VIDEO IMAGE PROCESSING USING MPEG TECHNOLOGY FOR A MOBILE ROBOT

By

Soradech Krootjohn

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

August, 2007

Nashville, Tennessee

Approved:

Professor D. Mitchell Wilkes

Professor George E. Cook

Professor Andrew W. Dozier

Professor Richard A. Peters

Professor Douglas P. Hardin

To my lovely family:

my parents (Lt. Chaiya and U-Thai)

and

my brothers (Denpong and Wanchai).

ACKNOWLEDGEMENTS

First of all, I would like to extend my gratitude to all of my committee members, Dr. Cook, Dr. Peters, Dr. Dozier, and Dr. Hardin. Also, I would like to give a big thanks to all of my friends in the Mobile Lab including Mert, Amy, Faisal, and Chris. The biggest thank you to the lab members has to go to Jonathan, "The Master of Procrastination", who kept me company day-and-night during my final phase towards this project completion.

Without the initial main financial support from the Thai Government, my study provided by Vanderbilt could never have taken place. Furthermore, I am very grateful for the additional financial support provided by the Vanderbilt office of Biomedical Research Education and Training (BRET). Especially, I am grateful to Dr. Chanchai McDonald, who introduced me to be a part of this prestigious institution.

I would like to give my appreciation to Anu and Dr. Natalie (Chapati & Macaroni) for their kind friendship and support from back stage. I am also grateful to Dr. Lason Watai, my first roommate. When you stay in one place for almost a decade, you gain a lot of friends. There are so many other people whom I would like to mention. However, although I am unable to bring up all the names here, you all are a part of my success.

Lastly, I am not able to find the words to say thank you to my Vanderbilt advisor, Professor Mitch Wilkes, who always kept my hope alive. If I was a rally driver, he would be my navigator. I would never have had a chance to find the finish line without his guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION AND RESEARCH MOTIVATION

Research into an autonomous mobile robot has drawn the attention of numerous research groups around the world. The ability to intelligently navigate through different environments requires a set of components that cooperatively perform the vehicle localization task. A mobile robot, in general, may be equipped with various types of sensors to accomplish its navigation task. Such sensory equipment includes sonars, bumper sensors, internal compass, wheel encoder, etc. A more sophisticated system may also employ a laser range finder, gyroscope correction system, or a Global Position System (GPS). However, these accessories are comparatively more expensive. Additionally, they mostly perform a single task specific to what they are designed for.

A video camera is also considered to be a common piece of sensory equipment. The captured image sequence, by its nature, contains an enormous amount of information to be extracted. Such information may include image intensity, color components, spatial motion, temporal motion, etc. Applications in digital image/video processing have been deployed in a variety of research areas. In medical research, for example, scientists utilize image processing to perform 3D segmentation of the internal structures in the MR images of the human brain [1] [2] for further analysis. Astronomers may use digital image processing to restore blurred star field images for discovering new objects in the universe [3]. Video surveillance may be used for security concerns. A suspicious person can be tracked by a real-time tracking system [4]. A face recognition system [5] can help

identify a person and may be used with a video surveillance system to authenticate an authorized person [6]. An application may use a simple motion detection technique to detect an intruder. Optical flow [7] can be used to model the camera motion, thus it can be utilized to stabilize a shaky digital video sequence [8] [9] that is caused by a moving camera. In addition, optical flow can be used to detect moving objects from a moving camera [10] [11] as well. Other research groups have adopted video processing analysis to control an autonomous driving system for a vehicle [12] [13]. These are only a few examples of how a video camera has been used in a myriad of research fields.

Recently, webcam prices keep have continuously dropped. A typical webcam capable of capturing RGB pictures may cost as little as $20. As a result of decreasing price, the webcam becomes almost a "necessity" that all household computers should have. Considering the fact that the video sequence contains substantial useful information as described above, video cameras have some promise to become cheap but effective robot sensors. One outstanding feature is that they can perform multiple tasks as a single unit. These encouraging factors then accelerate the use of video cameras among the research societies, especially in robotics.

Video cameras are employed in the navigation task in a variety of ways, ranging from heading detection, basic obstacle detection and avoidance, to visual odometry. A common method in many systems is to attach a video camera to the front of the vehicle, simply called an "observer". Motion fields are created while the observer is moving. With a set of constraints applied to how the camera is mounted and the environment where the observer is to be traveling, egomotion can then be estimated. Based on the perspective model [14], the observer's heading direction can be computed from 2D

velocity vectors, which radiate away from the vanishing point, called the *Focus of Expansion* (FOE).

Burger and Bhanu, [15] propose a technique, called "Fuzzy FOE", to compute the heading direction of a land-based vehicle. Instead of finding a single conventional FOE, a computation of a 2D region of FOE is performed, which they claim to be more robust than the conventional one. Dev et al. [16] proposed a technique to drive a mobile robot along the center of a corridor. The robot direction is achieved by computing the normal surface of the corridor's walls from the obtained optical flow field. The application can operate in real-time without additional supporting hardware.

Branca et al. [17] estimate the egomotion from the motion field to determine the robot's direction and the time to collide (TTC) with the environment. The mobile robot is restricted to travel on a flat surface in a stationary environment. Stoffler and Schnepf [18] [19] created an application for a mobile robot that is capable of detecting potential obstacles, roughly estimating their positions, and planning an avoidance task. The accuracy of the 3D reconstruction from the 2D motion field on the image plane is approximately 10cm.

Campbell et al. [20] proposed a new technique for estimating egomotion on which a video camera is mounted on a mobile robot so that the optical flow field is divided into two portions: the ground region and the sky region which are separated by a horizon line. The motion in the ground portion is used to compute the robot translation while the sky portion is used to compute the robot rotation. The application is created for the mobile robot's visual odometry. This work is then extended to detect obstacles and precipices in [21] based on the technique used in [20].

The above systems are implemented generally in such a way that a camera maps the 3D environment and projects it to the 2D image plane. Then, a motion field is created from the motion information on the image plane. One technique is to use a gradient method to compute the optical flow. The method is based on using spatial and temporal derivatives of the image intensity. The procedure is extensively time consuming, which may not be suitable for real-time systems [22]. Other techniques compute the displacement vectors by tracking a set of selected features over the video sequence. The method involves selecting good features. The problem in this method is how to define good features. Furthermore, a tracked feature will eventually disappear from the image plane. Thus, it is necessary to find new features to replace those that are about to move out of view. A block-based approach is then introduced to eliminate the long-term tracking. Instead of creating displacement vectors from the selected tracked feature, the current image is divided into blocks of sub-images. The displacement vectors are computed by applying a gradient method, finding correspondence features, or calculating a sum of absolute of the selected blocks with respect to a search region in the previous frame.

Research Motivation

As is widely known, in order to develop an application based on the above techniques, it can be very difficult in general and may require a high-level knowledge in mathematics. In real-time applications, the gradient method, in particular, demands high computational costs (and thus time) to process each image frame on a pixel-by-pixel

basis. Furthermore, the implementation of efficient code is very difficult and also requires a high level of programming skill in order to obtain highly efficient code that meets the real-time constraints.

The recent development of standardized streaming video technologies, such as MPEG-1, MPEG-2, MPEG-4, Quicktime®, and the H26x series, may be exploited as an existing platform of highly efficient code for developing computer or robot vision applications. These coding systems employ motion estimation and motion compensation methods based on blocks of an image, called "*macroblocks*", to predict a subsequent frame from the previously coded frame as its reference. The predicted frame basically consists of motion information, called "*motion vectors*", and the residual data that is associated with the particular macroblock. The motion vector is the offset from the current macroblock in the predicted frame to the reference region in the reference frame, whereas the residual data refer to the difference between the values in the current macroblock and the reference region. These motion vectors, especially, provide very valuable information that describes the motions occurring on a frame-to-frame basis in the video sequence. This useful information is freely available and can be accessed for further processing techniques and developments. A simple application such as a motion detection system can simply utilize these motion vectors in order to detect the changes in the image plane. If motion values become significant or above some threshold value, the system may alert the user by sending an email attached with the captured images or generate a warning sound to expel the intruder. However, a more sophisticated application may utilize the motion information to estimate the optical flow in the video sequence. This optical flow analysis is very useful for applications such as high-level

motion detection, video segmentation, object tracking [23], etc. Furthermore, it can also be used to model the motion of the camera and the moving objects in the scene [10], which is essential for an autonomous driving system for a ground vehicle [24], mobile robot [25], or helicopter [26]. Since the processing operation is performed on a block-based basis instead of pixelwise, the computational time spent on analyzing the motion information becomes smaller than pixel-based techniques. This is extremely suitable for real-time applications.

In this work, a low-cost solution to the visual odometry problem is proposed. A cheap webcam is attached to the front side of a mobile robot traveling on flat surfaces and restricted to performing translation and rotation, one at a time. The captured images are converted to the YUV420 format and fed to an MPEG encoder. Next, the motion field is constructed from the motion vectors obtained from the motion estimation module. The potential raw motion vectors are projected onto a virtual square that is used to determine the robot motion based on the majority vote method. As a result, a selected outlier rejection technique is employed in accordance with the resulting robot motion. Finally, the robot travel distance or rotation angle can be calculated from the valid motion vectors.

Furthermore, this low-cost motion field detector may be used in detecting a precipice while the robot is moving forward. A filtered motion field from the above technique is divided into 3 rows with 5 patches each. The detection stages of WATCHING, WARNING, and PANIC, are applied to the top, middle, and bottom rows, respectively. If the PANIC row is activated, the robot will stop immediately to protect it from potential damage.

The main focus of this work is to leverage the implementation of the motion field creation by means of utilizing what is already available from existing technology. The MPEG encoder, by its nature, performs the motion estimation to minimize the amount of residual data in the video. One benefit from deploying the MPEG encoder in the motion field construction is complexity reduction. Additionally, the process of modifying the software encoder to export its motion vectors is fairly simple and less time-consuming than creating the motion field from scratch. But considering the fact that the MPEG encoder performs motion estimation to optimize the video compression, rather than optimizing for motion estimation, this leaves some issues to be investigated. Since the quality of the visual odometry is determined by how accurately the system can perform, the questions that need to be answered are how good is it to use the MPEG motion vectors for this work and how accurate is the resulting system?

In the experiments, the performance of the proposed real-time visual odometry system is compared to those obtained from wheel encoders. The results indicate that the visual odometry is sufficiently accurate for use in a mobile robot that has no wheel encoder. It may also be used in cooperation with wheel encoders to enhance the accuracy. Nevertheless, the performance of the visual odometry is less consistent than the wheel encoder. In addition, it relies on the degree of the visual texture present in the floor. The performance will deteriorate considerably for a floor that causes large areas of glare.

However, one missing feature in many mobile robot sensor systems is the ability to detect a precipice. It is possible to detect a precipice by using a laser range finder scanning in a vertical direction attached to the front of a mobile robot. This may be successful but is often not a cost-effective means to resolve the problem. Consequently,

this inexpensive MPEG motion field estimate is applied to develop low-cost precipice detection. This work is inspired by the pixel-based precipice detection by Campbell et al., [21]. Our experiments demonstrate highly successful results. It is believed that this is the first implementation of precipice detection employing MPEG motion vectors.

Thesis Outline

This thesis specifically investigates employing MPEG motion vectors in mobile robot navigation. Two real-time applications are created. One is to study the feasibility of whether the MPEG motion vectors will be sufficiently accurate for the visual odometry application. The other is use these obtained motion vectors to detect a precipice. The thesis organization is given as follows.

Chapter II gives a detailed background study of how digital video processing works. This is the infrastructure of the implementation of this work. The techniques of video compression, in general, will be given. The second part of the chapter then describes the MPEG standard in some detail. The key topics related to the application design and implementation are provided.

Chapter III discusses related work regarding visual motion analysis and mobile robot visual navigation. It starts with explaining methods for creating the visual motion field. Next, techniques used to recover the camera motion from the 2D motion field on the image plane are given. Then, the application of the mobile robot navigation is described.

Chapter IV describes the hardware components as building blocks of the system. The hardware composition and setup will be given. The issues of software selected to implement the system are also mentioned.

Chapter V mainly describes the system design. The proposed approaches to tackling the problems will be given. Starting from how to obtain the motion field from the MPEG encoder, the chapter then proposes a technique for implementing a real-time visual odometry application. Finally, it proposes a method of detecting a precipice.

In Chapter VI, the given hardware components and software design are implemented. The system as a whole will be depicted. Then, the graphical user interfaces (GUI) of both applications are shown. The setup parameters, which are important to the system performance, are provided as well.

Chapter VII describes how the experiments are to be conducted. The application setup prior to the experiments will be explained. The experimental results are shown in graphical and numeric formats, and the data analysis and discussion is given. The chapter concludes by describing particular useful observations obtained during the experiments.

Chapter VIII gives the conclusions of this thesis. At the end, it discusses future work that may be extended from this work.

CHAPTER II

DIGITAL VIDEO PROCESSING AND MPEG TECHNOLOGY

Digital Video Signal

A digital video signal [27] is composed of a series of still images, representing a natural visual scene, sampled spatially and temporally as shown in Figure 2.1. A scene may be represented as a frame or field, where a field may be classified as top (odd) or bottom (even) field in a particular frame. (Fields are also used with interlaced sampling as shown in Figure 2.2.) Each subsequent frame is then sampled at intervals to produce a moving video signal.

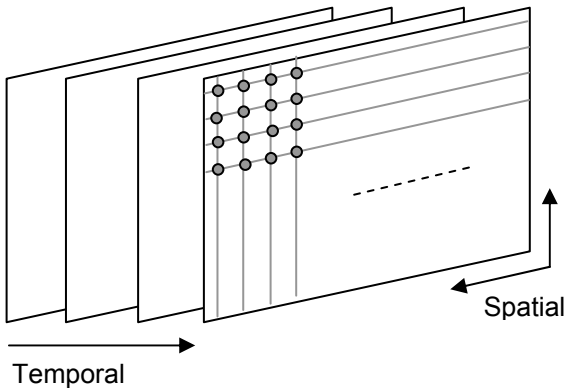


Figure 2.1: Spatial and temporal sampling of a video sequence.

Spatial Sampling and Temporal Sampling

Each sampled pixel on an image occurs at each of the intersection points on the grid. The most common format for a sampled image is a rectangle with the sampling points positioned on a square or rectangular grid. The image quality varies as the number of sampling points/rates. A higher spatial sampling rate produces a higher quality image. Each image/frame in a moving video signal is taken at a periodic time interval measured by the number of frames per second. A higher temporal sampling rate produces a smoother playback. Typically, frame rates below 10 frames/second are used for very low bit-rate video communications. However, the motion in the playback process is clearly jerky. The frame rates between 10 and 20 are more typical for low bit-rate video communications. The motion is smoother but may suffer from a little jerkiness in fast-motion scenes. Sampling at 25 or 30 complete frames per second is standard for television pictures (PAL [28] utilizes 30 frames/second, while NTSC [29] utilizes 25 frames/second).

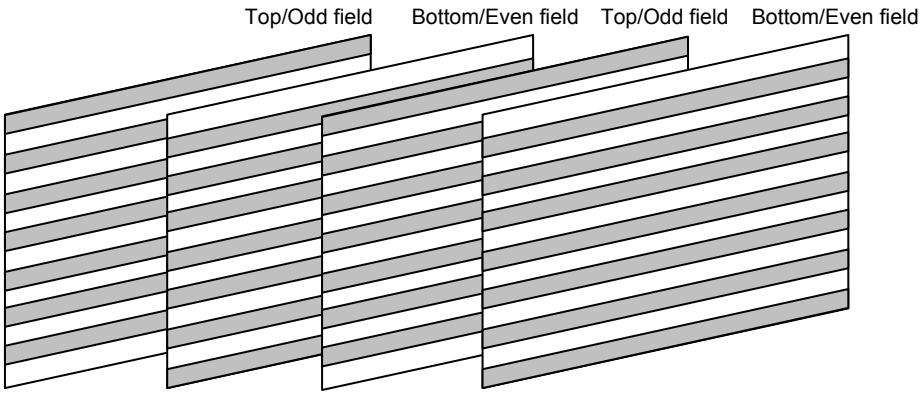


Figure 2.2: Interlaced video sequence.

Progressive and Interlaced Sampling

A video signal may be sampled using *progressive* sampling or *interlaced* sampling. Progressive sampling uses complete frames in a video sequence, while interlaced sampling divides each complete frame into two fields: odd (Top) field and even (bottom) field as indicated by the scan-lines as shown in Figure 2.2. The interlaced sampling method can send twice as many fields per second as the number of frames in an equivalent progressive sequence at the same data rate. As a result, it gives smoother motion in the playback process.

Color Spaces

A color image requires at least three components to accurately represent colors. A color space method describes the representation of brightness (luminance or luma) and color (chrominance or chroma). Commonly used color spaces include *RGB*, *YCrCb*, and *YCrCb* sampling formats.

RGB

In the RGB color space, any color can be created by the combination of three additive primary colors of light: Red, Green, and Blue. Therefore, an RGB image is typically composed of red, green, and blue components of varying shading. A typical application uses 8 bits/color or 2^{24} unique colors, providing over 16 million colors.

YCrCb

The YCrCb color space, also known as *YUV*, separates color image into two major components (luminance and chrominance) and can be derived from the RGB color

space. The luminance component (Y) represents the brightness of each pixel in the image, which can be computed as a weighted average of R, G and B:

$$Y = krR + kgG + kbB \quad \text{Eq (2.1)}$$

Where the *k*'s are weighting factors. These values may vary according to various standards.

The chrominance, on the other hand, represents the color difference components. Each component is the difference between R, G, or B and the luminance Y as described by:

$$C_b = B - Y \quad \text{Eq (2.2)}$$

$$C_r = R - Y \quad \text{Eq (2.3)}$$

$$C_g = G - Y \quad \text{Eq (2.4)}$$

YCrCb Sampling Formats

The YCrCb sampling format is the extension of YCrCb by adding the sampling format as the indication of the resolution for each component. Since human eyes are less sensitive to the chrominance than the luminance, some coding systems may choose to encode the Y component at a higher rate than Cr and Cb.

There are three sampling formats used in this color space: 4:4:4, 4:2:2, and 4:2:0. Format 4:4:4 indicates that all three components have the same resolution. In the 4:2:2

sampling format (sometimes referred to as *YUY2*), the chrominance components have the same vertical resolution as the luminance but half the horizontal resolution. The 4:2:2 video format is usually used for high quality color reproduction. The 4:2:0 sampling format, sometimes called "YV12", seems to be the most popular format. It is used widely in video conferencing, Video CD (VCD), digital television, and DVD. The chrominance components are sub-sampled by 2 in both directions. Thus, the number of samples for each color component contains only a quarter of the samples in the Y component and it requires only half the samples of the 4:4:4 video format.

