussian noise assumed for the dynamic model

G_k → Jacobian of this noise with respect to the state vector $x_{k|k-1}$

Propagating this predicted state $x_{k|k-1}$ through the measurement model h_i offers a Gaussian prediction for each measurement.

$$\begin{aligned} \hat{\mathbf{h}}_i &= \mathbf{h}_i\left(\hat{\mathbf{x}}_{k|k-1}\right) \\ \mathbf{S}_i &= \mathbf{H}_i \mathbf{P}_{k|k-1} \mathbf{H}_i^\top + \mathbf{R}_i \end{aligned}$$

where H_i is the Jacobian of the measurement hi with respect to the state vector , R_i is the covariance of the Gaussian noise assumed for each individual measurement. The actual measurement z^i should be exhaustively searched for inside the 99% probability region defined by its predicted Gaussian $N(\hat{h}_i, S_i)$ by comparison of the chosen local feature descriptor. This is followed by 1-point RANSAC hypothesis and selection of low innovation inliers.

4.8 1-Point Hypothesis and Selection of Low Innovation Inliers:

Following the principles of RANSAC, random state hypotheses x^i are generated and data support is computed by counting measurements inside a threshold. It is assumed that we are considering problems where the predicted measurements are highly correlated, such that every hypothesis computed from one match reduces most of the correlation in the measurement prediction, with inlier uncertainty close to the measurement noise. The threshold is fixed according to X^2 test with significance a^2= 0:05.

As the key difference with respect to standard RANSAC, random hypotheses will be generated not only based on the data $z^{IC} = (\mathrm{z}_1 \ldots\ldots \mathrm{z}_i \ldots\ldots \mathrm{z}_n)^{\mathrm{T}}$ but also on the predicted state $x_{k|k-1} \sim N(\hat{x}_{k|k-1}, P_{k|k-1})$. Exploiting this prior knowledge allows us to reduce the sample size necessary to instantiate the model parameters from the minimal size to define the degrees of freedom of the model to only one data point. The termination criteria of the RANSAC algorithm grows exponentially with the sample size and means a great reduction in the number of hypotheses.

The input to the 1-point RANSAC algorithm is an initial set of individually compatible matches z, extracted by cross-correlation search in the 99% probability region given by the predicted probability distribution function over the measurements$N(h_k(\hat{x}_{k|k-1}), S_k)$. Although this so called active search filters out several outliers, the initial set z may still contain some of them.

The hypothesize-and-vote loop, where the main difference with plain RANSAC resides, operates as follows: first, a random match zi is extracted from the individually compatible set z. Using this single random match and the state prediction from the EKF (following the Gaussian $N(\hat{x}_{k|k-1}, P_{k|k-1})$ the state is updated. It is worth noticing again that only the state $\hat{x}$ is updated in the loop and not the covariance P, keeping the computational complexity low.

After each 1-match state update, the residual innovation is computed for the rest of the matches. Those below a heuristic threshold, in our experiments 1:0 pixels, are considered low innovation inliers and support the current hypothesis.

Another key aspect for the efficiency of the algorithm is that each hypothesis $\widehat{x_i}$ generation only needs an EKF state update using a single match z^i. A covariance update, which is of quadratic complexity in the size of the state, is not needed and the cost per hypothesis will be low. Hypothesis support is calculated by projecting the updated state into the camera, which can also be performed at very low cost compared with other stages in the EKF algorithm. Next step is to partially update using low innovation inliers.

4.9 Partial Update using Low Innovation Inliers:

The data points voting for the most supported hypothesis $z^{li\_inliers}$ are designated as Low-innovation inliers. They are not at a small distance from the most supported hypothesis and are assumed to be generated by the true model. The rest of the points can be outliers but also inliers, even if they are far from the most supported hypothesis.

A simple example from visual estimation can illustrate this: it is well known that distant points are useful for estimating camera rotation, while close points are necessary to estimate translation. In the RANSAC hypotheses generation step, a distant feature would generate a highly accurate 1-point hypothesis for rotation, while translation would remain inaccurately estimated. Other distant points would in this case have low innovation and would vote for this hypothesis. But as translation is still inaccurately estimated, nearby points would presumably exhibit high innovation even if they are inliers.

So after having determined the most supported hypothesis and the other points that vote for it, some inliers still have to be “rescued" from the high-innovation set. Such inliers will be rescued after a partial state and covariance update using only the reliable set of low-innovation inliers:

$$\begin{aligned}
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{z}^{li_inliers} - \mathbf{h}' \left(\hat{\mathbf{x}}_{k|k-1} \right) \right) \\
\mathbf{P}_{k|k} &= \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k \right) \mathbf{P}_{k|k-1} \\
\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \left(\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k \right)^{-1} .
\end{aligned}$$

$H_k = (H_1 \ldots . H_i \ldots . H_{n'})^T$ stands for the Jacobian of the measurement equation that projects the low-innovation inliers into the sensor space. R_k is the covariance assigned to the sensor noise. This is followed by the rescue of high innovation inliers.

4.10 Rescue High Innovation Inliers:

Residual innovation will be high for outliers but also for some high-innovation inliers, which correspond to recently initialized points with uncertain depth estimates or close points significantly affected by camera translation. In order to finally discard spurious matches from this high-innovation set, the following rule is applied: high-innovation inliers will be individually compatible after a partial update eliminating correlated error; while outliers will not.

A state and covariance update is therefore carried out using all the low-innovation inliers from the first step. Individual compatibility will be evaluated for each one of high-innovation matches; those inside the 99% probability ellipse will be accepted as inliers, while those outside the ellipse will finally classified as outliers.