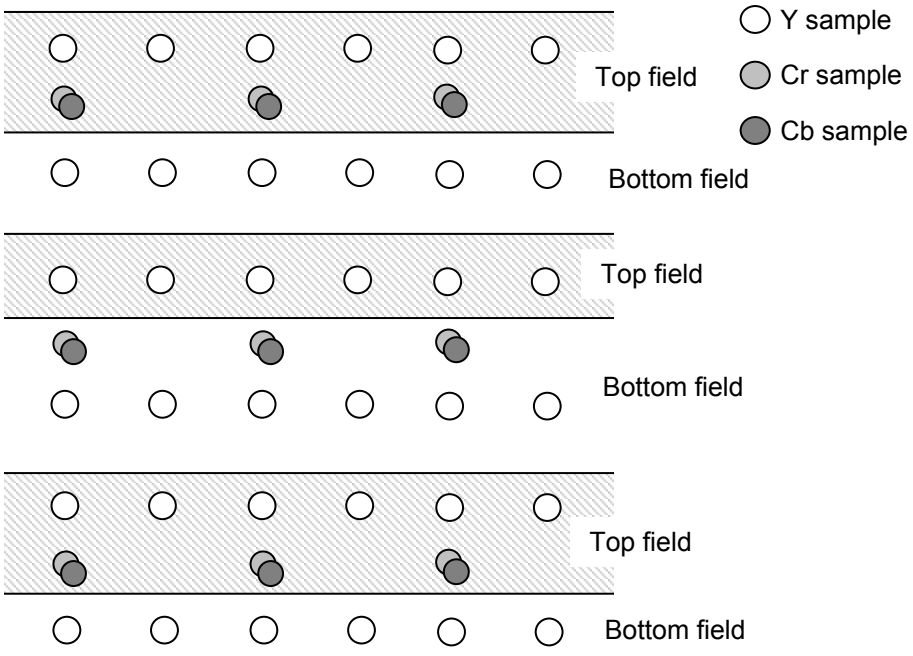


Figure 2.3: Allocation of 4:2:0 samples to top and bottom fields [27].

Video Coding

Typically, most video signals are composed of similar frames, which were captured at a series of consecutive times. Most consecutive frames are temporally correlated. In each of those succeeding frames, the neighboring pixels are usually spatially similar to each other. As a result, most video coding methods exploit both redundancies in order to reduce the amount of video information needed for transmission or storage.

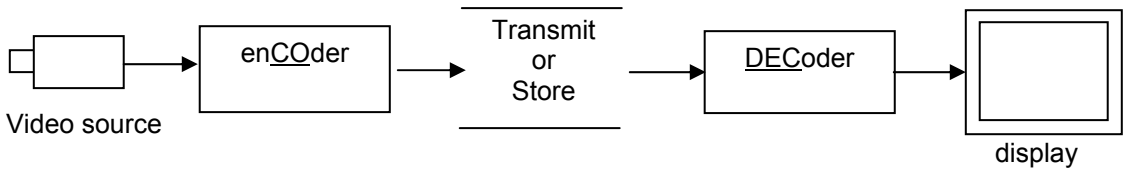


Figure 2.4: Video encoder/decoder.

From Figure 2.4, a video encoder encodes a source image or video sequence into a compressed form and either transmits it directly to the decoder or stores it in a video storage medium. The decoder, then, decodes this compressed data to produce a copy or an approximation of the source sequence. If the decoded video sequence is identical to the original, then the coding process is called "*lossless"*; if the decoded sequence differs from the original, the process is called "*lossy"*.

Note: The video encoder/decoder pair is often described as a CODEC(enCOder/ DECoder).

Temporal Model

As mentioned earlier, most frame sequences are highly correlated. Therefore, the primary goal of a temporal model is to reduce the redundant information between consecutive frames. A simple ideal is to create a predicted current frame by subtracting from its raw data the reference frame (previous frame, future frame, or both). This yields a difference frame, also called the "*residual*" frame. This residual frame, which actually has less information, will be encoded and sent to the decoding system at the receiver. The decoder uses this residual frame and reference frame to create a predicted current frame.

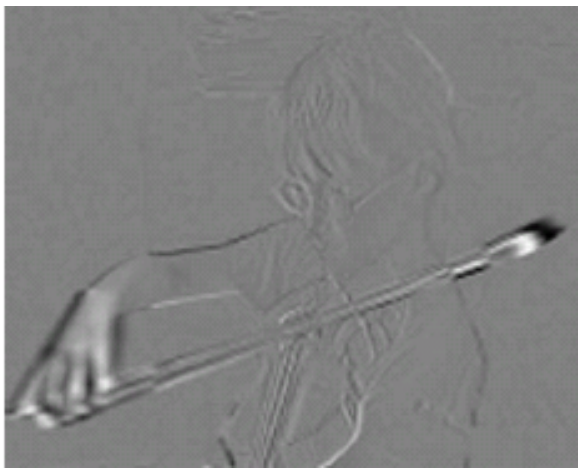
Prediction from the Previous Frame

This method employs the previous frame as a reference to produce the current predicted frame. The residual frame, as shown in Figure 2.5c, is created by subtracting the previous frame (Figure 2.5a) from the current frame (Figure 2.5b). From Figure 2.5c, however, the residual frame still contains a lot of information to be compressed. (The difference between two frames is indicated by light and dark grays). A better prediction for the residual frame is to compensate for the motions occurring between the two frames. The differences may occur from object motions such as rigid object motion (a moving car), deformable object motion (a rotating asymmetric object), camera motion (pan, tilt, zoom, rotation), uncovered region (occluded object), or lighting changes. Except for the last two factors, a difference frame may be modeled as the movements of the pixels in the video frame. Consequently, it is possible to estimate the trajectories between those two consecutive frames. Then, we can create a frame/field of trajectory pixels, which is also known as "*optical flow"*. Figure 2.6 shows the optical flow from

Figure 2.5a and 2.5b. Nevertheless, accurate optical flow estimation requires extensive computation, since it may be required to compute the trajectories for all pixels in the frame. As a result, there is still a huge amount of information to be sent to the decoder.

a)

b)



c)

Figure 2.5: a) Frame 1, b) Frame 2, c) Residual (no motion compensation).

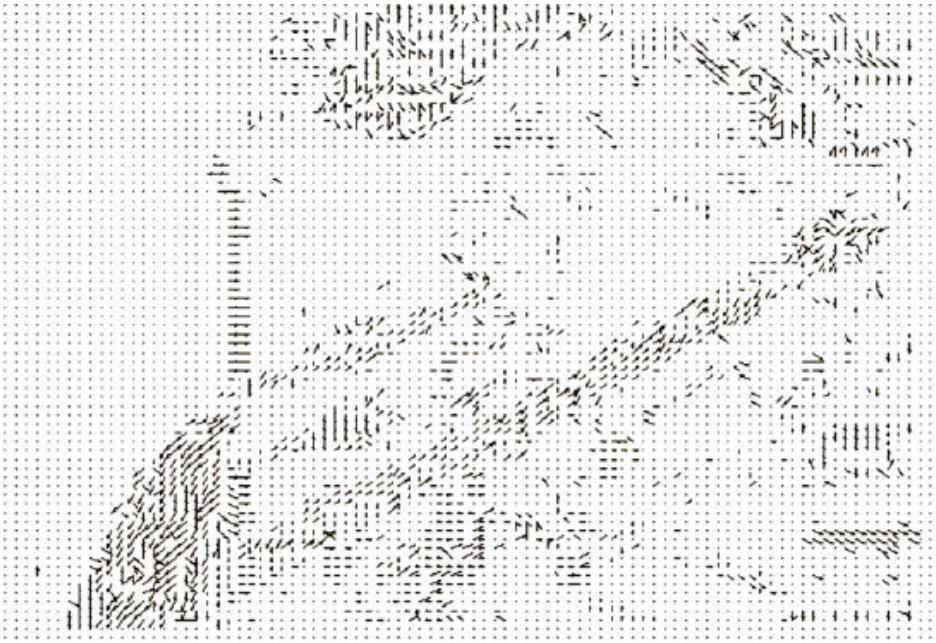


Figure2.6: Optical flow.

Block-Based Motion Estimation and Compensation

A widely used method of motion compensation is to divide a current frame into blocks of MxN pixels and compensate for movement in those blocks. This can simply be done by searching in the reference frame for the best matching block in the search area. Usually, the best match is considered as the candidate region that gives the minimum residual energy. This process of finding the best match is known as *motion estimation*. Examples of a number of matching criteria can be found at [30] [31] [32]. The chosen region is then subtracted from the current block to form a residual MxN block (motion compensation).

Since the motion is considered as a block instead of a pixel, block-based motion compensation can significantly reduce the residual energy in the predicted frame. Realistically, most objects are not rectangular. Several types of objects, such as deformable objects, rotation and warping, complex motion, are hard to compensate by using the block-based method. However, block-based motion compensation remains the most common temporal model used by all current video coding standards including MPEG-1, 2, 4 and the H26x series.

The most popular video coding systems, such as MPEG-1 [33], MPEG-2 [34], MPEG-4 Visual [35], H.261 [36], H.263 [37] and H.264 [38], employ a block of 16x16 pixels, called a *macroblock*. An example of the video 4:2:0 format is shown in Figure 2.7. The luminance is composed of four 8x8-sampled blocks. The chrominance components, blue and red, are sub-sampled and composed of one 8x8 block.

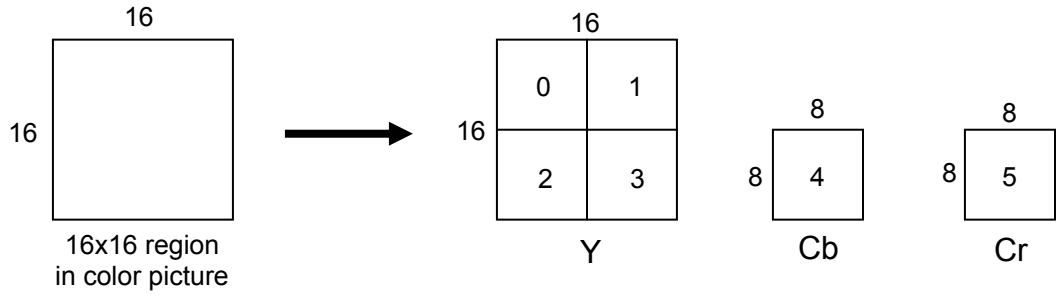


Figure 2.7: Macroblock (4:2:0).

The macroblock motion estimation is performed by finding the best match of a 16x16-pixel region in the search area as shown in Figure 2.8. The residual macroblock is

created by subtracting the selected matching region from the current macroblock. The residual is then encoded and sent to the decoder along with its motion vector (*d)*, the offset from the current macroblock to the reference region. An example of motion vectors from 16x16-block frames is displayed in Figure 2.9. It is obvious that the amount of information contained in the frame is significantly smaller than the ones represented by the optical flow as shown in Figure 2.6

If there is a major difference between the reference and the current frames, some macroblocks may be coded without motion compensation, the *intra* mode, and some may be coded with compensation, the *inter* mode. This can be done on a macroblock-to-macroblock basis, depending primarily on the contents in each macroblock. If the difference is too substantial, the whole frame may have to be coded without motion estimation and compensation at all. This kind of frame is called an "*intra frame*", and is called an "*inter frame*", otherwise. In some cases, the reference frame may be selected from the previous frame, the next future frame, or both. Therefore, it is possible that the future predicted frame may have to be encoded before the current predicted frame.

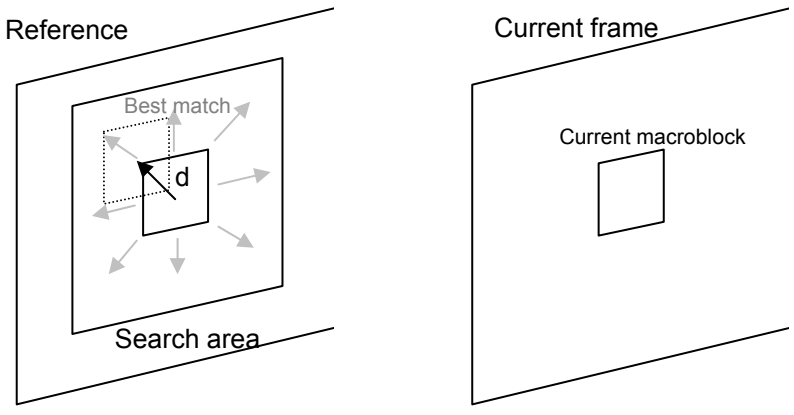


Figure 2.8: Macroblock motion estimation.

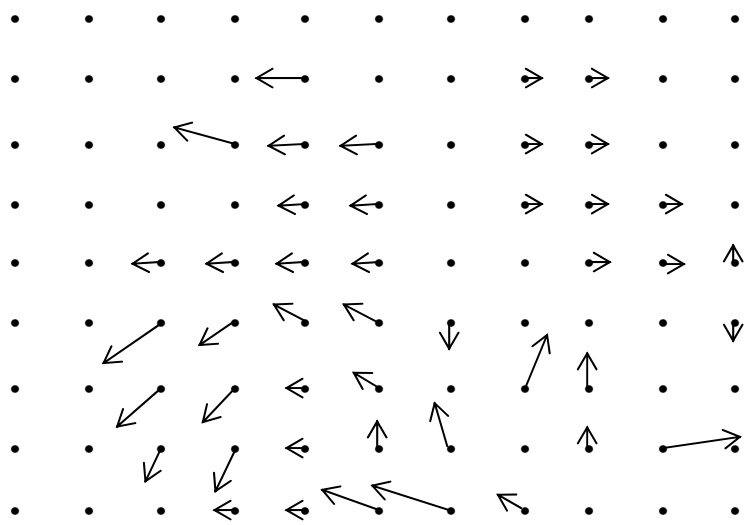


Figure 2.9: Motion vectors of a 16 x 16 block frame.

Image Model

Most images have high correlation in neighboring samples. It is hard to compress them in their original form, thus the primary function of the image model is to decorrelate the image or the residual data further so that it can be converted into a form that can be efficiently compressed using an entropy coder [39].

In order to achieve more efficient compression, practical image models typically perform three main operations: *transformation* (decorrelates and compacts the data), *quantization* (reduces the precision of the transformed data), and *reordering* (arranges the data to group together significant values).

Predictive Image Coding

A main function of predictive image coding is to compare the current component with the previous or future component(s). Then it computes the difference/residual between the current and the reference components. The goal is to reduce the number of information in the residual component. The more accurate the prediction, the less energy that is left to be compressed. Motion compensation is an example of predictive coding. In this scheme, a region of a current frame is predicted according to the previous frame. The selected region is then subtracted from the current region to produce a motion-compensated residual.

Predictive coding is also used to predict an image sample, or region, based on the previously-coded samples or region within the same image or frame. This is also used for Intra coding in H.264 [40]. The spatial prediction is sometimes called "*Differential Pulse Code Modulation*" (DPCM), which is named after a method of differentially encoding PCM samples used in telecommunications system.

Transform Coding

The primary goal of this procedure is to convert image or motion-compensated residual data into another domain (the transform domain). Typical tranforms in image and video compression fall into *block-based* and *image-based* transforms. Currently, the most common of the block-based transforms is the *Discrete Cosine Transform* (DCT). In the DCT, an image is divided into, and operated on, NxN blocks. Its advantages are that it requires low memory for its operation, and fits into the compression of block-based motion compensation. On the other hand, the most common image-based transform is the

Discrete Wavelet Transform (DWT). The DWT operates on the entire image or frame, known as the *tile*. Even though it performs better on still images than the DCT does, it is not suitable for video compression, because it requires more memory and it does not fit block-based motion compensation. Therefore, the DWT will not be discussed in this paper.

Discrete Cosine Transform (DCT)

The DCT operates on X, an NxN block, and produces Y, an NxN block of coefficients, as its output. This process is invertible. The forward DCT (FDCT) and inverse DCT (IDCT) are given in (2.5) and (2.6) respectively.

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^{T} \qquad \text{Eq (2.5)}$$

$$\mathbf{X} = \mathbf{A}^{T}\mathbf{Y}\mathbf{A} \qquad \text{Eq (2.6)}$$

Where X is a matrix of sampled pixels

Y is a matrix of coefficients

A is a transform matrix

A^T is a transpose of matrix A.

The elements of A are given by:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} \quad \text{where} \quad C_i = \sqrt{\frac{1}{N}} \quad (i=0) \quad C_i = \sqrt{\frac{2}{N}} \quad (i>0) \qquad \text{Eq (2.7)}$$

Then, equations 2.5 and 2.6 can be written in summation form as:

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \quad \text{Eq (2.8)}$$

$$X_{ij} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \quad \text{Eq (2.9)}$$

An example of a 4x4 transform matrix A is given in 2.10:

$$A = \begin{bmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{2\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{6\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{10\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{14\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{9\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{15\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{21\pi}{8}\right) \end{bmatrix} \quad \text{Eq (2.10)}$$

The cosine function is symmetrical and repeats after 2π radians and hence A can be simplified to:

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & -\sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \end{bmatrix} \quad \text{Eq (2.11)}$$

or

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & -b & c \end{bmatrix} \qquad \text{Eq (2.12)}$$

where $a = \frac{1}{2}$, $b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right)$, $c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right)$

Evaluating the cosine gives:

$$A = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.271 & -0.271 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.271 & -0.653 & -0.653 & 0.271 \end{bmatrix} \qquad \text{Eq (2.13)}$$

These coefficients in the DCT domain can be considered as "*weights*" of a set of standard *basic patterns*. The basic pattern for the 4x4 and 8x8 DCT's are given in Figure 2.10 and Figure 2.11 respectively and are composed of combinations of horizontal and vertical cosine functions. Any image block may be reconstructed by combining all NxN basis patterns, with each basic pattern multiplied by the appropriate weighting factor (coefficient).

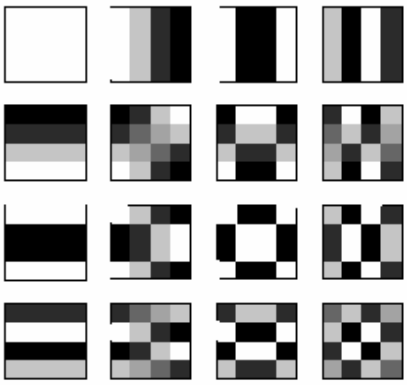


Figure 2.10: 4x4 DCT basis patterns [27].

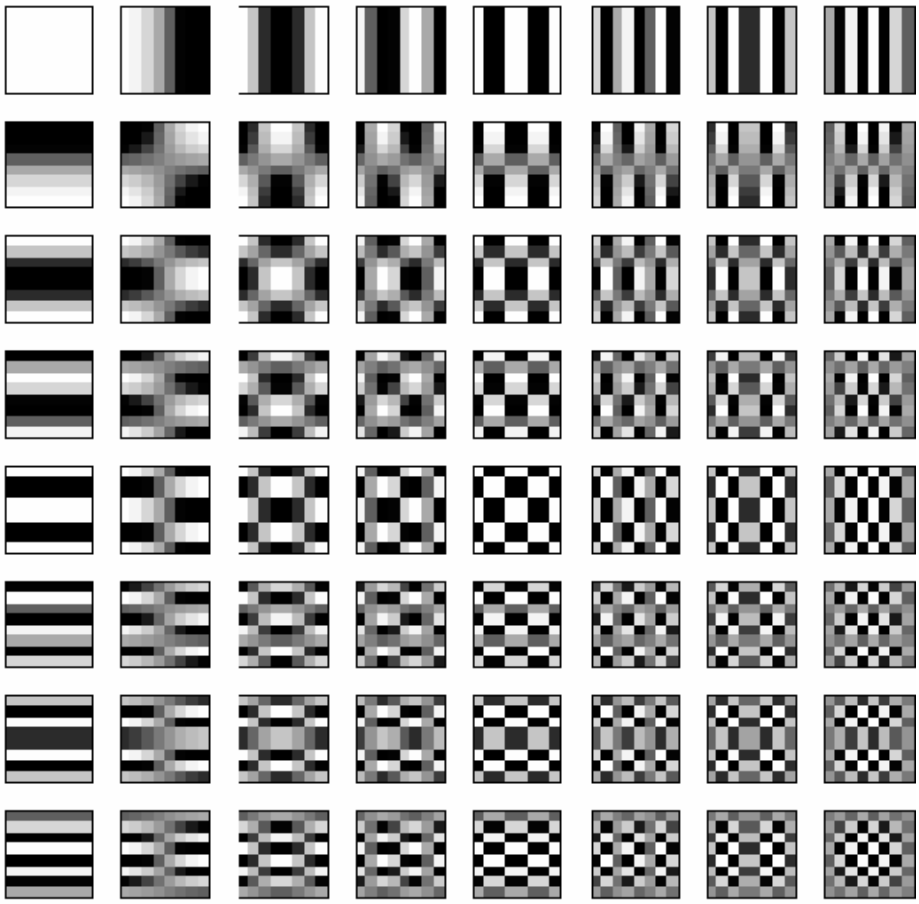


Figure 2.11: 8x8 DCT basis patterns [27].

Quantization

This process is a way to represent a quantized signal with fewer bits than the original signal. It can be done by reducing the range of input values to a smaller range in the output. The process falls into two main categories, *scalar* quantization and *vector* quantization.

Scalar Quantization

Scalar quantization maps one value of input to one quantized value of output. An example is to round a floating-point number to the nearest integer. The process is "lossy"

since the rounded integer cannot be inverted to its original value. A simple example of a uniform quantizer is given as follows:

$$FQ = round\left(\frac{X}{QP}\right) \quad \text{Eq (2.14)}$$

$$Y = QP \cdot FQ \quad \text{Eq (2.15)}$$

$$Y = QP \cdot round\left(\frac{X}{QP}\right) \quad \text{Eq (2.16)}$$

$$FQ = round(\boldsymbol{X}/QP) \quad \text{Eq (2.17)}$$

$$\boldsymbol{Y} = QP.FQ \quad \text{Eq (2.18)}$$

$$\boldsymbol{Y} = QP.round(\boldsymbol{X}/QP) \quad \text{Eq (2.19)}$$

where *QP is a quantization, step size.*

The quantized output values, Y, are spaced at uniform intervals of QP. The essential parameter to consider is the step size QP. If the step size is large, the forward quantizer (FQ) produces small quantized values. The compression efficiency then becomes high but the inverse quantizer (IQ) produces output values with a crude approximation (higher error). Hence, the output quality is lower. On the contrary, if the step size is small, the de-quantized values in the inverse quantizer will be closer to the original values, producing higher quality outputs, but the compression efficiency becomes lower.

Figure 2.12 depicts two samples of scalar quantizers: *linear* and *nonlinear* quantizers. The linear quantizer maps linear inputs to linear output, whereas the nonlinear quantizer maps small values in the input to zero. The big gap near the zero value in the output is called the "*dead zone*". The video encoder utilizes the forward quantization to reduce the precision of the image data. This is utilized by mapping insignificant coefficients after DCT transformation to zero, while retaining the significant coefficients outside the dead zone.

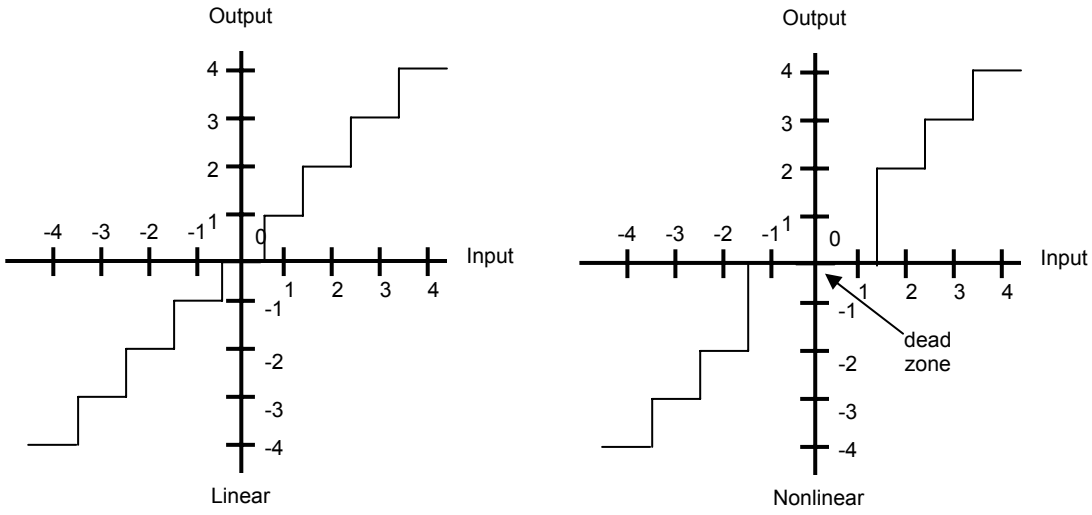


Figure 2.12: Scalar quantizers: linear, nonlinear with dead zone.

Vector Quantization

Vector quantization maps a group of input values, such as a block in the image, to a single value, the codeword (typically an integer). A simple idea for understanding vector quantization is that both the encoder and decoder have a set of vectors stored in a codebook. The encoder divides the image into MxN blocks and finds a vector that best matches an input block. Only the vector index is transmitted to the decoder. The decoder,

on the other hand, uses the index to look up the vector in its codebook and outputs the result as illustrated in Figure 2.13.

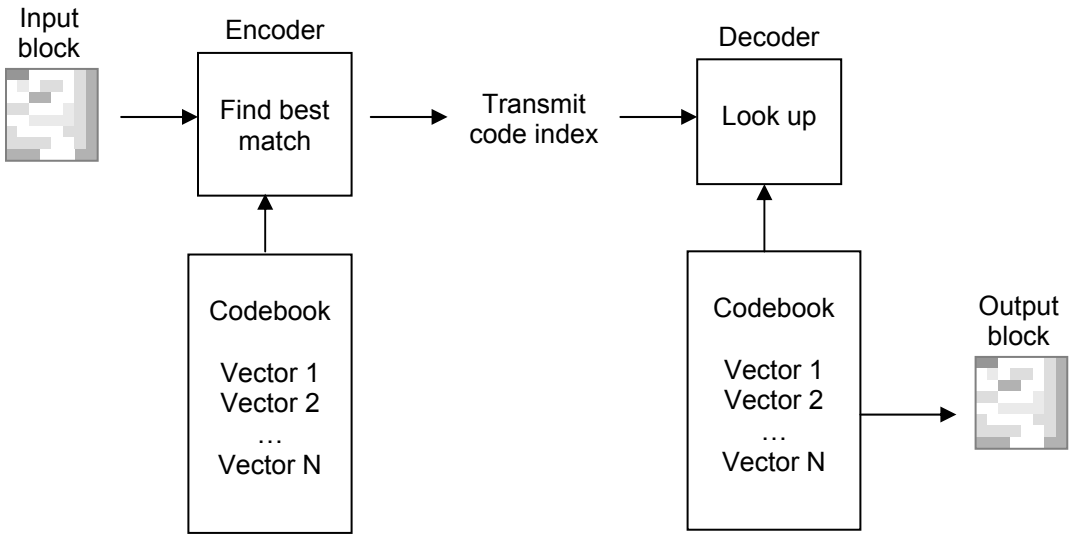


Figure 2.13: Vector quantization.

Reordering and Zero Encoding

After the DCT coefficients have been quantized, the majority of the quantized coefficients become zero. However, the data order is not quite uniform. The reordering process groups non-zero coefficients together, so that entropy encoding, which will be described later in this chapter, can perform more efficiently.

DCT Coefficient Distribution

A transformed block of image or residual samples typically contains low frequencies. Its non-zero coefficients are generally clustered around DC (0, 0) at the upper left corner as shown in Figure 2.14. The distribution is fairly symmetrical in both

the horizontal and vertical directions. In the residual field as shown in Figure 2.15, the non-zero coefficients are also clustered around the DC position but are skewed. Figure 2.16 illustrates the distribution of a field of coefficients, where most non-zero coefficients occur on the left hand side of the plot. Since the picture is subsampled in the vertical direction, the vertical axis then contains strong high-frequency components resulting in larger DCT coefficient in the vertical direction.

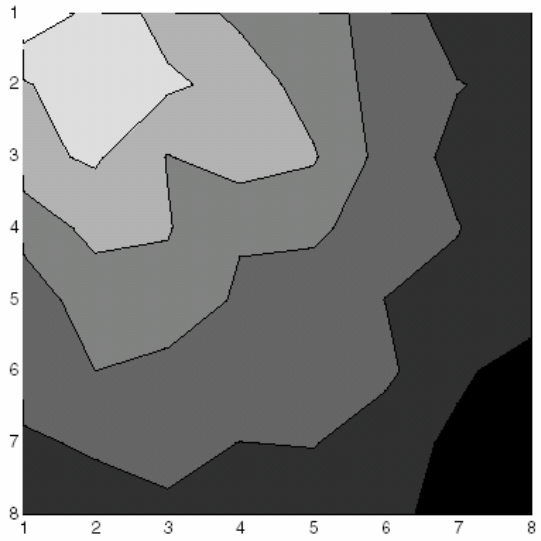


Figure 2.14: 8x8 DCT coefficient distribution of the image in Figure 2.5c (frame).

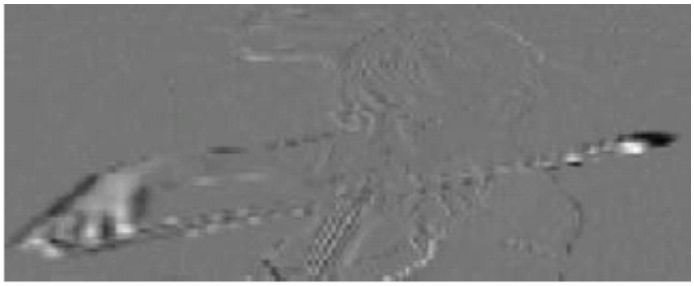


Figure 2.15: Residual field picture.

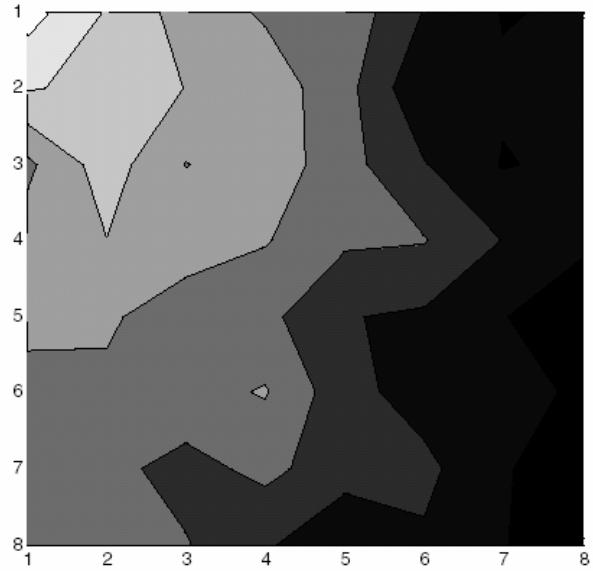


Figure 2.16: 8x8 DCT coefficient distribution (field).

Scanning DCT Coefficients

As mentioned above, most of the non-zero coefficients in the frame distribution concentrate around the DC position, leaving the zero coefficients lying outside. Hence, we can take advantage of this scenario by reordering the DCT coefficients in such a way that the non-zero values are grouped together. One common method is by zigzag scanning the order of coefficients starting from the DC coefficient as displayed in Figure 2.17. However, this zigzag scan is not applicable to the field coefficients because of its skewed distribution. The scan order must be modified to suit the skewed distribution as shown in Figure 2.18.

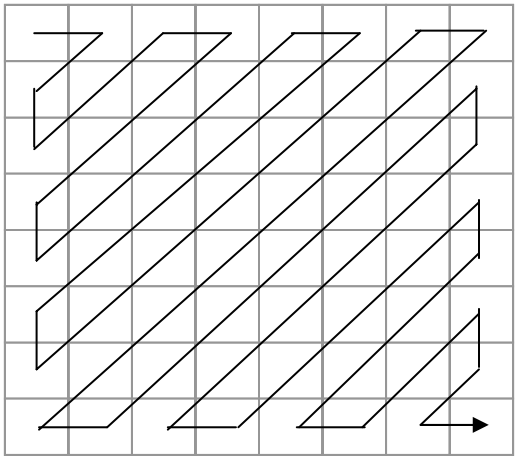


Figure 2.17: Zigzag scan order (frame block).

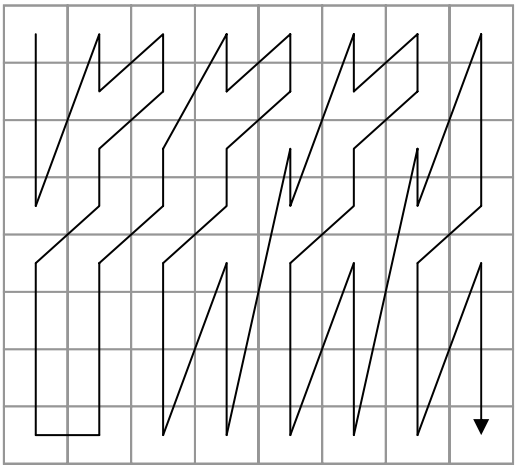


Figure 2.18: Zigzag scan order (field block).

Run-Level Encoding

After quantization and reordering, high-frequency coefficients are typically quantized to zero and grouped together. Thus, the output of the reordered coefficients is

an array of one or more clusters of nonzero coefficients followed by a number of zero coefficients. A number of consecutive zeros can be represented as a series of (run, level) pairs, where *run* indicates the number of zeros preceding a nonzero coefficient and *level* indicates the magnitude of the nonzero coefficient. For example, an input array containing [16,0,0,-3,5,6,0,0,0,0,-7....] can be represented as (0,16),(2,-3),(0,5),(0,6),(4,-7)..... Each run-level pair is then encoded as separate symbol by the entropy encoder.

In some cases, a special code symbol, *last*, is required to indicate the final nonzero coefficient in the input array. This "Two-dimensional" (run, level) encoding is used to encode each run-level pair as usual and a *last* is encoded to indicate the end of nonzero coefficients. However, if "Three-dimensional" run-level encoding is used, each symbol encodes three quantities, *run*, *level*, and *last*. From the example above, if –7 is the final nonzero value, its output in 3D run-level coding can be written as (0,16,0), (2,-3,0), (0,5,0), (0,6,0), (4,-7,1), where "1" in the final code indicates the final nonzero coefficient in the array input.

Entropy Encoder

The entropy encoder converts a series of symbols representing elements of the video sequence into a compressed bitstream, which makes it more suitable for transmission or storage. The input symbols include quantized transform coefficients, motion vectors, resynchronization marker, headers of macroblock, picture, sequence, and some other supplementary information. This section will discuss the methods of predictive pre-coding followed by two entropy coding techniques: Huffman variable length codes and arithmetic coding.

Predictive Coding

The data in a frame such as blocks, macroblocks, and motion vectors seem to be correlated to their neighbors. The DC values in adjacent intra-coded block as well as the neighboring motion vectors may be very similar to each other as well. Hence, the coding efficiency may be improved by predicting the elements in the block or macroblock with the previously encoded one. Then, only the different value is left to be encoded. In case of a large object moving or camera pan, for instance, the adjacent macroblocks in the moving area are likely to have similar displacement. A vector for the current macroblock may be predicted from the previous coded value. Then, the difference between the predicted and actual motion vector (*Motion Vector Difference* or MVD) is encoded and transmitted.

Variable-Length Coding

Variable-length coding is a way to compress data by mapping input symbols into a shorter form of codewords (variable-length codes, VLCs).

Huffman Coding

Huffman coding [41] constructs a set of VLCs according to the probability of the occurrence of each symbol. The resulting VLCs are then assigned to each symbol. Typically, the most commonly occurring symbol contains the shortest codeword.

The procedure of Huffman coding starts with creating a probability table and generating a Huffman code tree. Then the encoding and decoding process can be obtained by using the Huffman code tree. The example of Huffman coding given below

demonstrates how to encode and decode a set of motion vector differences (MVD) for a video sequence. Table 2.1 lists the probability of the motion vectors in the video sequence. $Log_2(1/p)$ represents the number of bits utilized to achieve optimum compression. Figure 2.19, displays the Huffman code tree with the binary symbols (0,1) associated with each edge. The leaves of the binary tree represented by square boxes are the MVD values at which each leaf is mapped to a VLC. Each VLC can be found by traversing the tree from the root to the leaf, the target value. For example, the series of vectors (1, 0, -2) would be transmitted as the binary sequence 0111000. Table 2.2 lists all codes generated from the tree. The decoding process is very straightforward. Given the Huffman tree and a sequence of encoded binary, each uniquely-decodeable code is converted back to the original data, for example: 011 is decoded as (1), 1 is decoded as (0), and 000 is decoded as (-2).