4.11 Partial Update using High Innovation Inliers:

After a partial update using low-innovation inliers, most of the correlated error in the EKF prediction is corrected and the covariance is greatly reduced. This high reduction will be exploited for the recovery of high-innovation inliers: as correlations have weakened, consensus for the set will not be necessary to compute and individual compatibility will suffice to discard inliers from outliers.

An individual Gaussian prediction $h^j \sim N(h^j, S^j)$ will be computed for each high innovation for every match z^j by propagating the state after the first partial update $x_{k|k}$ through the projection model. The match will be accepted as an inlier if it lies within the 99% probability region of the predicted Gaussian for the measurement.

After testing all the high-innovation measurements a second partial update will be performed with all the points classified as inliers z^{hi} inliers, following the usual EKF equations. A summary of this SLAM algorithm is given in next topic.

4.12 Summary

Using the MonoSLAM algorithm, we could build online only a sparse map of point landmarks and show that despite the challenges presented by high acceleration motion, we can form a persistent map which permits drift free real time localization over a small area. The important requirement is that localization and mapping should be repeatable so that uncertainty in the camera's position does not increase with time during these repeated movements.

The 3D trajectory of the monocular camera moving in an unknown scene is also known used EKF (Extended Kalman Filter) and outlier rejection is also done using 1-point RANSAC algorithm.

Finally, this chapter has described the SLAM problem, the essential methods for solving the SLAM problem and has summarized key implementations and demonstrations of the method. While, there are still many practical issues to overcome, especially in more complex outdoor environments, the general SLAM method is now a well understood and established part widely used in robotics. The next chapter gives the results of both BRISK and SLAM algorithms.

5. RESULTS

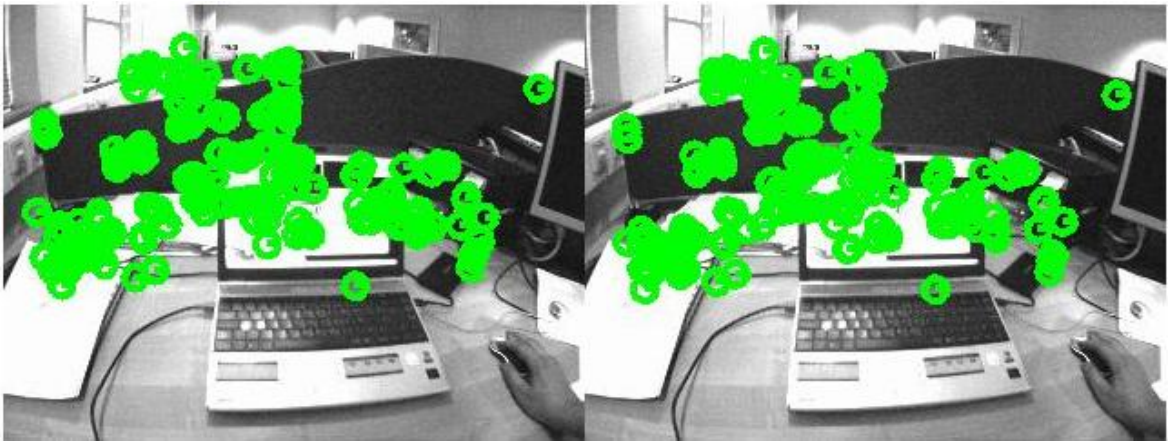
5.1 BRISK

In order to verify the BRISK algorithm proposed in previous chapters, a video sequence is been taken such that it has a lot of information regarding features, like robust corners, points, edges, blobs etc. From this video, any two successive frames are extracted and the code is implemented on these frames. The following figures show the simulation results.

Frame 1 Frame 2

Result of Feature Detection:

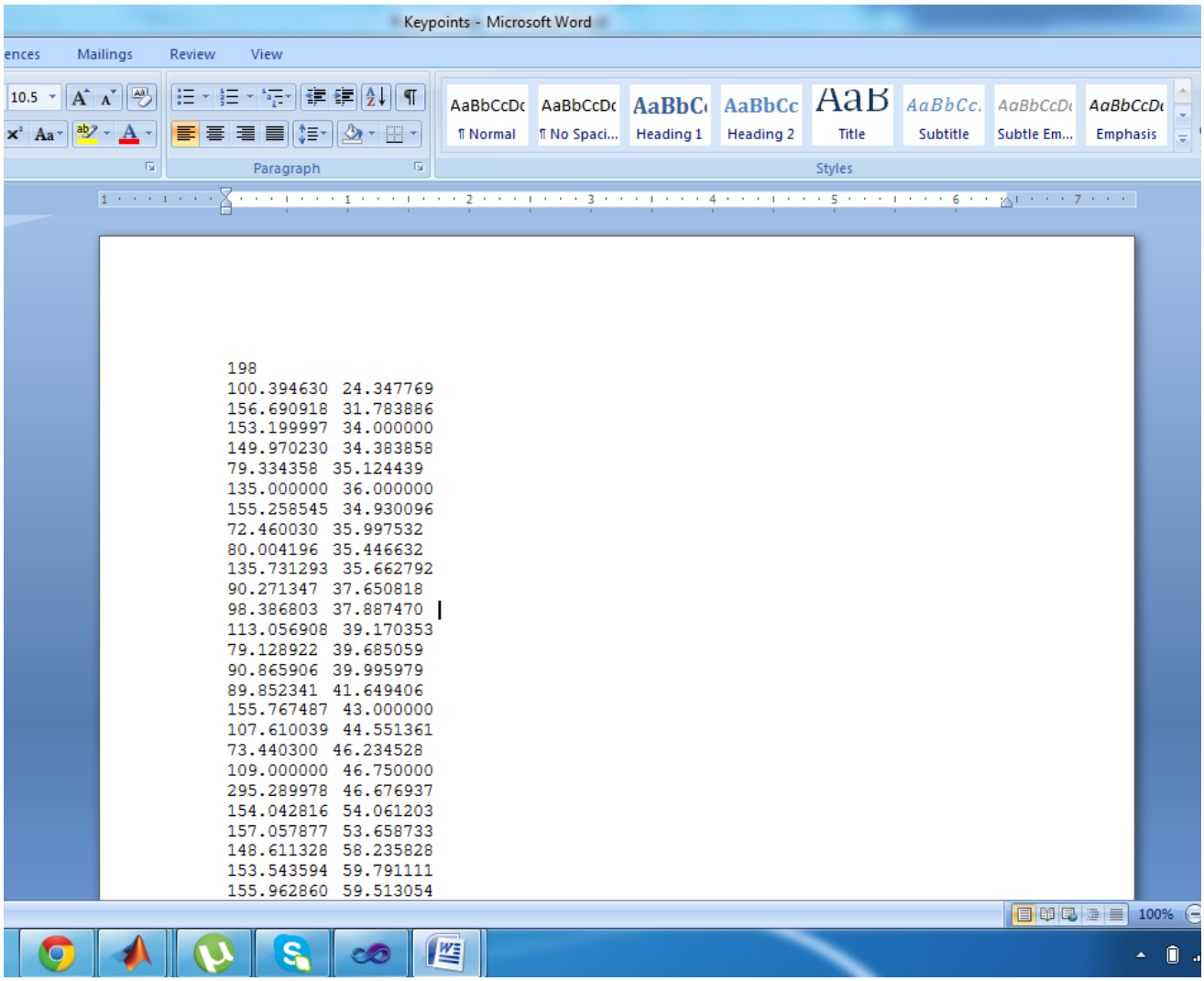
Using FAST, the features from these two frames are detected. The following figure shows the result of feature detection. The green circles in the image are the features.



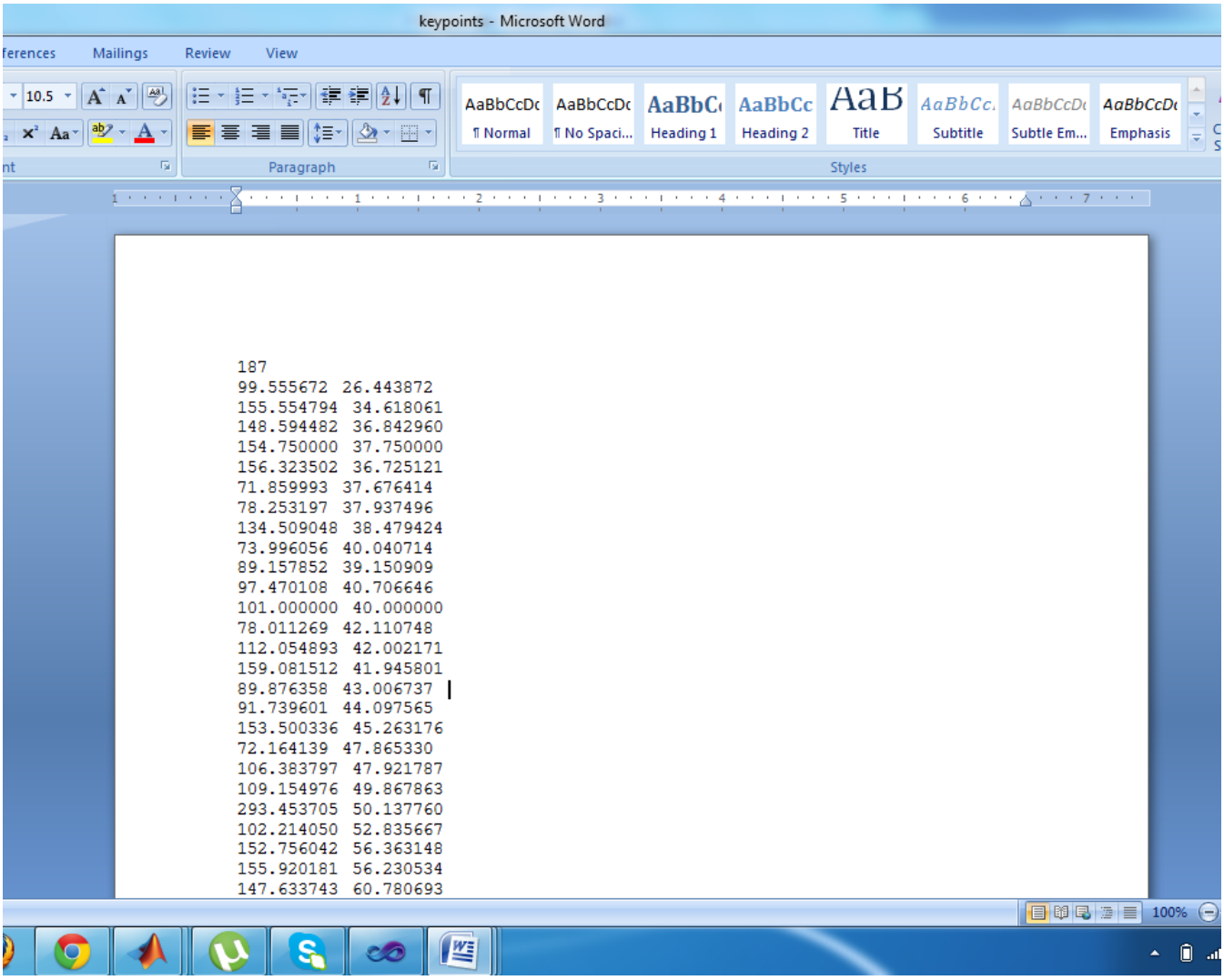
Frame 1 Frame 2

Result of Feature Extraction:

In the feature extraction step, from the features detected, some of them are deleted and more robust features are extracted. Here, 198 keypoints are obtained from frame 1 and 187 keypoints are obtained from frame 2. The x and y co-ordinates of the features are stored in a MS-Word Document.