Table 2.1: Probability of occurrence of sample motion vectors.

Vector	**Probability *p***	**log2(1/*p*)**
-2	0.1	3.32
-1	0.2	2.32
0	0.4	1.32
1	0.2	2.32
2	0.1	3.32

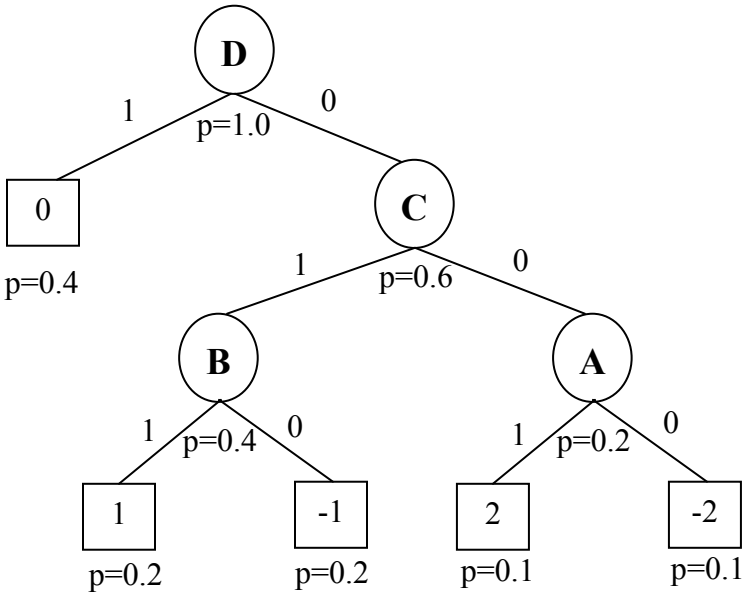


Figure 2.19: Huffman code tree for sample motion vectors.

Table 2.2: Huffman codes for sample motion vectors.

Vector	Code	Bits (actual)	Bits (ideal)
0	1	1	1.32
1	011	3	2.32
-1	010	3	2.32
2	001	3	3.32
-2	000	3	3.32

In the MPEG encoder, an MVD is encoded as a pair of VLCs, one for the *x* component and another for the *y* component. However, one major disadvantage of Huffman-based codes is that they are sensitive to transmission errors. If an error occurs in the VLC bitstream, it can cause the decoder to lose synchronization and fail to decode the

following codes correctly. The error may also spread to the decoded sequence. Reversible VLCs (RVLCs) [42] that can be successfully decoded in either the forward or backward direction can resolve this problem and prevent error propagation.

The Hybrid DPCM/DCT Video CODEC Model

Most major video coding standards, such as H.261 [36], H.263 [37], MPEG-1 [33], MPEG-2 [34], MPEG-4 Visual [35] and H.264 [38], employ the same generic model, which includes motion estimation and compensation (also known as DPCM), transform coding, and entropy coding. This model is described as a hybrid DPCM/DCT CODEC. The block diagrams of the video encoder and decoder are shown in Figures 2.20 and 2.21 respectively. The encoder predicts the current frame F_n using frame F'_{n-1} as a reference to produce the coded bitstream, whereas the decoder conversely decompresses the coded bitstream to reconstruct the current frame F'_n for a playback.

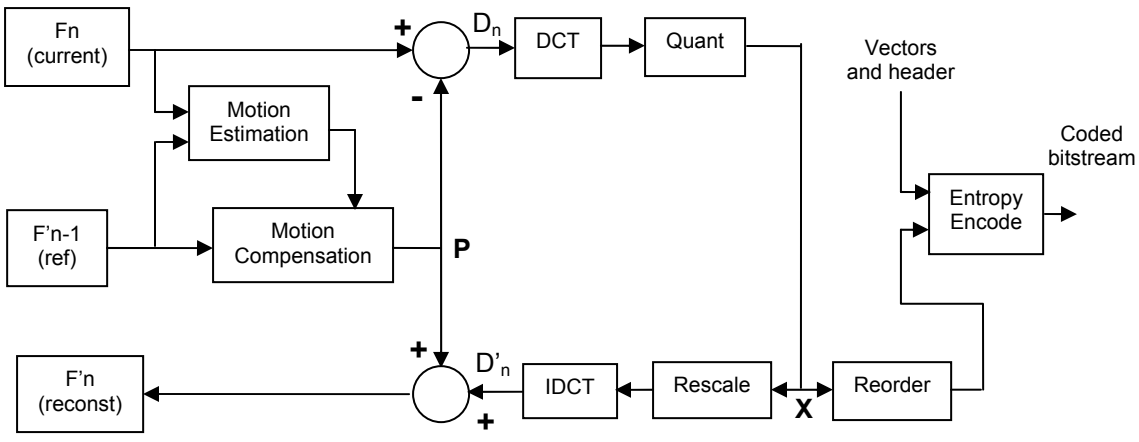


Figure 2.20: DPCM/DCT video encoder.

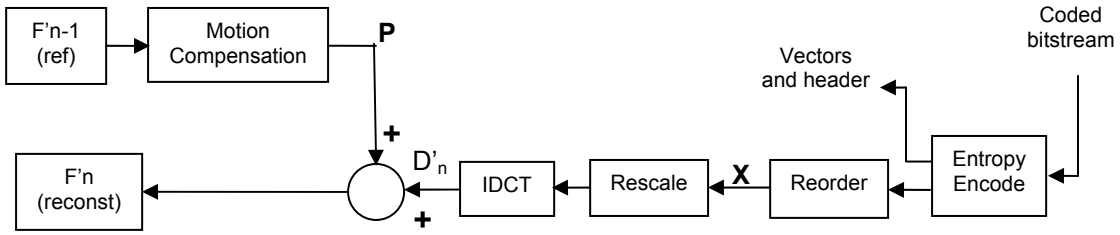


Figure 2.21: DPCM/DCT video decoder.

The video encoder in Figure 2.20 is divided into two parts according to the flow of the signals. The flow of left to right represents the encoding part, while the flow of right to left represents the reconstruction part. In the encoding operation, the input frame F_n is processed in a macroblock unit. The motion estimation uses the previous reconstructed frame F'_{n-1} as a reference to find the best matching region and the offset (motion vector) for the current macroblock. Then, a motion compensated prediction **P** is created. After that, **P** is subtracted from the current macroblock to generate a residual $\mathbf{D_n}$. $\mathbf{D_n}$ is typically divided into four 8x8 blocks and each block is transformed using DCT. The coefficients from the DCT are quantized, reordered, and run-level coded. Finally, the coded coefficient, motion vector, and associated header information of the current macroblock are entropy encoded to produce the coded bitstream.

The decoder in Figure 2.21 simply reverses the encoding procedure in the encoder part. First, it entropy decodes the coded bitstream to extract the macroblock header, motion vector, and the coded coefficients from the bitstream. The coefficients are then reorder and rescaled (dequantized). After that, the decoded residual $\mathbf{D'_n}$ is created by the inverse DCT. At this point, the decoded motion vector is extracted and can be used to

locate the 16x16 region in the previous decoded frame F'_{n-1}. This region now becomes the motion compensated prediction **P**. Finally, **P** is added to $\mathbf{D'_n}$ to generate a reconstructed macroblock. Once this reconstruction for frame F'_n is completed, the reconstructed frame is ready for display and may be saved for further decoding in frame F'_{n+1}.

Note that, the reconstruction part in the encoder, shown in Figure 2.20, is similar to the decoder. This is to make sure that the encoder and decoder use the same reconstructed frame to generate the motion compensated prediction. This is to prevent the problem of a cumulative mismatch ("drift") between the encoder and decoder.

Figures 2.22-2.27 [27] illustrate the sample input frame F_n, reconstructed reference frame F'_{n-1}, residual frame, $F_n - F_{n-1}$, with no motion compensation, 16×16 motion vectors (superimposed on frame), Motion compensated reference frame, and motion compensated residual frame respectively.

Figure 2.22: Input frame F_n.

Figure 2.23: Reconstructed reference frame F'_{n-1}.

Figure 2.24: Residual $F_n - F'_{n-1}$ (no motion compensation).

Figure 2:25: 16x16 motion vectors (superimposed on frame).

Figure 2.26: Motion compensated reference frame.

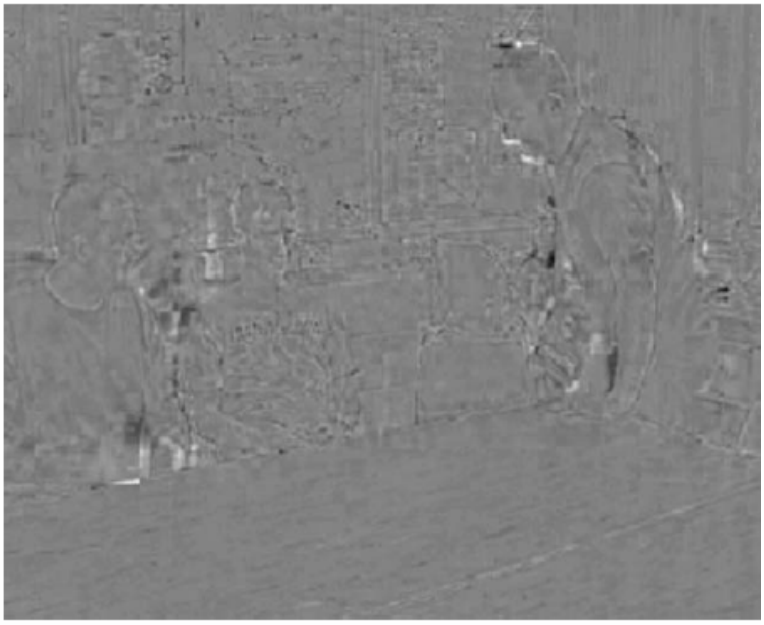


Figure: 2.27: Motion compensated residual frame.

MPEG Standard

"MPEG, which stands for *Moving Picture Experts Group*, is the name of a family of standards used for coding audio-visual information (e.g., movies, video, and music) in a digital compressed format." [43]. MPEG was first established in 1988. The definition of the video algorithm (Simulation Model 1) was completed by September 1990 and the MPEG-1 standard was approved as an international standard by late 1992. The MPEG family of standards includes MPEG-1, MPEG-2 and MPEG-4, formally known as ISO/IEC-11172 [33], ISO/IEC-13818 [34] and ISO/IEC-14496 [35] respectively. MPEG standards are mainly categorized as Systems, Video, and Audio standards. MPEG Systems specify how to multiplex MPEG Video and Audio bitstreams into a single stream. An MPEG System stream contains the proper timing information so that MPEG

players (or decoders) can playback the video and the audio properly synchronized. MPEG Audio, in particular, is a subgroup of MPEG working on all audio aspects of the MPEG standards.

MPEG-1

MPEG-1 is a generic standard in that it standardizes a syntax that supports the operations such as motion estimation, motion compensated prediction, discrete cosine transformation (DCT), quantization, and variable-length coding. In fact, the syntax does not standardize how a video sequence is to be encoded or what encoding algorithm is to be used. Instead, it allows the encoder implementers flexibility in designing the encoder. However, a number of parameters defining the coded bitstream and decoders are contained in the bitstream itself. The implementers can utilize this to pass parameters from the encoder to the decoder.

MPEG-1 provides progressive (noninterlaced) video only. Typically, the video input is converted into the MPEG Standard Input Format (SIF) with (Y, Cr, Cb) color space. The luminance component is sampled at 352x240 pixels, 30 frame/second with 8 bits per pixel (for both luma and chroma). However, the chrominance component is subsampled by 2 in both spatial directions (horizontal and vertical) as described in the previous chapter.

Data Structure and Compression Modes

The MPEG-1 bitstream is categorized as six layers of its data structure as shown in Figure 2.28.

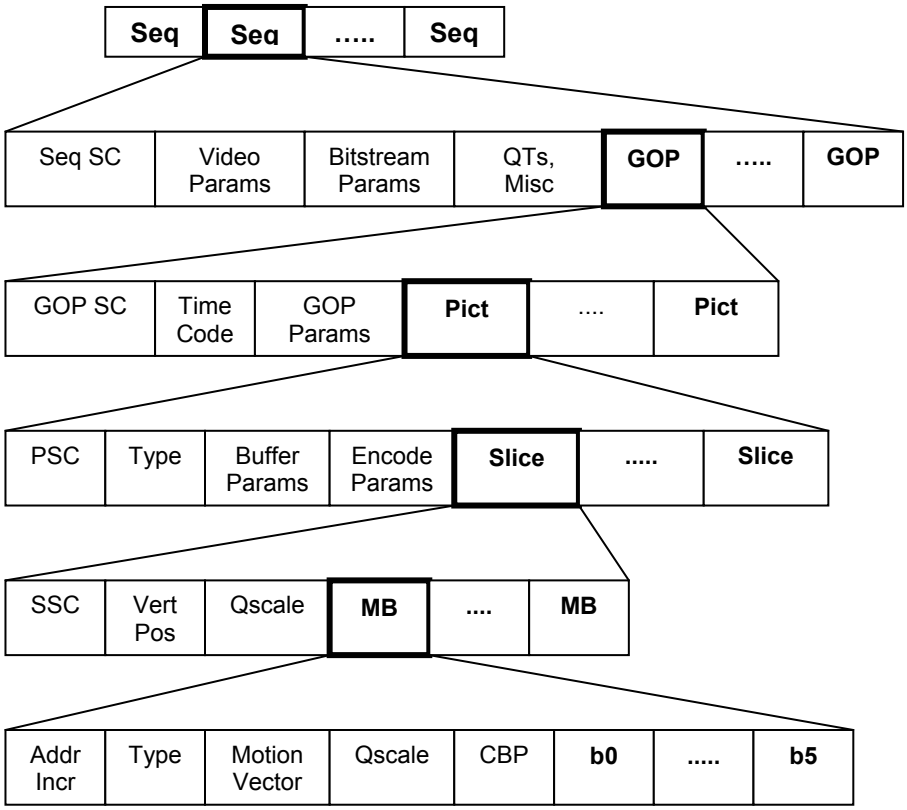


Figure 2.28: MPEG video bitstream.

Figure 2.28 indicates that one layer is formed by the intermediate layer underneath it. Each layer is described as follows:

1) *Sequences* are formed by several groups of pictures.

2) *Group of Pictures* (GOP) are made up of pictures.

3) *Pictures* are composed of slices. There are four types of pictures in MPEG-1: I-pictures, P-pictures, B-pictures, and D-pictures. I-pictures are intra-frame DCT encoded using a JPEG-like algorithm. They are also used as access points to the video sequence. P- and B-pictures are encoded as inter frames using the MC prediction algorithm. A P-picture uses a previous I- or P-frame for its forward

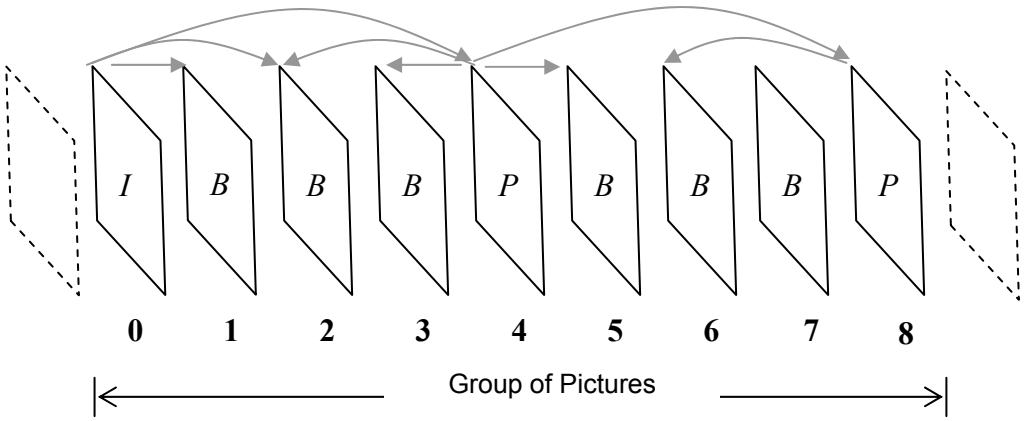
prediction, whereas prediction of a B-picture can be forward, backward, or bidirectional using previous and/or future I- or P-pictures as its reference(s). D-pictures contain only the DC component of each block. They are used for browsing purposes at very low bitrates. Since, the B-pictures may use a future frame as a reference frame for its motion-compensated predictions, each picture in a GOP, as a result, may not be coded sequentially as shown in Figure 2.29. (*The number of I-, P-, and B-frames in a GOP are application-dependent.*)

4) *Slices* are made of Macroblocks. They are mainly used in error recovery.
5) Each *Macroblock* is composed of one luminance block and two chrominance blocks (Cr and Cb).
6) *Blocks* are the lowest layer in MPEG-1 Data Struture. They are composed of 8x8 pixel arrays.

The details regarding the fields in each layer are described in Table 2.3.

Table 2.3: MPEG-1 data structure.

SEQ	*Video Params*	Width, height, aspect ratio of pixels, picture rate.
	Bitstream Params	Bit rate, buffer size, and constrained parameters flag.
	QTs	Quantization Tables for I-frames and Quantization Tables for P-frames
GOP	*Time code*	Bit field with SMPTE time code indicating hours, minutes, seconds, frame.
	GOP Params	Bits describing structure of GOP
Pict	*Type*	I, P, or B-frame.
	Buffer Params	How full decoder's buffer should be before starting decoding.
	Encode Params	Indicate whether half pixel motion vectors are used
Slice	*Vert Pos*	Line that this slice starts.
	Qscale	Quantization table scaled used in this slice.
MB	*Addr Incr*	Number of MBs to skip.
	Type	MB type and if it has a motion vector.
	Qscale	Quantization table scaled used in this MB.
	Coded Block Pattern (CBP)	Bitmap indicating which blocks are coded



The pictures in this GOP may be encoded as:
0, 4, 1, 2, 3, 8, 5, 6, 7
or
0, 1, 4, 2, 3, 8, 5, 6, 7

Figure 2.29: Group of pictures in MPEG-1.