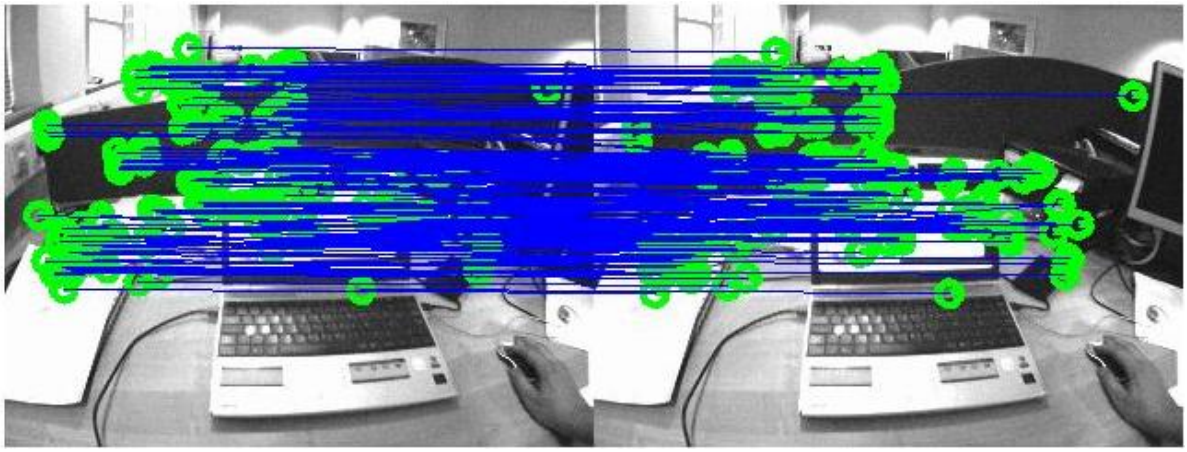
Keypoints from frame 1



Keypoints from frame 2

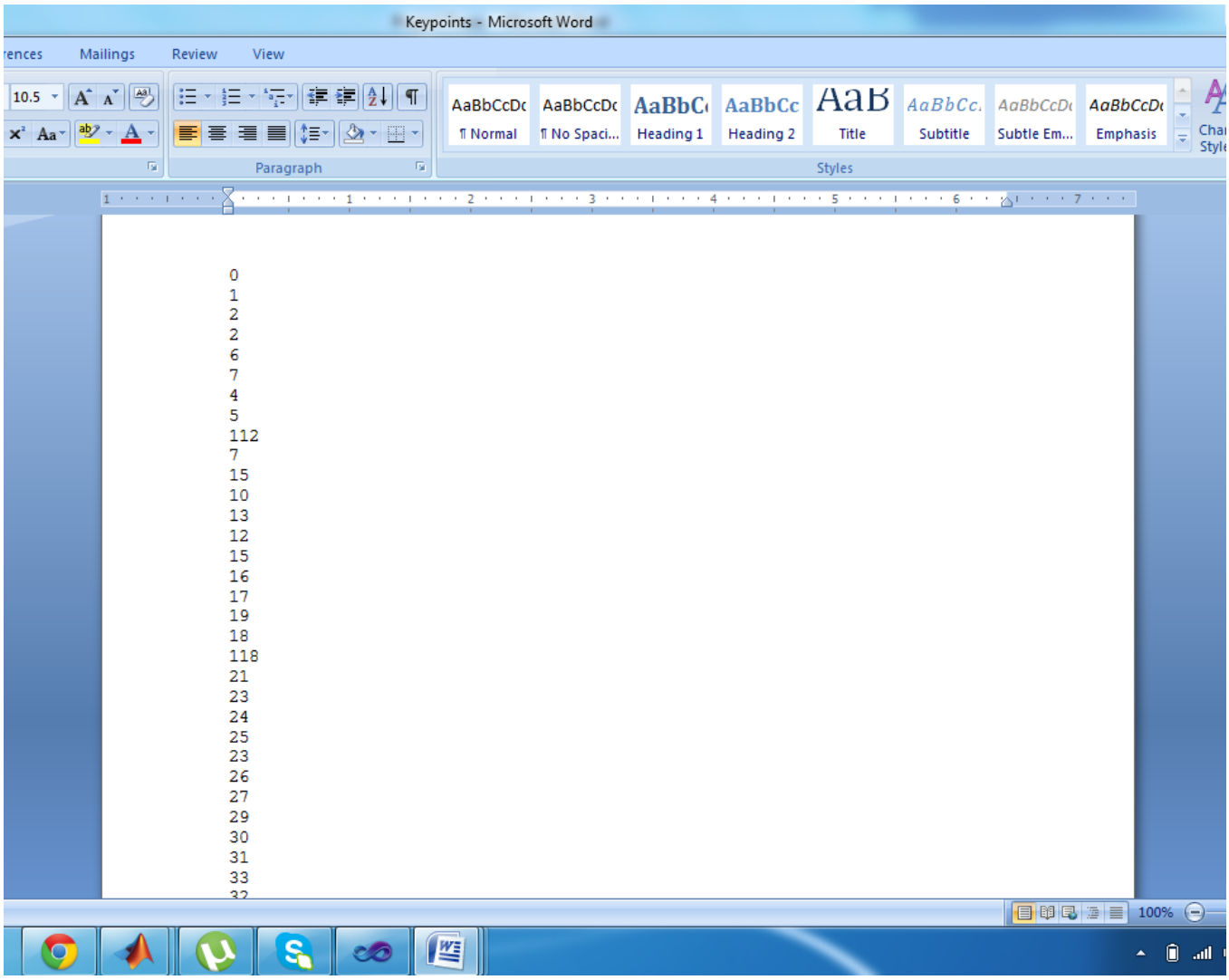
Result of Feature Matching:

After extracting features from frames 1 & 2, the hamming distance algorithm is implemented and the matched features are determined. The following figure illustrates the results of feature matching between frame1 and frame2. The blue lines in the figure indicate the matched points in the two consecutive frames of the video sequence.



Frame 1 Frame 2

The best matches obtained are stored in a MS-Word document.



The document stores which keypoint of frame 1 is matched with a keypoint of frame 2. Here, it can be seen that keypoint1 of frame1 is not matched with any keypoint in frame2 and is given a value '0'. Keypoint2 of frame1 is matched with keypoint1 of frame2 and similarly keypoint3 of frame 1 is matched with keypoint2 of frame 2 and so on.

5.2 SLAM

A video sequence is captured from a real time camera the mono SLAM algorithm is run on this video.

Map Management

A map is being constructed from an unknown evironment i.e. without any prior knowledge.

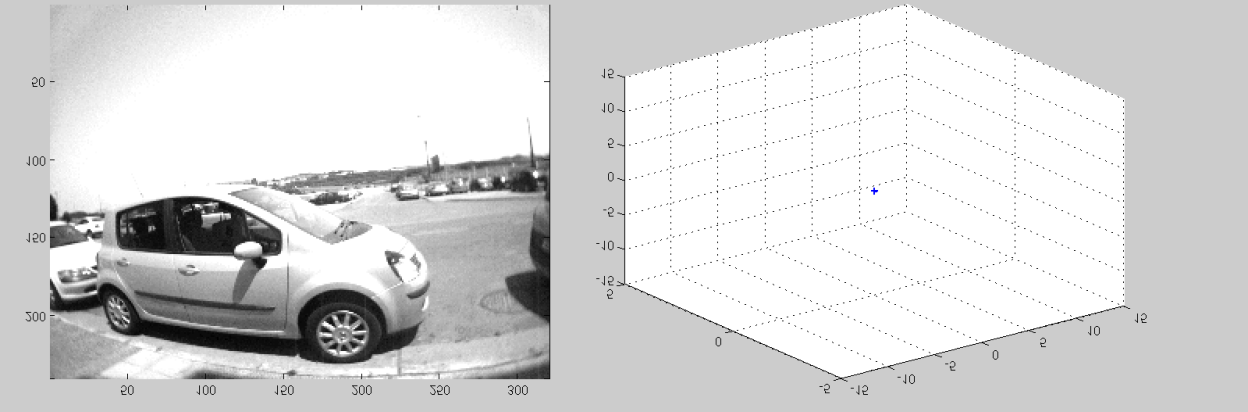
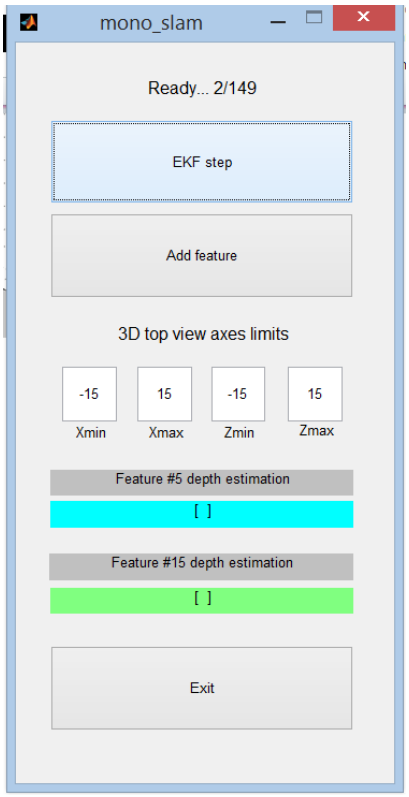


Figure a indicates the initial output of the algorithm without adding any features.

Figure b is a matlab generated Graphical User Interface. Using this, we can add features, update their information using EKF step and estimate the depth.



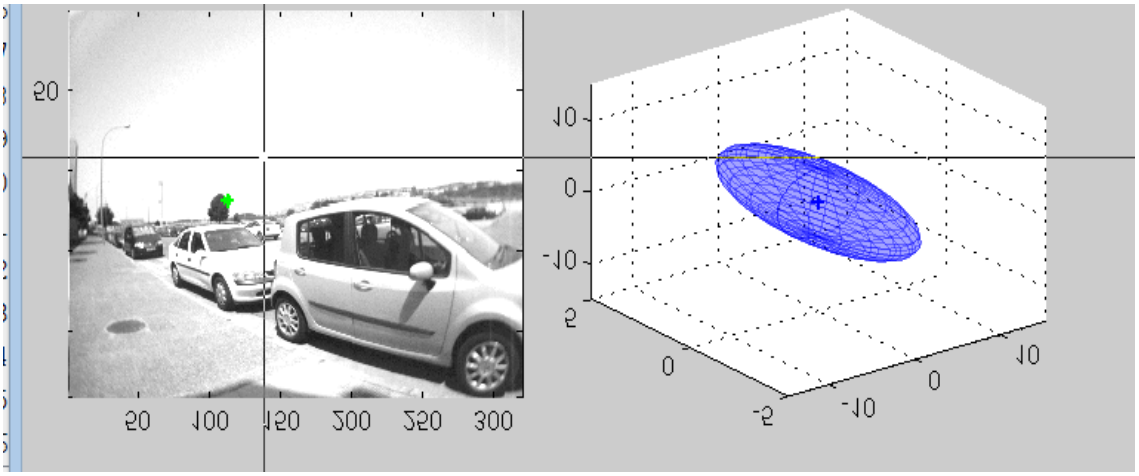


Figure c illustrates the addition of feature at a particular location using horizontal and vertical axes

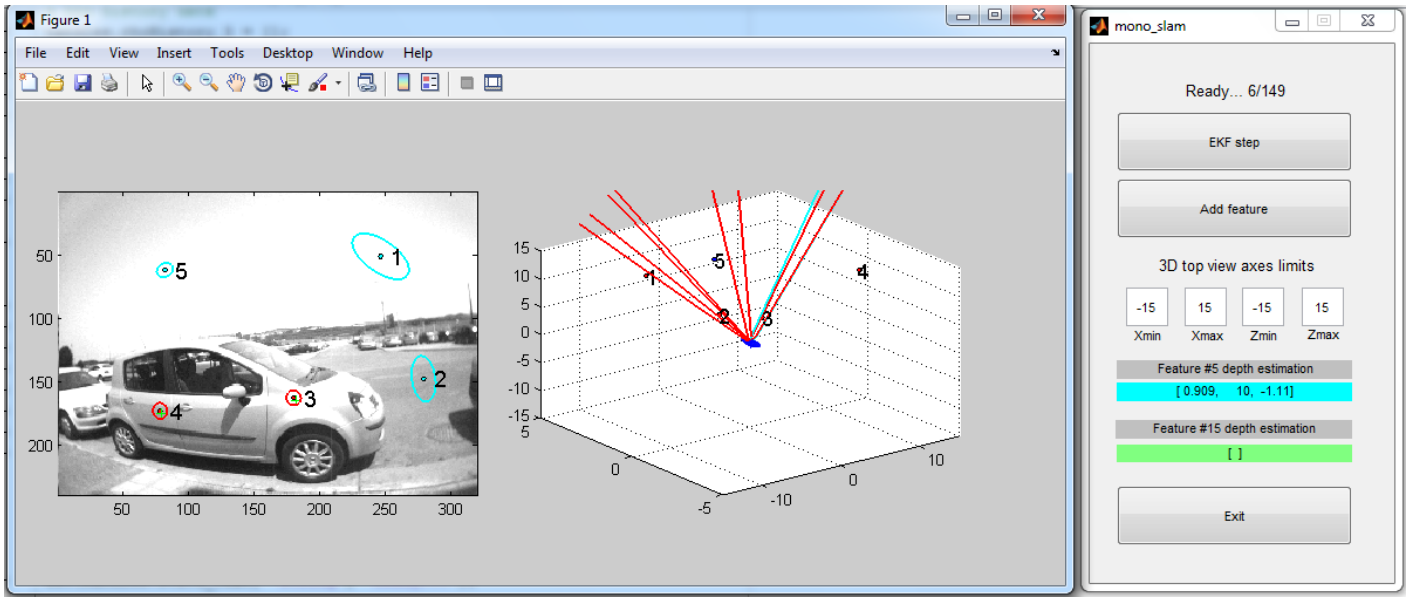
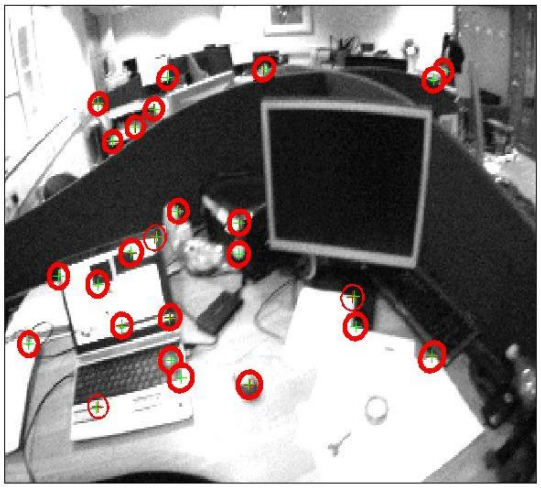
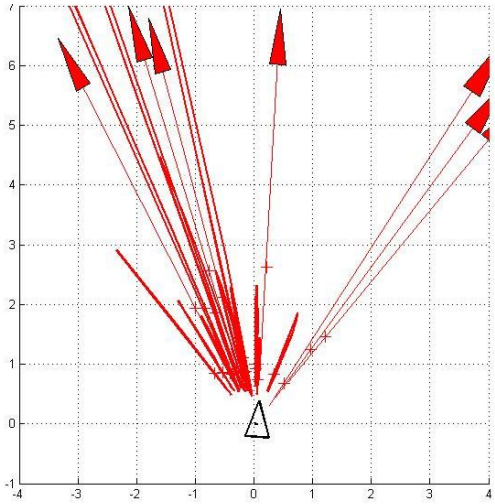