Intra Frame Compression Mode

Since there is no temporal compression in the intra frame compression mode, the intra frame is coded without motion-compensated prediction. The macroblocks are transformed into DCT coefficients and are divided by the quantization matrix. The results are rounded to the nearest integer giving the quantized coefficients. There are two types of macroblocks in I-pictures: Intra and Intra-A. The Intra MB is coded using the default quantization as shown in Table 2.4, while the Intra-A MB is coded using MQUANT, a quantizer scale parameter, which is transmitted in the header. In MPEG, MQUANT can be varied on a macroblock-to-macroblock basis to control the bitrate for subjective quantization. The quantized DC coefficients are compressed by DPCM with a DC Huffman table giving the result of an 8-bit VLC code. The quantized AC coefficients, on

the other hand, are zigzag scanned and converted into (run, level) pairs. A single Huffman-like code table is used. There is no downloadable custom table used in MPEG-1. Only those pairs, which are highly probable, are VLC coded. The rest of them are coded with an escape symbol followed by a fixed-length code. This is to purposely avoid having very long codewords.

Table 2.4: MPEG default intra quantization matrix.

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Note: From the Table 2.4, the weight becomes larger as the frequency increases in both horizontal and vertical directions. As a result the high frequencies become diminished or eliminated after applying this quantization matrix.

Inter Frame Compression Modes

Inter frame is coded with motion-compensated prediction to reduce the temporal redundancy. MPEG-1 employs two types of temporal prediction modes: forward prediction (P-pictures) and bidirectional prediction (B-pictures).

P-Picture

The P-picture utilizes a previous I- or P-picture as a reference picture for its forward prediction. The temporal prediction of a current macroblock is given by:

$$\hat{b} = \tilde{c} \qquad \text{Eq (2.20)}$$

where:

$\hat{b}$ is the predicted macroblock in the current frame

$\tilde{c}$ is the reconstructed macroblock

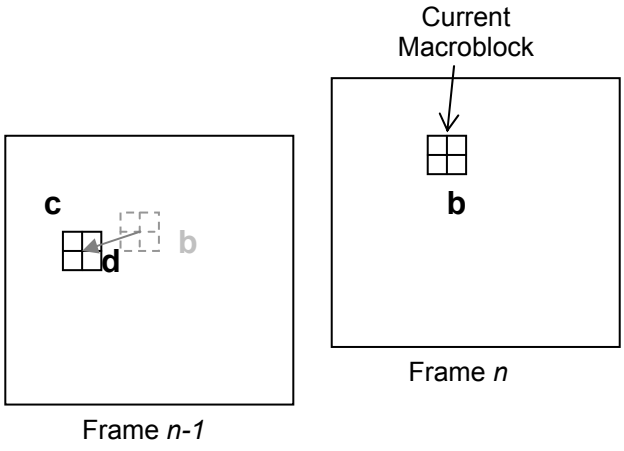


Figure 2.30: MPEG-1 forward prediction.

Figure 2.30 illustrates the prediction of a P-picture resulting in a motion vector, *d*, that indicates the displacement from the current macroblock to the reference one in the reconstructed previous frame. Then, the residuals for macroblock **b** will be computed. In the P-picture, each macroblock may be coded in one of those listed in Table 2.5. "Intra" and "Intra-A" MBs are coded independently without motion compensation (MC) as they are coded in I-pictures. MBs in the "Inter" mode family are coded with MC and/or adaptive quantization. The subscript "*D*" means that the DCT residual is coded, "*F*" means that forward MC is ON and the motion vector will be sent to the receiver, and "*A*" indicates the use of adaptive quantization (MQUANT). Therefore, the MB coded as

"Inter-FDA" indicates that the DCT coefficients, motion vector, and a new set of MQUANT are transmitted to the receiver. In certain circumstance, a macroblock may be "skipped". In this case, the MB in the previous frame can be used without coding at all.

Table 2.5: Macroblock types in MPEG-1.

I-pictures	**P-pictures**	**B-pictures**
Intra	Intra	Intra
Intra-A	Intra-A	Intra-A
	Inter-D	Inter-F
	Inter-DA	Inter-FD
	Inter-F	Inter-FDA
	Inter-FD	Inter-B
	Inter-FDA	Inter-BD
	Skipped	Inter-BDA
		Inter-I
		Inter-ID
		Inter-IDA
		Skipped

B-Picture

The B-picture, also known as bi-directional prediction, may utilize both the previous I- or P-picture and the future I- or P-picture as references for its motion-compensated prediction. The temporal prediction is given by:

$$\hat{b} = \alpha_1 \tilde{c}_1 + \alpha_2 \tilde{c}_2 \qquad \text{Eq (2.21)}$$

where $\alpha_1, \alpha_2 \in \{0, 0.5, 1\}$

$\alpha_1 + \alpha_2 = 1$

$\tilde{c}$ = reconstructed macroblock

$\alpha_1 = 1$, $\alpha_2 = 0 \Rightarrow$ forward prediction

$\alpha_1 = 0$, $\alpha_2 = 1 \Rightarrow$ backward prediction

and $\alpha_1 = 0.5$, $\alpha_2 = 0.5 \Rightarrow$ bi-directional prediction

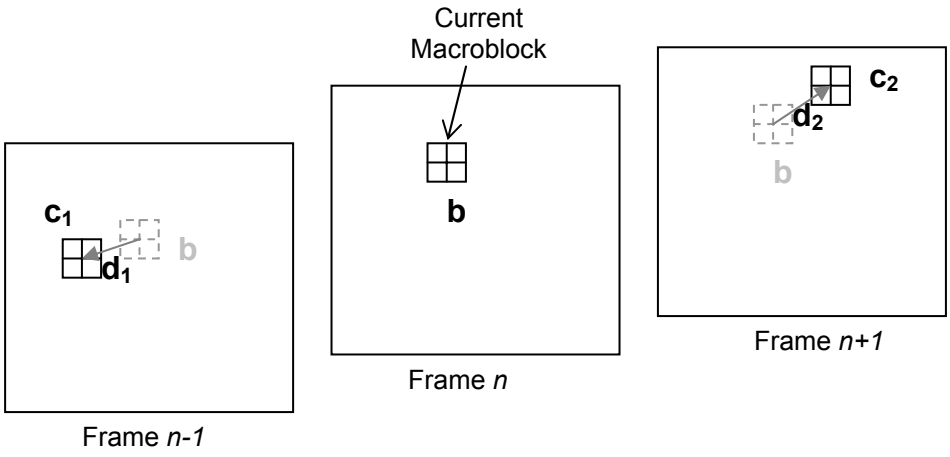


Figure 2.31: MPEG-1 bi-directional prediction.

Figure 2.31 illustrates that the prediction of B-picture results in two motion vectors, d_1 and d_2. Then, the residuals for macroblock **b** will need to be computed for each prediction. For B-picture predictions, the I- and P-pictures must be encoded first. Then, the remaining B-pictures can be interpolated from the reconstructed I- and P-pictures. This is why the bi-directional prediction is also called "*interpolative coding*".

Each macroblock in a B-picture can be coded in different compression modes as shown in Table 2.5. Two additional subscripts, "*B*" and "*I*", are introduced at which "*B*" indicates backward prediction with motion compensation, while "*I*" indicates interpolated prediction with motion compensation.

One major advantage of B-pictures is that they may be useful in handling covered or uncovered objects. For instance, if the object is about to be covered in the next frame, it can still be predicted from the previous frame and vice versa. However, its

disadvantage is that at least two frames are needed in the encoder and decoder. Using more B-pictures will lead to longer coding delay.

MPEG-1 Encoder and Decoder

An encoder consists of the following modules:

- Motion estimation
- MTYPE (selection of compression mode per macroblock)
- Setting the values of MQQUANT
- Motion compensated prediction
- Quantizer and Dequantizer
- DCT and IDCT
- Variable-length coding (VLC)
- Multiplexer,
- Buffer
- Buffer regulator

Note that the dequantizer and IDCT are needed to reconstruct a coded frame for further prediction. The IDCT used in the encoder should match the IDCT that is used in the decoder. This is to avoid the propagation of errors in prediction processes. The number of I-, P-, and B-pictures (B is optional) in a GOP is application-dependent but at least one I-picture must be included in every 132 pictures to avoid error propagation. The motion estimation algorithm, selection of MYTPE, and MQUANT are not parts of the standard. The motion estimation is performed using the luminance component only. One-half (0.5) pixel may be used for more accuracy in motion estimation. The maximum

length of motion vector can vary from picture to picture to allow maximum flexibility. Nevertheless, motion vectors that reference to the pixels outside the picture are not allowed.

On the other hand, the decoder is a reverse operation of the encoder. It demultiplexes the incoming bitstream into DCT coefficients, MTYPE, motion vectors, MQUANT, and etc. The decoder may require a buffer to store two frames in order to decode the B-pictures. An interesting breakdown of time spent in an MPEG decoder [44] is given in Table 2.6.

Table2.6: Breakdown of time in MPEG decoder.

Function	% Time
Parsing Bitstream	17.4%
IDCT	14.2%
Reconstruction	31.5%
Dithering	24.5%
Misc. Arith	9.9%
Other	2.7%

MPEG-2 Standard

MPEG-2 was intended for high quality video applications such as DVD, Digital Cable TV, Satellite TV, and HDTV. It is an extension of MPEG-1 with higher bitrates at 2-20 Mbps. Its main additional features include interlaced inputs, higher-definition inputs, alternative subsampling of chroma components. It also provides a scalable bitstream and improved quantization and coding options. The syntax is categorized into five "*profiles*": the *simple* profile, *main* profile, *SNR scalable* profile, *spatially scalable* profile, and *high* profile. In addition, each profile is divided into a number of "*levels*" to

impose constraints on some of the video parameters [45]. (More details about profiles will be described in the MPEG-4 section.) MPEG-2 allows three chroma subsampling schemes for its macroblocks as 4:2:0 (as for MPEG-1), 4:2:2 (horizontal subsampling), and 4:4:4 (no chroma subsampling).

Coding Interlaced Video

MPEG-2 accepts both progressive and interlaced inputs. Interlaced video can be coded into either field pictures or frame pictures. This may be switched from frame-to-frame. A frame picture refers to a combination of even and odd fields to form a complete frame, while a field picture is simply an even or odd field considered as a separate picture. Both frame and field pictures can be encoded as I-, P-, or B-pictures. The field pictures are suitable for video scenes that contain significant motion. Furthermore, MPEG-2 introduces the field/frame DC option per MB for frame pictures and new Motion Compensation (MC) prediction modes for interlaced video to handle interlaced inputs more efficiently.

A group of pictures (GOP) may contain a combination of field and frame pictures. However, the field pictures always appear in pairs of top and bottom fields. If the top field is a P- (B-) picture, the bottom field then must be P- (B-) picture. If the top field is an I-picture, then the bottom field can be an I- or a P-picture. A pair of field pictures is encoded in the order they should appear.

In a frame picture, each MB may be coded with a field or frame DCT. MPEG-2 allows this operation on a field-to-field basis for specific parts of a frame picture. Macroblocks that contain high motion may use field-DCT, while ones with little or no

motion but containing high spatial resolution may employ frame-DCT. Figure 2.32 depicts macroblocks with frame DCT and field DCT.

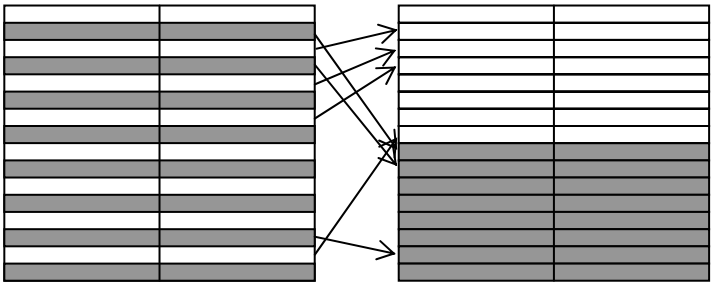


Figure 2.32: DCT options for interlaced frame pictures: frame DCT and field DCT

MC prediction modes for interlaced video.

MPEG-2 introduces two main types of predictions: *simple field* and *simple frame* prediction. Simple field prediction independently predicts each field using data from one or more previously decoded fields and it can use field predictions only. Simple frame prediction, on the other hand, performs a prediction for an entire frame using one or more previously decoded frames and either field or frame prediction can be used on a macroblock-to-macroblock basis.

Furthermore, two more prediction modes, 16x8 MC and dual-prime modes, are also introduced. 16x8 MC mode can be used in field pictures only. When this mode is used in a P-picture, it yields two motion vectors: one for the top and the other for the bottom 16x8 regions of top and bottom fields respectively. In a B-picture, this mode will produce four motion vectors. On the other hand, dual-prime mode is used only for P-

pictures. It uses one motion vector and a differential vector in frame pictures and uses two motion vectors for each field in field pictures.

Scalable Extensions

MPEG-2 assumes that different decoders with different spatial-temporal resolutions can decode and display the video sequence from the same bitstream. The minimum decodable subset of the bitstream is called the "*base*" layer. All other layers are "*enhancement*" layer. Two or three layers are allowed in MPEG-2 syntax. There are different forms of scalability: *Spatial*, *SNR*, *Temporal*, and *Hybrid* scalabilities. Spatial scalability refers to the ability to decode video at different spatial resolutions. SNR scalability refers to the ability to decode video using different quantizer step sizes for the DCT coefficients. Temporal scalability refers to the ability to decode video at different frame rates. Lastly, Hybrid scalability refers to a combination of the above scalabilities.

MPEG-2 also provides some extensions in the quantization and coding steps such as a new scanning scheme, a finer quantization, finer set of MQUANT values, and a separate VLC table for the DCT coefficients for the intra MBs. The alternate scan pattern, which fits interlaced video, is shown in Figure 2.33. The quantization weight for DC coefficients in intra MBs can be 8, 4, 2, or 1, which is actually fixed to 8 in MPEG-1, while the AC coefficients are quantized in the range of [-2048, -2047], as opposed to [-256, 255] in MPEG-1. All coefficients in non-intra MBs are quantized into the range of [-2048, 2047], whereas it is [-256, 255] in MPEG-1. In addition, MPEG-2 allows the optional set of 31 values to include real numbers ranging from 0.5 to 56. These values are listed in Table 2.7.

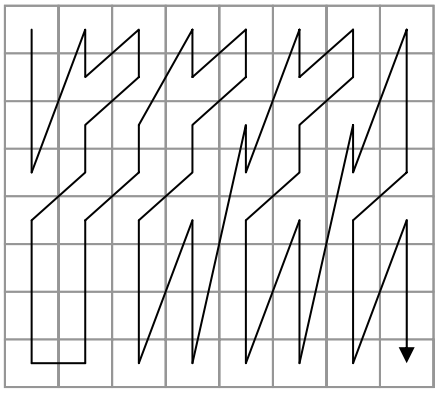


Figure 2.33: Alternate scan [46].

Table 2.7: Optional set of MQUANT values [46].

0.5	5.0	14.0	32.0
1.0	6.0	16.0	36.0
1.5	7.0	18.0	40.0
2.0	8.0	20.0	44.0
2.5	9.0	22.0	48.0
3.0	10.0	24.0	52.0
3.5	11.0	26.0	56.0
4.0	12.0	28.0	

Profiles and Levels

A *profile* is a "defined subset of the syntax of the specification". In other words, a profile imposes some bounds on the full syntax. It defines which tools or functionalities may be used to produce a bitstream and how [45].

A *level* is a defined set of constraints on the values, which may be taken by the parameters of the specification within a particular profile. An example of parameter constraints for each level is given in Table 2.8.

Table 2.8: Parameter constraints according to levels.

Level	Max. Pixels	Max. Lines	Max. Frames/sec
Low	352	288	30
Main	720	576	30
High-1440	1440	1152	60
High	1920	1152	60

Table 2.9: Profiles and bitrates (Mbps) at each level.

Profile \ Level	Low	Main	High-1440	High
Simple		1.5		
Main	4	15	60	80
SNR Scalable	4(3)	15(10)		
Spatially Scalable			60(15)	
High		20(4)	80(20)	100(25)

Table 2.9 summarizes the bitrates at each level that each profile supports. The numbers in parentheses indicate the maximum bitrate for the base layer. Note that the Simple profile does not include B-pictures.

MPEG-4

While MPEG-2 focused on a high-quality audiovisual application, MPEG-4, on the other hand, supports a wide range of applications from low bit rates, such as wireless/mobile multimedia and applications for the Internet, to very high bit rate applications used in studios. MPEG-4 is far beyond the traditional video and audio to end-users as provided by its predecessor, MPEG-2. Apart from typical audiovisual

representation, MPEG-4 also provides other types of media such as graphics, text, and synthetic visual and audio objects. Since MPEG-4 is object-based, each media is described as a media object and is coded into an independent stream. Audiovisual objects may include natural video sequence with/without shape information, general 3D animation, still image, natural audio, structured audio (MIDI), text-to-speech (TTS), etc. However, the individual objects can be composed to form a display scene by the Scene Descriptor (BIFS: Binary For Scenes). With the use of the Scene Description, an end-user can turn on/off individual objects, change the position of individual video objects, zoom in/out on interesting objects, choose different audio tracks (language/music), etc. This feature allows end users to have interactivity and full control over how the media should be presented.

MPEG-4 Visual

Two distinctive features that MPEG-4 visual improved on MPEG-2 are the ability to achieve better compression for the same visual quality and providing a wider variety of applications with larger set of profiles and levels. The core compression tools are based on the ITU-T H.263 standard [37], plus a mechanism of coding irregularly-shaped video objects. As a result, it enables the coding of separate foreground and background objects, which can be used to form a composite video scene. MPEG-4 also provides error resilience tools to help a decoder in recovering from a transmission error in an error-prone network. Its scalable coding also supports flexible transmission at a range of bitrates. Essentially the coding of animated visual objects such as 2D and 3D polygonal meshes, animated faces, and animated human bodies are also introduced. In addition,

MPEG-4 provides a coding for applications requiring "studio" quality video. In this case, visual quality is likely more important than high compression.

Profiles and Levels

Technically, profiles limit the tool set a CODEC has to implement and the levels for each profile restrict the computational complexity of the CODEC. A Profile@Level combination allows a CODEC builder to implement only the subset of the standard that is needed, while maintaining interworking with other MPEG-4 devices built to the same combination, and checking whether MPEG-4 devices comply with the standard ('*conformance testing*') [35]. Profiles in MPEG-4 exist for various types of media content (audio, visual, and graphics) and for scene descriptions. All profiles defined in MPEG-4 include Visual Profiles, Audio Profiles, Graphics Profiles, Scene Graph Profiles, MPEG-J Profiles, and Object Descriptor Profiles. Since this paper primarily focuses on regular video coding, only the Visual Profiles will be elaborated in the following section.

Visual Profiles

Visual profiles define the decoding of natural, synthetic, and synthetic/natural hybrid visual content. The profiles for natural video content and for synthetic and synthetic/natural hybrid visual content are described in Table 2.10 [35] and Table 2.11 [35] respectively.