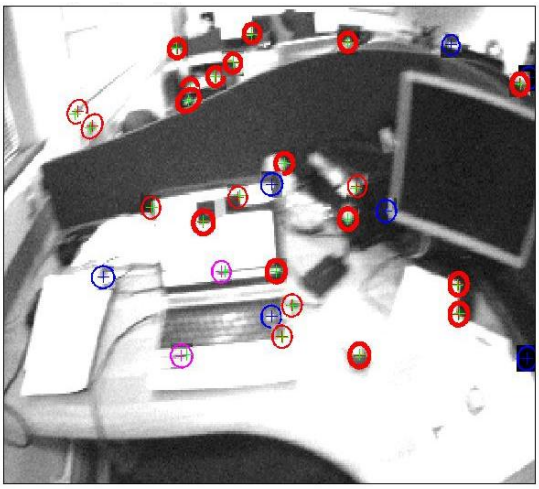
Figure d: 1, 2, 3, 4, 5 are the features added into the video sequence. EKF step is run to update their information. Red circles indicate that the predicted state estimate is matched with the actual measurement. Cyan circles indicate that they are not matched.

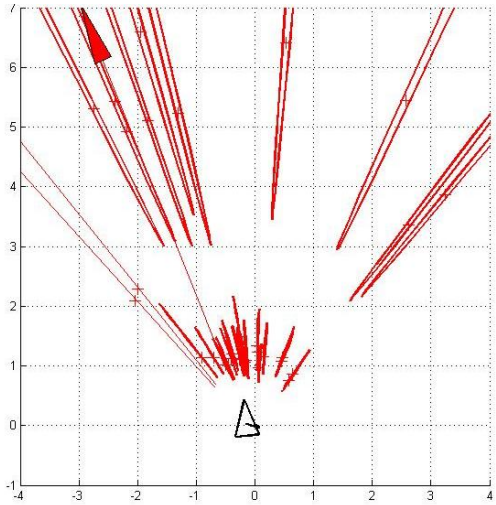
1-point RANSAC

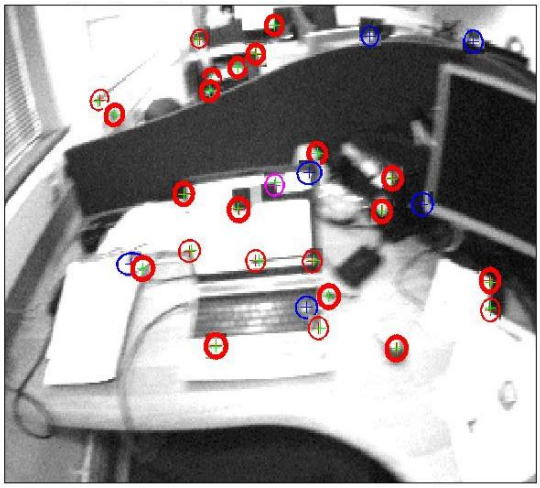
The map management is followed by 1-point RANSAC hypothesis in order to obtain the trajectory of the camera. The following sequence of figures show the simulation result of 1-point RANSAC hypothesis on a video. The right hand side figures indicate the trajectory of the camera where the camera is represented by a triangle.

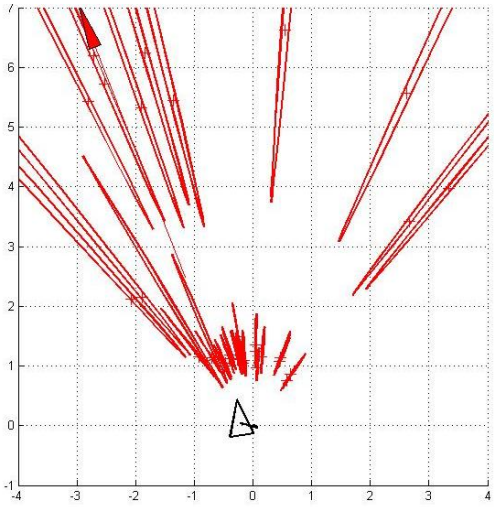


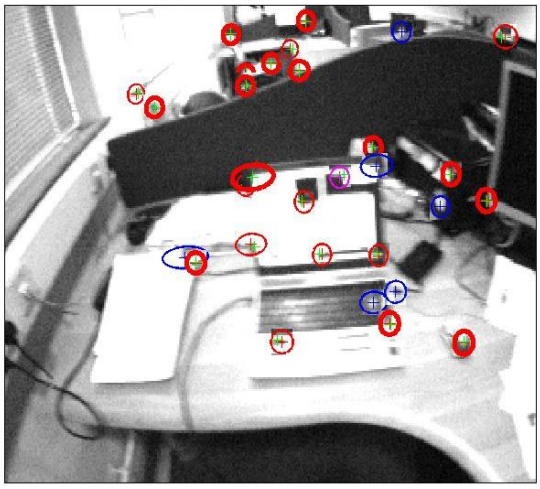


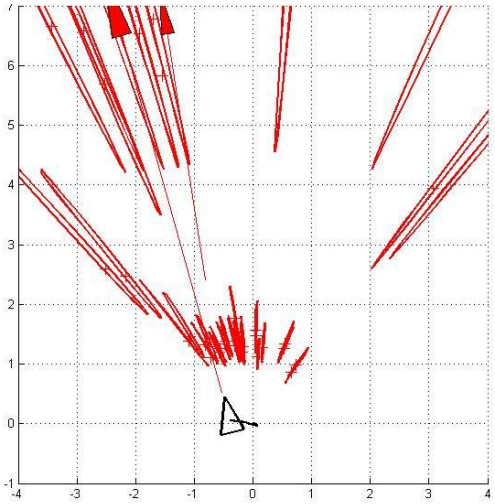












The following indicates the inliers

Thick red →Low innovation inliers

Thin red →High innovation inliers

Magenta→ 1-point RANSAC

Blue → No match found by cross correlation

Camera motion and scene in top view

The red arrow marks in the figure represents the point feature in infinite depth uncertainty.