Table 2.10: Visual profile for natural video.

Profile	Description	Applicaton
Simple	Provide efficient, error resilient coding of rectangular video objects.	Mobile networks video, videoconferrencing
Simple Scalable	Add support for coding of temporal and spatial scalable objects to the *Simple Visual Profile*.	Services providing more than one level of quality ie. Internet decoding.
Core	Add support for coding of arbitrary-shaped and temporally scalable objects to the *Simple Visual Profile*.	Simple content-interactivity: Internet multimedia.
Main	Add support for coding of interlaced, semi-transparent, and sprite objects to the *Core Visual Profile*.	Interactive and ntertainment quality broadcast and DVD.
N-Bit	Add support for coding video objects that have pixel-depths ranging from 4 to 12 bits to the *Core Visual Profile*.	Surveillance applicationsn
Advanced Real-Time Simple (ARTS)	Provide advanced error resilient coding techniques of rectangular video objects using a back channel and improved temporal resolution stability with the low buffering delay.	Real-time coding such as the videophone, tele-conferencing and remote observation
Core Scalable	Support for coding of temporal and spatial scalable arbitrarily shaped objects to the *Core Profile*. The main functionality of this profile is object based SNR and spatial/temporal scalability for regions or objects of interest.	Internet, mobile and broadcast.
Advanced Coding Efficiency (ACE)	Improve the coding efficiency for both rectangular and arbitrary shaped objects.	Mobile broadcast reception, the acquisition of image sequences, and other applications with high coding efficiency and small footprint is not the prime concern.
Advanced Simple	Support only rectangular objects, add support for B-frames and ¼ pel motion compensation, extra quantization tables, and global motion compensation (GMC).	Broadcast Television, video storage and playback such as DVD.
Fine Granularity Scalability	Allow truncation of the enhancement layer bitstream at any bit position so that delivery quality can easily adapt to transmission and decoding circumstances. (It can be used with *Simple* or *Advanced Simple* as a base layer.)	Streaming video
Simple Studio	Provide very high quality. It only supports I-frames, but does support arbitrary shape and multiple alpha channels. Bitrates go up to almost 2 Gigabit per second.	Studio editing applications, studio distribution.
Core Studio	Add P-frames to *Simple Studio,* making it more efficient but requiring more complex implementations	Studio editing applications, requiring less storage and bitrates.

Table 2.11: Visual profile for synthetic and synthetic/natural hybrid visual.

Profile	Description	Application
Simple Facial Animation	Provide a simple means to animate a face model.	Audio/video presentation for the hearing impaired.
Scalable Texture	Provide spatial scalable coding of still image (texture) objects.	Applications needing multiple scalability levels, such as mapping texture onto objects in games, and high-resolution digital still cameras.
Basic Animated 2-D Texture	Provide spatial scalability, SNR scalability, and mesh-based animation for still image (textures) objects and also simple face object animation.	
Hybrid	Combine the ability to decode arbitrary-shaped and temporally scalable natural video objects (as in the *Core Visual Profile*) with the ability to decode several synthetic and hybrid objects, including simple face and animated still image objects.	Various content-rich multimedia applications.
Advanced Scaleable Texture	Support decoding of arbitrary-shaped texture and still images including scalable shape coding, wavelet tiling and error-resilience.	Applications requiring fast random access as well as multiple scalability levels and arbitrary-shaped coding of still objects i.e. fast content-based still image browsing on the Internet, multimedia-enabled PDA's, and Internet-ready high-resolution digital still cameras.
Advanced Core	Combine the ability to decode arbitrary-shaped video objects (as in the *Core Visual Profile*) with the ability to decode arbitrary-shaped scalable still image objects (as in the *Advanced Scaleable Texture Profile*).	Various content-rich multimedia applications such as interactive multimedia streaming over Internet.
Simple Face and Body Animation	A superset of the *Simple Face Animation Profile*, adding, obviously, body animation.	

Levels

A *Level* in a profile indicates constraints on the parameters of the bitstream and the maximum performance required to decode an MPEG-4 coded sequence. A summary of the levels for simple-based profiles in MPEG-4 visual is listed in Table 2.12 [46].

Table 2.12: Levels for Simple-based profiles.

Profile	Level	Typical Resolution	Max. Bitrate	Max. Objects
Simple	L0	176x144	64 kbps	1 simple
	L1	176x144	64 kbps	4 simple
	L2	352x288	128 kbps	4 simple
	L3	352x288	384 kbps	4 simple
Advanced Simple (AS)	L0	176x144	128 kbps	1 AS or simple
	L1	176x144	128 kbps	4 AS or simple
	L2	352x288	384 kbps	4 AS or simple
	L3	352x288	768 kbps	4 AS or simple
	L4	352x576	3 Mbps	4 AS or simple
	L5	176X576	8 Mbps	4 AS or simple
Advanced Real-Time Simple (ARTS)	L1	176x144	64 kbps	4 ARTS or simple
	L2	352x288	128 kbps	4 ARTS or simple
	L3	352x288	384 kbps	4 ARTS or simple
	L4	352x288	2 Mbps	16 ARTS or simple

Tools and Objects

A *tool* defines a subset of coding functions for a specific feature, such as basic video coding, interlaced video, coding object shapes, etc. An *object* is simply a video element (such as a sequence of rectangular frames, a sequence of arbitrary-shaped regions, and a still image) that is coded using one or more tools.

Video Object

A video object (VO) in MPEG-4 is not limited to an ordinary rectangular video frame. In fact, it refers to an area of a video scene including an arbitrary-shaped region, and it may present for an arbitrary length of time. An instance of a VO at a particular point in time is called a "*video object plane*" (VOP). MPEG-4 visual allows user to access (seek, browse) and manipulate (cut, paste) video objects.

Basically, each VOP is considered as a single frame or a sequence of frames that form a VO. Figure 2.34 shows a VO that consists of three irregular-shaped VOPs at which each one of them exists within a frame and can be coded separately (this refers to object-based coding in MPEG-4). Figure 2.35 illustrates a video scene, which is composed of two foreground objects (VOP1and VOP2) with a background object (VOP3). In this scenario, each video object may be coded separately using different visual qualities and temporal resolutions. The end-user may manipulate these objects by composing them with different types of objects from other sources, such as a synthetic object, to form a particular scene. Figure 2.36 shows a new video scene composed by adding VO1 and VO2 with a new background VO3.

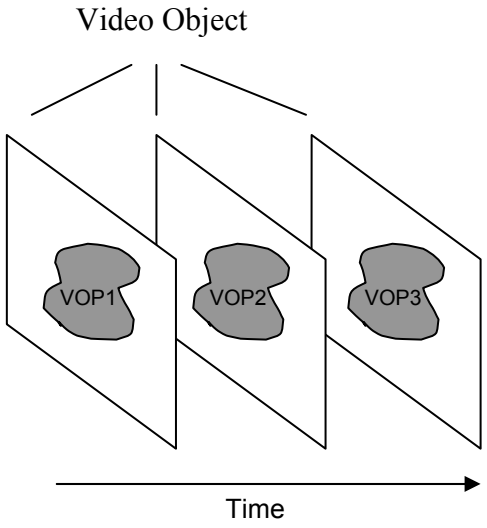


Figure 2.34: VOPs and VO (arbitrary shape) [27].

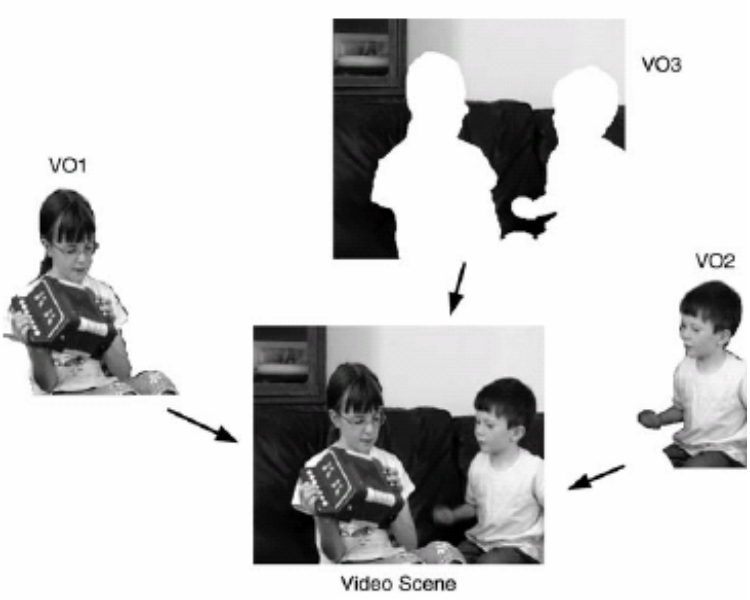


Figure 2.35: A video scene consisting of three VOs [27].

Figure 2.36: Video scene composed of VOs from separate sources [27].

Even though MPEG-4 provides such an advanced coding technology, most applications are still employing the rectangular frame coding. The simple profiles family provides tools to handle rectangular VOPs as shown in Figure 2.37. Since MPEG-4 is the extension for MPEG-1 and MPEG-2, the basic tools used here are similar to those used in its predecessors. The advance simple and advance real-time simple profiles add more tools to the simple profile to support better coding scheme and capacities to handle error.

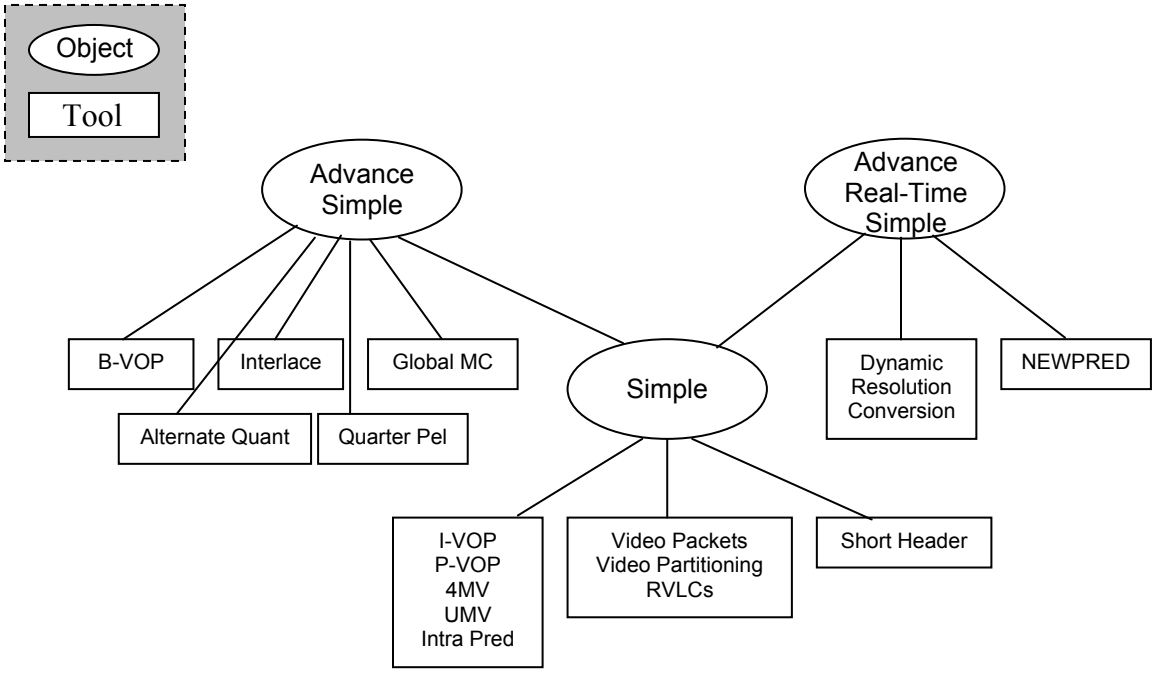


Figure 2.37: Tools and objects for coding rectangular frames [27].

Video Packet

A video packet in MPEG-4 is similar to a slice in MPEG-1 and MPEG-2. MPEG-4 transmits a VOP, which consists of one or more video packets. A video packet, as

shown in Figure 2.38, consists of a resynchronization marker, a header field, header extension code (HEC), optional extension header, and a group of coded macroblocks in raster scan order. The resynchronization marker is followed by a count of the next macroblock number to enable the decoder to locate the first macroblock of the packet. Next to it is the quantization parameter. If the flag HEC is set to 1, a duplicate of this VOP header will follow. This helps the decoder to recover the VOP header in case the first one is lost or corrupted. Essentially, if an error occurs in a current video packet, the decoder can use the first resynchronization marker in the next video packet to prevent error propagation.

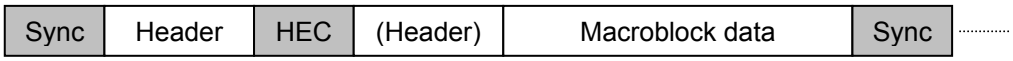


Figure 2.38: Video packet structure.

Data Partitioning

The data partitioning tool divides a video packet into two partitions in order to reduce the impact of transmission errors. The first partition contains coding mode information for each macroblock with DC coefficients for each block of intra macroblocks or motion vectors of inter macroblocks. The second one contains AC and DC coefficients of inter macroblocks. The first partition is placed right after the video packet header, while the second one follows the resynchronization marker. The first partition is considered to be more important than the second one. In case an error is detected, if the first part can be recovered, it is adequate for the decoder to reconstruct the packet.

Reversible VLCs

Another attempt for error recovery is using reversible VLCs to encode DCT coefficient data. Reversible VLCs are decodable in both the forward and backward directions. If an error occurs, a decoder will decode the video packet in the reverse direction at the next resynchronization mark. Consequently, the error may possibly be limited to only a single macroblock.

Short Header

The short header tool allows MPEG-4 to encode an I- or P-VOP that has identical syntax to an I- and P-picture coded in the baseline mode of H263. The macroblocks within a VOP are organized into Groups Of Blocks (GOBs) as used in H263. Each GOB may contain one or more complete rows of macroblocks and may start with a resynchronization marker for error resilience concealment.

Four Motion Vectors Per Macroblock

MPEG-4 introduces this tool to increase efficiency in motion compensation by assigning one motion vector for each 8x8 block in the luminance component. As a result, one luma macroblock contains four motion vectors. Similarly, the chrominance component is divided into four 4x4 blocks and contains four motion vectors, one for each block. This tool is useful especially in reducing residual energy in some areas that contain complex motion or near the boundaries of the moving object. However, this tool may incur overhead in sending more motion vectors to the decoder, but it can be applied on a macroblock-by-macroblock basis for better quality in a video sequence.

Unrestricted Motion Vectors (UMV)

The UMV tool allows motion vectors to point to a region outside of the reference VOP boundary. It is necessary, for instance, in certain situations where an object is moving toward the edge of the VOP and the best match for a macroblock is found on the boundary of the VOP. In doing so, the reference VOP may be extrapolated beyond the boundary of the VOP giving a better match for motion compensation as shown in Figure 2.39 [27].

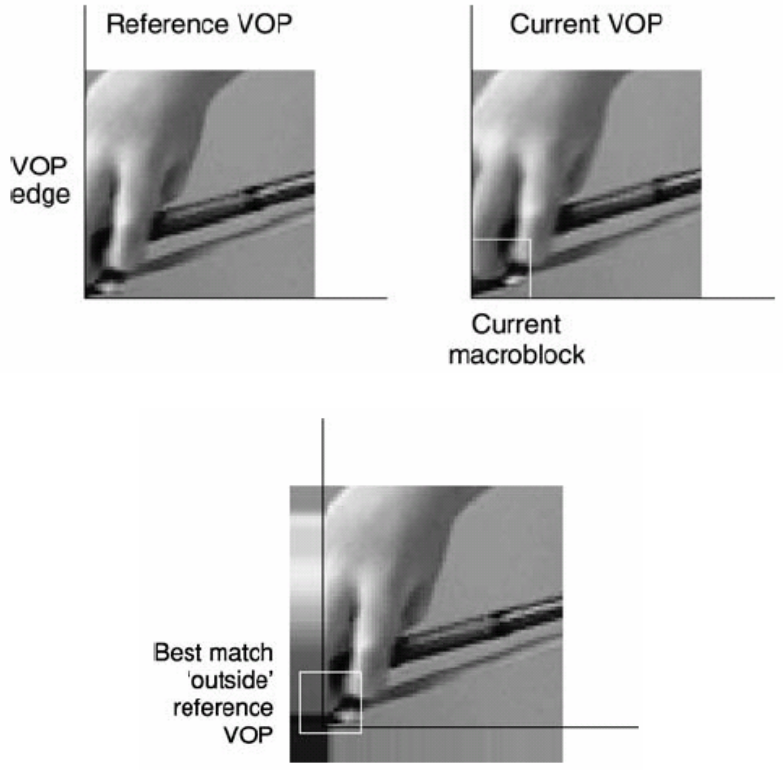


Figure 2.39: Reference VOP, current VOP, and reference VOP extrapolated beyond boundary.

Global Motion Vector (GMV)

In some cases such as camera pan or a large moving object, the majority of the macroblocks in a video object may produce similar motion. The camera zoom and rotation may create more complex motion vectors but they tend to be moving in relative directions. Hence, these motion vectors can be described as a default '*global*' motion for the entire VOP. As a result, only a small number of motion parameters may be needed to transmit to a decoder. GMC takes advantage of this mechanism by encoding and sending the global motion parameter in the VOP header to the decoder. This alternative coding method can be activated by setting the parameter *sprite_enable* to 'GMC' in a Video Object Layer (VOL) header.

In MPEG-4, the global motion compensation (GMC) enables the encoder to allow a VOP to contain up to four GMVs along with the location for each of them. For each pixel position in the VOP, an individual motion vector is calculated by interpolating between the GMVs as displayed in Figure 2.40. Additionally, the GMC tool also enables compensation for a variety of types of motion including rotation (as shown in Figure 2.41a.), camera zoom (as shown in Figure 2.41b), warping, and traditional or linear motion.

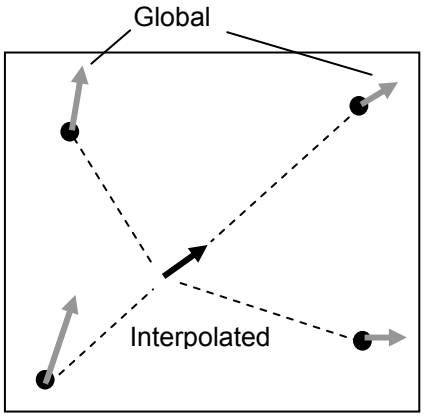


Figure 2.40: VOP, GMVs and interpolated vector.

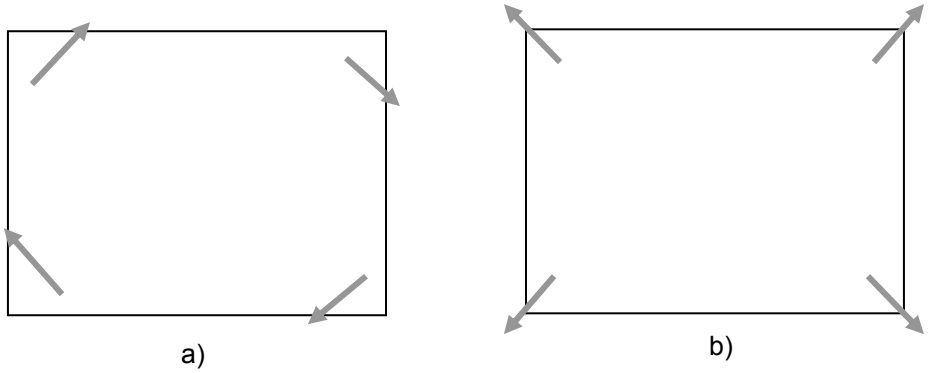


Figure 2.41: GMC a) compensating for rotation. b) compensating for camera zoom.

Simple Profile

The simple profile of MPEG-4 visual employs a CODEC called a Very Low Bitrate Video (VLBV) Core, which is similar to the hybrid DPCM/DCT model as described earlier. The basic requirements of the coding process for intra VOPs in the simple profile of MPEG-4 visual is shown in Figure 2.42 and for inter VOPs in Figure 2.43.

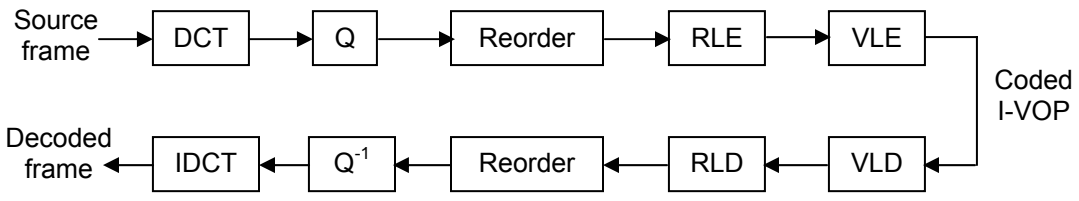


Figure 2.42: I-VOP encoding and decoding stages.

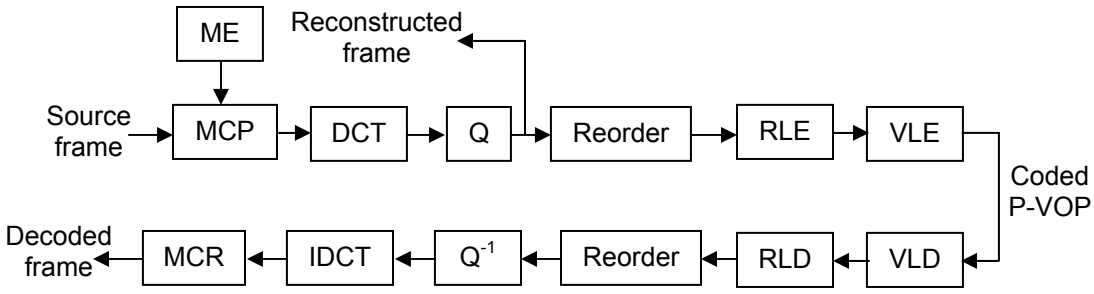


Figure 2.43: P-VOP encoding and decoding stages.

This chapter provides a background study regarding the digital video processing model, color spaces, and video coding techniques. The key topics of the MPEG system, which are related to the application design and implementation, are described in more detail. Some topics that may be useful for future work are provided as well.

CHAPTER III

VISUAL MOTION ESTIMATION

The analysis of motion in video sequences has become a pivotal topic in various fields of research. One common task is to track moving object(s) of interest in a video scene. The target object's trajectory may be calculated in order to have the camera keep an eye on that moving object over a period of time. In this case, the motion in the picture that we have to pay particular attention to is that of the moving object. This is called "*local motion*". However, the object motion may turn out to be undesirable to other researchers interested in the motion of the camera itself. This type of visual motion is known as "*egomotion*". In this scenario, the motion of the entire scene will be analyzed while the motions of the local objects are considered as random noise. The key operation of the egomotion is the transform model for recovering the 3D camera motion with respect to the real world environment from the 2D coordinates of the perspective projection from the image plane. The possible camera motions include translation, rotation, and zoom factor. Generally, a camera may experience a translation along the X, Y, or Z axes, or a combination of them. In addition, there may have been rotation around the X, Y, Z axes, or a combination of these. Applications that employ egomotion estimation include video stabilization, video mosaicing, video annotation, autonomous vehicle driving systems, and mobile robot navigation.

A video stabilizer, for example, analyzes the camera motion to compensate for the jerky image sequence from a shaky hand. Video mosaicing may be used to create a

panoramic view [47] or may be used in "Sprite Coding" for MPEG-4 [48]. Furthermore, egomotion estimation may be used in driving a vehicle autonomously [49]. A more advanced system by Muratet et al. [26] is to compute egomotion to fly a helicopter without human assistance. This chapter will discuss egomotion estimation methods in general. Subsequently, we will focus mainly on applying egomotion estimation for mobile robot navigation.

Visual Motion Field

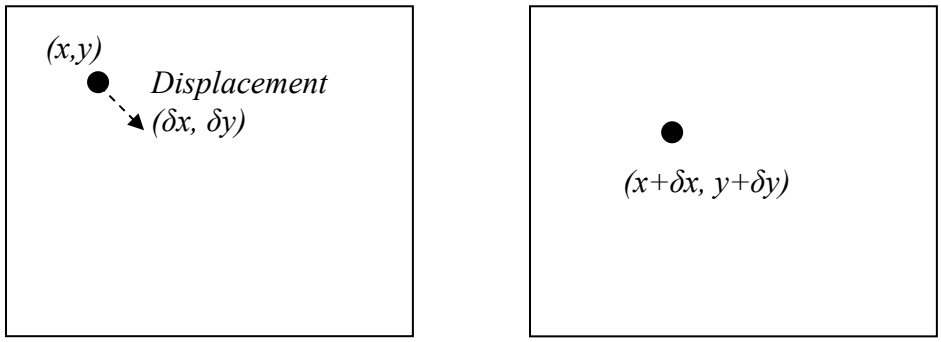
The 3D camera motion parameters are computed from estimates of the 2D motion field that occurs in a sequence of video images. Typically, it is assumed that the environment's structure is stationary and the projected 2D motion in the image plane is predominantly caused solely by the motion of the camera. A number of approaches have been proposed for tackling this problem, and can generally be classified into two methods, direct and indirect.

Direct Method

The direct method, also called the *gradient method*, employs the spatial and temporal gradient of the image sequence based on the Taylor series expansion of the image function with respect to the motion parameter space or image space [50]. Generally speaking, the method is based on using spatial and temporal derivatives of the image intensity. Its main purpose is to compute the velocity vectors, called optical flow, at each pixel in the image.

Optical Flow

Optical flow [7] is the distribution of apparent velocities of movement of brightness patterns in an image. However, a single image does not sufficiently describe the motion information. Rather, optical flow can be generated by the relative motion of a camera or the objects in the scene between two successive frames as shown in Figure 3.1.



a) Image I(x, y) at time t *b) Image I(x, y) at time t+δT*

Figure 3.1: Displacement vector between image at time *t* and *t+δT*.

From the figure, when the approximate motion and time interval are small, the basis of differential optical flow, the *Motion Constraint Equation*, can be written as [51]:

$$I(x, y, t) = I[(x + \delta x), (y + \delta y), (t + \delta t)] \qquad \text{Eq (3.1)}$$

It is assumed that the change in the image's spatial intensity is caused by pixel translation. By taking a 1st order Taylor Series Expansion in equation (3.1), a new equation can be expressed as:

$$I[(x+\delta x),(y+\delta y),(t+\delta t)] = I(x,y,t) + \frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} + higher\ order\ terms. \quad \text{Eq (3.2)}$$

The higher order terms are typically small and can be omitted. Then combining equation 3.1 and equation 3.2 yields

$$\left[I(x,y,t) + \frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} \right] - I(x,y,t) = 0 \quad \text{Eq (3.3)}$$

or

$$\frac{\partial I}{\partial x}v_x + \frac{\partial I}{\partial y}v_y + \frac{\partial I}{\partial t} = 0 \quad \text{Eq (3.4)}$$

where $\upsilon_x = \frac{\delta x}{\delta t}$ and $\upsilon_y = \frac{\delta y}{\delta t}$ represent the x and y components of image velocity or optical flow and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ *and* $\frac{\partial I}{\partial t}$ represent image intensity derivatives at (x, y, t) which normally are written in partial derivative form as:

$$I_x = \frac{\partial I}{\partial x}\ ,\ I_y = \frac{\partial I}{\partial y}\ \ and\ \ I_t = \frac{\partial I}{\partial t} \quad \text{Eq (3.5)}$$

Then, we can group together the intensity derivatives and optical flow parts and rewrite it as:

$$(I_x, I_y) \cdot (\upsilon_x, \upsilon_y) = -I_t \quad \text{Eq (3.6)}$$

or

$$\nabla I \cdot \vec{\upsilon} = -I_t \qquad \text{Eq (3.7)}$$

where $\nabla I = (I_x, I_y)$ represents the spatial intensity gradient and $\vec{\upsilon}$ represents the image velocity or optical flow at pixel (*x, y*) at time *t*. (*Note: $\vec{\upsilon}$ is a column vector without the transpose sign*).

It is obvious that the gradient method relies primarily on the consistency of image texture and continuous image motion. It performs well under small displacements between subsequent images. The method, however, suffers from the estimate of the velocity vectors at points perpendicular to the tangent of a boundary or edge in the image. This is known as "*aperture problem*" [52].

Indirect Method

The indirect method, also known as the *displacement method*, performs motion estimation on the basis of the feature-based technique. It is mainly dependent upon computing displacement vectors on a set of corresponding features in the image. Discontinuities in image intensity or motion, which cause the aperture problem in the gradient methods, are selected as features for its operation. The indirect method may be further categorized into the tracking feature approach and the block-based approach.

Tracking Feature Approach

The features under consideration include line segments, high contrast regions where brightness and darkness occur, and (the most widely used) corners are selected and

tracked over a series of consecutive image frames. Then, at each frame, the associated displacement vectors are discretely generated from the tracked correspondences.

Even though the indirect method seems to have advantages over the direct one, it also raises two problems to be taken into account. First, due to perspective, a feature in the distance may not be able to retain its shape as it moves closer to the observer. A discussion on the "*range dependence*" of feature extraction is given in [53]. Second, the selection of reliable features to be tracked between consecutive frames for a length of time is not easy. This problem is referred to as the "*correspondence problem*" [54].

Block-Based Approach

The principle idea behind this approach is that the current image frame is divided into blocks of sub-images. The selected blocks contain features needed in the matching process. Subsequently, each block will be searched for the best match of its correspondence within a search window in the previous frame. The associated displacement vector will then be calculated.

The techniques used to compute the motion displacement may include applying the gradient method, finding the correspondence features, or calculating the sum of absolute error of the current block respect to search region. An example of the latter, the technique of Sum of Absolute Difference (SAD), with a block size of 16 pixels is given in Equation 3.8.

$$SAD(u,v) = \sum_{k=-8}^{7}\sum_{l=-8}^{7}\left|SW(u+l,v+k) - RB(l,k)\right| \qquad \text{Eq (3.8)}$$

where SW is the Search Window and RB is the Reference Block.

Interestingly, this technique has been deployed by the MPEG motion compensation and estimation unit as well. As a result, this motion displacement field can be obtained directly from the MPEG encoder without further calculation.

Camera Model on Perspective Projection

The perspective projection maps a point, **P**, in 3D space environment onto the 2D image plane using an ideal pinhole camera. Basically, the ray from the projected point, *P*, will pass straight into the origin of the projection, which is the center of the lens. The projected point position is derived by

$$\frac{x}{F} = \frac{X}{Z} \quad , \quad x = F\frac{X}{Z} \qquad \text{Eq (3.9)}$$

and

$$\frac{y}{F} = \frac{Y}{Z} \quad , \quad y = F\frac{Y}{Z} \qquad \text{Eq (3.10)}$$

where X, Y, and Z are the principal axes and F is the camera focal length.

Figure 2 illustrates a side/top view of a perspective projection model. Two points, $\mathbf{P_1}$ and $\mathbf{P_2}$, in 3D space are projected onto an imaginary *xy* image plane, which is located at a distance of the camera focal length (*F*). The origin of the coordinate system lies at the center of the camera lens.

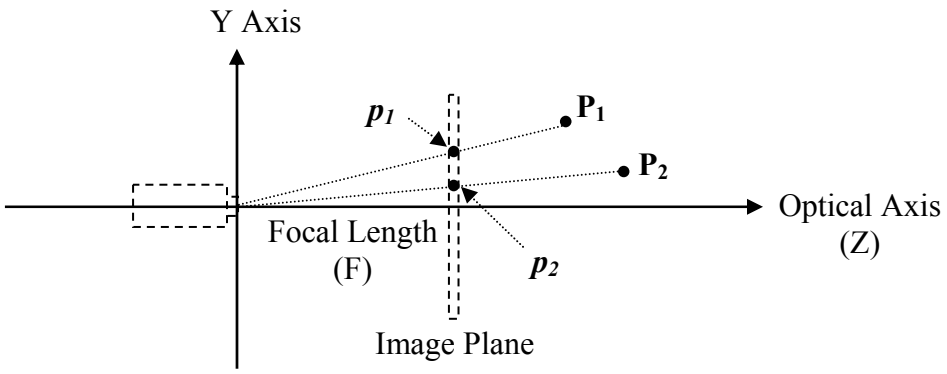


Figure 3.2: Perspective projection model.

Camera Motion Model

By assuming the origin of the coordinate system is located at the center of the camera, motion of the camera will usually cause a point in 3D space to change its location in the image. Such motion refers to rotation around principal axes, translation along principal axes, and change of focal length (zoom parameter).

Effect of Camera Rotation

When a camera undergoes a pure 3D rotation around X, Y, or Z axis, a new 3D camera coordinate system $[X'\ Y'\ Z']^T$ can be the result of

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \text{Eq (3.11)}$$

where R is a 3x3 rotation matrix and rotations about X, Y, and Z can be expressed as

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad \text{Eq (3.12)}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \text{Eq (3.13)}$$

$$R_z(\varphi) = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Eq (3.14)}$$

When successive rotations are applied to the camera coordinate system (X, Y, and Z), R will become a combination of those rotation as

$$R = R_x(\phi)R_y(\theta)R_z(\psi) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \text{Eq (3.15)}$$

where

$r_{11} = \cos\psi \cos\theta$

$r_{12} = \sin\psi \cos\theta$

$r_{13} = -\sin\theta$

$r_{21} = -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi$

$r_{22} = \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi$

$r_{23} = \cos\theta\sin\phi$

$r_{31} = \sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi$

$r_{32} = -\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi$

$r_{33} = \cos\theta\cos\phi$

Effect of Camera Translation

Camera translation along principal axes simply moves its coordinate system to the opposite direction that it translates to. A new coordination system can be written as

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \quad \text{Eq (3.16).}$$

A special case of camera motion, zoom operation, is defined as

$$x' = F' \frac{X'}{Z'}$$ Eq (3.17)

$$y' = F' \frac{Y'}{Z'}$$ Eq (3.18)

where $F' = f \cdot F$, at which f indicates the zoom factor.

Camera Motion Estimation Model

Camera motion estimation is the process of estimating 3D camera motion based on the observed 2D motion field on the projected image plane. A certain set of basic constraints are made such that the observed points in the 3D environment remains stationary and the 2D motion field is predominantly caused by camera motion. A restriction on camera movement may be applied as well. From the above assumptions, it is possible to model camera operation, with respect to the 3D coordinate system.

Affine Model

Based on research and studies [55] [56] [57] [58] [59] [60], basic camera operation can be categorized as zoom, pan, tilt, and rotation (around the Z axis). These camera motions can be described by the affine model as shown in Equation 3.19.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_5 \\ a_6 \end{bmatrix} \quad \text{Eq (3.19).}$$

The affine model provides 6 parameters to describe the basic camera motions where a_5 and a_6 indicate a translation operation along the X and Y axes respectively, a_1 and a_4 indicate the scaling factor, and a_2 and a_3 indicate shearing of the object. In other words, the affine model can be rewritten in terms of camera operation as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} zoom & rotate \\ -rotate & zoom \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} pan \\ tilt \end{bmatrix} \quad \text{Eq (3.20).}$$

The affine model eliminates the non-linear terms by generalizing the rotation around the Y axis to a simple *pan* and the rotation around the X axis to a simple *tilt*. This makes it possible to solve for all camera parameters linearly. Consequently, a straightforward technique, such as linear least square fitting, can be used to solve the problem.

Because of its simplicity, a number of researchers in camera motion estimation have adopted the affine model to their work on the basis of their theoretical assumption. Wang and Huang [55], for instance, use Equation 3.20 to analyze camera motion in the MPEG domain. The system to estimate the camera parameters is defined as

$$\vec{Y} = \vec{H} \cdot \vec{X} + \vec{N} \qquad \text{Eq (3.21)}$$

where $\vec{Y} = \begin{bmatrix} u1 \\ v1 \\ u2 \\ v2 \\ \dots \end{bmatrix}$ is a vector of MPEG motion vectors ,

$\vec{H} = \begin{bmatrix} x1 & y1 & 1 & 0 \\ y1 & -x1 & 0 & 1 \\ x2 & y2 & 1 & 0 \\ y2 & -x2 & 0 & 1 \\ \dots & \dots & \dots & \dots \end{bmatrix}$ represents the spatial matrix of macroblock center locations,

$\vec{N}$ represents a noise term, and $\vec{X} = \begin{bmatrix} zoom & rotate & pan & tilt \end{bmatrix}$ is the vector of camera motion parameters to be estimated. As a result, $\vec{X}$'s least square estimator can be obtained by

$$\tilde{\vec{X}} = \left(\vec{H}^T \vec{H}\right)^{-1} \vec{H}^T \vec{Y} \qquad \text{Eq (3.22).}$$

In the noise term, outliers may be eliminated gradually by examining the error from the current estimate at each iteration.