6. CONCLUSION AND FUTURE SCOPE

This project implements two major techniques, BRISK and SLAM. The BRISK deals with the classic computer vision problem of detecting, extracting and matching features of images from a video sequence without having a prior knowledge on scene and camera poses. BRISK relies on an easily configurable circular sampling pattern from which it computes brightness comparisons to form a binary descriptor string. The unique properties of BRISK can be useful for a wide spectrum of applications, in particular for tasks with hard real-time constraints or limited computation power. BRISK finally offers the quality of high-end features in such time-demanding applications.

We have also explained MonoSLAM, a real time algorithm for simultaneous localization and mapping with a single freely moving camera. Using SLAM, we could successfully build a map in an unknown environment. SLAM is also used to update a map within a known environment while at the same time keeping track of the current location, which is used to track the camera pose while building a 3D map of unknown scene.

In future work, the results of BRISK can be combined with that of SLAM and the pose of the camera can be estimated. This algorithm has an impact in application areas including low cost and advanced robotics, wearable computing, augmented reality for industry and entertainment and user interfaces.

The performance of the algorithm can be improved to cope with larger environments, more dynamic motions and more complicated scenes with significant occlusions. With the information provided by camera pose estimation, we can insert virtual objects into a real video sequence, where we can have a live direct or indirect view of a physical, real world environment whose elements are augmented by computer generated sensory input such as sound, video, graphics or GPS data, which is nothing but Augmented Reality.

REFERENCES

[1] Stefan Leutenegger, Margarita Chli and Roland Y. Siegwart. *BRISK: Binary Robust Invariant Scalable Keypoints*. Autonomous Systems Lab, ETH Z¨urich.

[2] A. J. Davison, N. D. Molton, I. Reid, and O. Stasse. *MonoSLAM: Real-time single camera SLAM*. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 29(6):1052–1067, 2007.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. *SURF: Speeded up robust features*. Computer Vision and Image Understanding (CVIU), 110(3):346–359, 2008.

[4] Rodehorst V., Koschan A., 2006. Comparison and evaluation of feature point detectors. Proc. 5th International Symposium Turkish-German Joint Geodetic Days *Geodesy and Geoinformation in the Service of our Daily Life*, Technical University of Berlin, ISBN 3-9809030-4-4, Berlin, Germany.

[5] Rosten, Edward; Reid Porter, Tom Drummond (2010). *FASTER and better: A machine learning approach to corner detection*. IEEE Trans. Pattern Analysis and Machine Intelligence **32**: 105–119. doi:10.1109/TPAMI.2008.275

[6] S. Dasgupta, and A.Banerjee, An augmented-reality-based real-time panoramic vision system for autonomous navigation. *IEEE Trans.* Systems, Man and Cybernetics, Part A, vol. 36, no 1, pp. 154 –161, 2006

[7] F. Ababsa and M. Mallem, *Robust camera pose estimation combining 2D/3D points and lines tracking.* In Proc. of the 2005 IEEE International Symposium on Industrial Electronics (ISIE'08). Cambridge, UK, 2008, pp. 774-779.

[8] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, *A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem,* in Proc. IEEE Int. Conf. Robotics Automation, 2000, vol. 2, pp. 1009–1014.

[9] T. Bailey, J. Nieto, and E. Nebot. *Consistency of the FastSLAM algorithm*. In Proc. IEEE Int. Conf. Robotics and Automation, 2006.

[10] R.M. Haralick, "Pose Estimation From Corresponding Point Data". *IEEE Trans. Systems, Man, and Cybernetics*, vol. 19, no°6, pp. 1426-1446, 1989.

APPENDIX

INSTALLATION OF OPENCV IN WINDOWS:

1. Download the Open CV source forge.

2. Run the file on the system and generate all the files.

3. Set the system path as shown:

    My computer -- System settings -- Advance System Settings -- Advanced -- Environment variables -- path.

    At this path, give the path of Open CV to build to x86 to vc10 to bin.

    For example, C:\opencv\build\x86\vc10\bin

    Restart the system.

4. Go to Project Properties in Microsoft Visual Studio and go to VC++ Directories in Configuration Properties.

I. In include directories give the path till open cv, build and include. For example, C:\opencv\build\include

II. In library directories include the path from open cv to build to x86 to vc10 to lib. For example, C:\opencv\build\x86\vc10\lib.

III. Go to Additional Dependencies of Input in Linker and add all the following header files

opencv_calib3d243d.lib
opencv_contrib243d.lib
opencv_core243d.lib
opencv_features2d243d.lib
opencv_flann243d.lib
opencv_gpu243d.lib
opencv_haartraining_engined.lib
opencv_highgui243d.lib
opencv_imgproc243d.lib
opencv_legacy243.lib

opencv_ml243d.lib
opencv_objdetect243d.lib
opencv_ts243d.lib
opencv_video243d.lib

Thus, the installation of Open CV is complete. If any linking error is shown then check for the path given for include directories and library directories.