Jin et al. [56] employ the affine model in their proposed technique, called *a probabilistic model for camera zoom detection*. The model uses the Expectation-

Maximization (EM) algorithm to estimate the probability of a zoom versus a non-zoom operation from standard MPEG motion vectors. The results indicate superiority over the traditional methods such as those introduced by Wang and Huang [55] and Park et. al. [57] in terms of precision and recall.

Kim et. al. [58] also utilize the affine model with threshold-based qualitative interpretation to detect basic camera motions. They extend the interpretation model to represent a hyperbolic flow, which is assumed to occur when object motion is dominant over global motion.

Furthermore, the affine model is adapted to a variety of techniques in estimating camera motion. Chiew et al. [59] use Hough transforms based on the affine model for global motion estimation. The application is reported to be accurate and suitable for real-time implementation. Smolic et al. [60] extends the generic affine model to motion parameters in higher order. The model employs a higher order polynomial to create a parabolic transformation, which is described with 12 parameters. The technique is designed for the long-term global motion estimation of image objects. The global image, the background scene, from the motion estimation is stored in long-term memory. Later, the predicted image can be generated from the estimated parameters and this long-term memory. This technique is very useful for MPEG-4 sprite coding.

Camera Rotation Estimation

As mentioned above, the affine model assumes that camera pan and tilt are the consequences of camera translation along the X and Y axes respectively. On the other hand, if the camera is fixed to a wall, then the pan effect will be generated by camera

rotation around the Y axis while tilt will be an effect of rotation around the X axis. Hence, the use of the affine model is no long applicable to solving for these camera parameters.

In order to find the rotation angles, the rotation matrix of Equation 3.15, is needed. Suppose, given that (x, y) is a projected coordinate of a stationary in 3D point P before the camera operation and that (x', y') is the new coordinate after the camera operation, the relationship in a particular camera operation can be expressed as

$$x' = F' \frac{r_{11}x + r_{12}y + r_{13}F}{r_{31}x + r_{32}y + r_{33}F} \qquad \text{Eq (3.23)}$$

$$y' = F' \frac{r_{21}x + r_{22}y + r_{23}F}{r_{31}x + r_{32}y + r_{33}F} \qquad \text{Eq (3.24).}$$

Note that $F' = f\,F$; where f is a zoom parameter and the new coordinate, (x', y') after zoom operation is $x' = F'\frac{X'}{Z'}$, $y' = F'\frac{Y'}{Z'}$.

Equation 3.23 and Equation 3.24 contain only 2D variables on the projected plane. As a result, the camera operations, rotation, zoom, and focal length, can be estimated by applying non-linear least square fitting method from the motion field on the image plane. Park et al. [61] demonstrate the estimate of camera parameters in 5 degrees of freedom, focal length, zoom, and 3D rotation parameters. The estimating parameters are least square fitted using the Levenberg-Marquardt method [62]. The simulations show that the proposed method is able to estimate the focal length from the test video. The proposed method also performs very well on detecting 3D rotation and zoom.

Later, Park et al. [57] apply the estimate of global camera motion for video annotation and retrieval. The application detects zoom, yaw, pitch, and roll operations from motion vectors of encoded video. The estimation model is based on Equation 3.23 and Equation 3.24 with $F' = 1 + f$. Two least-square estimation techniques, the Levenberg-Marquardt method [62] and the extended Kalman filter [63] [64], are compared. The results indicate that both techniques yield similar performance but the Levenberg-Marquardt method is computationally more efficient than the extended Kalman filter.

Broszio and Grau [65] propose a very interesting application for estimating camera pan, tilt, and zoom for integration of virtual objects into video sequences. The application uses the indirect method based on conventional feature detection and correspondence analysis. The camera parameters are estimated from a real moving camera using a least square technique on the selected correspondences. Once the parameters have been established, the virtual scene with a computer generated (CG) object can be synchronized. The ultimate goal is to create a combination scene as if they were taken from only one single camera.

Global camera motion estimation is essential for many areas of study. One valuable field that provides the motivation for this work is the estimation of egomotion for mobile robot navigation. The following sections will discuss camera motion estimation and mobile robot navigation.

Camera Motion Estimation for Mobile Robot Navigation

A mobile robot may be equipped with various types of sensors such as bumpers, sonars, laser range finder, wheel-encoders, etc. These offer specific functionality for the robot to fulfill the localization task and obstacle avoidance task. For instance, wheel-encoder odometry passively provides the travel speed, distance, and orientation while the sonar and laser range finder provide useful information to help the robot actively make a decision on how to react to a detected obstacle. Interestingly, there has been a lot of work on using video camera to perform these two essential tasks successfully. Fundamentally, a camera will be attached to a mobile robot. While the robot moves, a motion field on the image plane will be generated. The main focus is the implementation of egomotion estimation for the robot heading direction and detecting obstacles. The heading direction can be calculated based on the perspective model with the fact that the 2D velocity vectors radiate from the vanishing point, called the *Focus Of Expansion* (FOE). The FOE technically indicates the direction along which the observer (robot) moves. On the other hand, an obstacle may be detected from the motion field segmentation. With some constraint applied, the object location can be obtained. However, the detailed implementation may vary among research groups.

Burger and Bhanu, 1990 [15] address a difficulty in precisely computing a single-point FOE for vehicle navigation from a noisy displacement vector field. They, instead, suggest that the FOE should be located from a computation of a 2D region FOE. Hence, they propose a technique, called "Fuzzy FOE", to compute the heading direction of a land-based vehicle. This method is reported to be more robust than the conventional single-point FOE techniques. Since the system uses the indirect method, a discrete model,

to compute the egomotion from perspective image sequences, the resulting FOE is considered as the "direction of accumulated translation" over a certain period of time (as opposed to the velocity-based models which treat the FOE as the "direction of instantaneous heading").

The experiments they conducted are performed on an autonomous land-based vehicle traveling at approximately 14 kilometers per hour. The camera captures images at 2 frames per second with a spatial resolution of 512x512 pixels. About 25 features of selected endpoints and corners are tracked from the computation of binary edge detection in the range of 2-16 frames. At each frame, three components, fuzzy FOE, angles of horizontal and vertical rotation, and approximate traveled distance, are estimated from the displacement vector field. In the presence of noise, small displacement vectors tend to cause a problem in locating the FOE. Hence, from the results, they suggest that using the longer vectors in the lower central area of the image plane, located near the vehicle, are more reliable for estimating the approximate vehicle velocity and travel distance.

Dev et al. 1997 [16] analyze the optical flow field from a sequence of images taken from an on-board camera for mobile robot navigation. The main objective is to drive the robot through the center of a corridor. Under this constraint, the spatial structure of the environment can be obtained. Egomotion estimation and relative depths toward the walls can be derived from optical flow. The key feature is the use of the temporal derivative of the flow field to compute the normal surfaces of the walls. This feature is very significant in estimating the robot orientation as well as maintaining the robot position in the middle of the corridor. It is claimed that the implementation is able to achieve real-time performance on a standard PC without any special hardware.

Branca et al.1997 [17] give a definition of passive navigation as "*the ability of an autonomous agent to determine its motion with respect to the environment*". They propose a technique of estimating egomotion parameters to determine the heading direction and the time to collision (TTC) with the environment. They also address a difficulty in accurately estimating the motion field due to the need for *a priori* knowledge of the viewed scene properties. Therefore, certain constraints on some known information must be imposed so that a more robust 3D motion parameter estimation can be achieved in terms of accuracy. In this work, the robot motion is restricted to travel on a flat surface in a stationary environment and rotates around the Y axis. Since rotation with translation will cause the FOE location to shift but still preserve the radial shape, the location of the FOE on the X axis can be computed from

$$FOE_x = \frac{T_x}{T_z} \qquad \text{Eq (3.25)}$$

where T_x is a translation along the X axis and T_z is a translation along the Z axis.

Stoffler and Schnepf 1998 [18] and Stoffler et al.2000 [19] create an equivalent optical flow field, motion field, for robot obstacle avoidance based on the MPEG-like motion estimation. A special block-based motion estimation processor, called MEP, is developed in order to create a robust optic-flow-like vector field in real time. Each reference block (RB 16x16 pel) in the current frame searches for the best match of its correspondence in a search window (SW 32x32 pel) in the previous frame with the lowest SAD as given by Equation 3.8. Additionally, the system is equipped with an external

subsystem to calculate an additional confidence value to sift out the unwanted motion vectors, which may not represent real motion. On the mobile robot, a camera is mounted parallel to the heading direction. Furthermore, the robot motion is constrained to two degrees of freedom, translation along the trajectory of the vehicle and rotation around the vertical axis. The motion vector field on the image plane is divided into two main portions. The lower part, located right in front of the robot, called the "ground window", is used to detect obstacles and calculate their position for avoidance planning. The upper part, mostly containing the small motion vectors around the FOE location, is called the "blind spot". Like its name, these motion vectors are not used in object localization because they do not provide sufficient information in estimating the object position. In the obstacle detection process, the 3D points in the environment are transformed into the robot coordinate system to generate a 2D obstacle map for robot navigation. The system is reported to perform well in real-time. The magnitude of the accuracy of 3D reconstruction is 10cm, which typically is sufficient for the obstacle avoidance task.

Campbell et al., 2004 [20] point out that the decreasing price of digital video cameras relative to 1D sonar rangefinders makes cameras more cost effective for robot navigation. In this work, a camera is mounted on a mobile robot having no wheel encoders and running at 4 cm/sec. The prerecorded image sequences of a move-turn-move operation are analyzed in terms of the precision of visual odometry on 4 different surfaces, carpet, grass, asphalt, and ice. The optical flow field is created from uncompressed video using a direct method. A consensus method, RANSAC [66], is used to eliminate outliers during the optical flow creation. A new technique for estimating egomotion is introduced, in which the optical flow field is divided into two portions: the

ground region and the sky region separated by a horizon line. The vectors in the ground portion are used to compute the *x* and *z* translation whereas the vectors in the sky portion, which is less affected by translation, are used for rotation estimation. Since the camera is fixed with zero roll and yaw and its pitch angle is constant relative to presumably flat surfaces, simple trigonometry can be used to map the optical flow vectors to the traveled distance on the surface. The estimates of tangential and normal translation are described as follows. The vectors in the sky region are estimated for the robot rotation from the consensus value. The vectors in the ground region are estimated based on the perspective model for the displacement along the ground plane. The ground vectors are then subtracted from the estimate rotation, and finally the tangential and normal translation are obtained from the remaining ground motion field left from the consensus.

Table 3.1 shows the performance of visual odometry on different surfaces both indoor and outdoor. The results indicate the best overall performance on the indoor carpet. However, the worst performance on estimating rotation is posted on the indoor test, which is on a carpet surface. This is because the calculation of rotation relies more on points at a distance, and the indoor environment provides less distant points.

Table 3.1: Visual odometry performance by terrain type [20].

Terrain	**Incremental Error Translation**	**Incremental Error Rotation**	**Average Cumulative Error Rate**
Indoors/Carpet	0.3	14.2	0.26
Outdoors/Grass	2.2	4.7	0.41
Outdoors/Asphalt	4.3	5.8	0.49
Outdoors/Ice	3.5	10.5	0.43

Two reasons of using uncompressed over compressed video are given: firstly, it is because of the compatibility between the embedded video system and camera via a high-speed interface; secondly, it is due to the result of degraded performance on some vision algorithms from the use of compressed data.

Campbell et al., 2005 [21] pursue the previous work on the same basis of the falling price of video cameras. Due to the fact that a video camera can capture abundant information, it should be able to perform multiple sensory operations at the same time as well. Two types of cheap cameras, USB and IEEE1394, are used in two types of experiment, closed-loop and open-loop tests. The open loop tests are performed at 7.5 fps by the IEEE1394 camera. The experiment is performed based on the previous work. The closed loop task, however, is run on a USB camera capable of running at approximately 10 fps. Since the open loop tests are similar to the previous work, only the closed loop experiments will be reviewed here.

In [21], the Open Computer Vision Library (OpenCV) [67] is used in a standard computer vision algorithm throughout this discussion. The estimate of the optical flow field is performed by an improved version of the Lucus Kanade algorithm [68]. A low corner threshold is chosen to track selected features. The outliers are screened out based on the smoothness of observed tracking patches. As used in the previous work, the optical flow vectors are divided into two main parts: sky and ground with an ignored horizon zone. A median filter is used to estimate the observed angular displacement in the sky region. The x-displacements and y-displacements in the ground region are median filtered to estimate the translation along the side-ways and forward/backward directions

respectively. The results from the closed-loop tests reveal the cumulative error (Cartesian distance as % of distance traveled) at 7.1%. Nevertheless, it is also reported that the total latency in the control loop, obtaining the images, processing them, and commanding the robot, is approximately 300 ms. As for performing obstacle and precipice detection, the recovery of the depth in the video scene is not utilized. Instead, it relies primarily on the discontinuities in the optical flow field in the image's sub-regions to detect the potential hazards. The median optical flow field direction and velocity are independently computed for each region. The longer vector, positive violation, signals a detected obstacle while the shorter vector, negative violation, flags a detected precipice.

In conclusion, this chapter discusses the techniques of the motion estimation in a video sequence. The direct method and indirect method describe how a motion flow field is created from a pair of two consecutive video frames. Afterward, the camera motion models are given, which leads to their deployment on mobile robot navigation. The principal methods are pixel-based and block-based techniques are discussed according to previous work by various researchers.

CHAPTER IV

HARDWARE SPECIFICATION AND SETUP

The ultimate goal in this work is to develop a visual odometry system and a precipice detection based on MPEG encoding technology. A camera is mounted on the top of the front part of a mobile robot, and the image sequence captured from the camera is fed to an MPEG encoder. The software system, then, utilizes the motion field provided by the MPEG encoder to analyze the robot's egomotion. The robot's localization can be estimated from the operation. In addition, the MPEG motion field is also used to perform precipice detection. If a precipice is detected successfully, the robot needs to stop moving or perform other actions to avoid damage.

This chapter focuses mainly on the hardware components used to build the system. The hardware specifications and their performance will be discussed. The hardware setup will be described as well, as it is one of the vital parts for maximizing system performance. Some software issues are taken into consideration in order to help in selecting proper equipment and minimizing potential problems and difficulties. The main hardware components utilized in this system include a mobile robot, a camera unit, and a laptop computer. The details of each component will be given in the following sections.

Mobile Robot

The system is designed and implemented based on a Pioneer 2AT mobile robot [69]. The robot is equipped with 8 forward and 8 rear sonar sensors. It is driven by 4 motors, uses skid steering, and has wheel encoders. The robot microcontroller communicates with an external computer via a serial port (RS-232). Figure 4.1 depicts the Pioneer 2AT mobile robot used in the work.

Figure 4.1: Pioneer 2AT mobile robot.

Camera Unit

In this work, a Logitech USB QuickCam Express is utilized and the sourcecode driver for Linux distribution can be downloaded from [70]. This camera model is one of the cheapest sets available in the market at an approximate price of $25. It is capable of capturing RGB images at about 10 frames per second with somewhat decent quality. The camera parameters necessary for the implementation are shown in Table 4.1. Since the MPEG encoder requires the YUV420 picture format, the captured RGB buffer of the camera driver must be converted before feeding to the encoder.

Table 4.1: Camera parameters.

Camera Parameters	
Focal Length (f)	50 cm
Horizontal View Angle (β)	35.5 Degree
Vertical View Angle (λ)	27 Degree

Camera Setup

The fundamental objective of the camera setup is to try to eliminate as many unknown variables as possible. By establishing constraints on fixing the camera angle and running the robot on fairly flat surfaces, for instance, the coordinate system for 3D points on the floor can be recovered. Since the distance to the points on the floor is known, a lookup table that maps the 2D points on the image plane onto the 3D points on the floor can be created. This pre-calculated table can tremendously reduce the computational time during the real-time operation.

For this system, the camera is placed in the middle of the front side of the Pioneer robot at which the center of the projection, *h*, is located at 31 cm above the floor. The camera is tilted down so that the point in the center of the image plane is the projection of a point on the floor located about 100 cm, *d*, horizontally away from the camera. Figure 4.2 displays the side view of the camera setup on the robot.

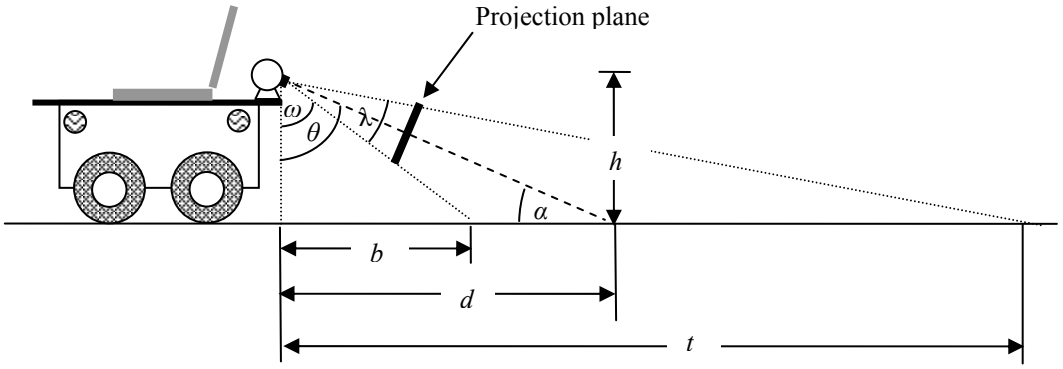


Figure 4.2: Side view of camera setup.

From the figure, the tilt angle, θ, can be calculated from Eq (4.1) and α can be computed from Eq (4.2).

$$\theta = \tan^{-1}\left(\frac{d}{h}\right) \qquad \text{Eq (4.1)}$$

$$\alpha = 90 - \theta \quad \text{or} \quad \alpha = \tan^{-1}\left(\frac{h}{d}\right) \qquad \text{Eq (4.2).}$$

The distance of the bottom line in the image plane to the front of the robot, *b*, can be obtained by Eq (4.3).

$$b = h \tan(\omega) \qquad \text{Eq (4.3)}$$

where $\omega = \theta - \left(\frac{\lambda}{2}\right)$ and λ is the vertical viewing angle of the camera.

The distance of the top line in the image plane to the front of the robot, *t*, can be computed as Eq (4.4).

$$t = h \tan(\omega + \lambda) \qquad \text{Eq (4.4).}$$

Given the above constraints and parameters, the projected area on the floor can be displayed as in Figure 4.3. While the robot is moving, motion in the image plane will reflect motion in the shaded area.

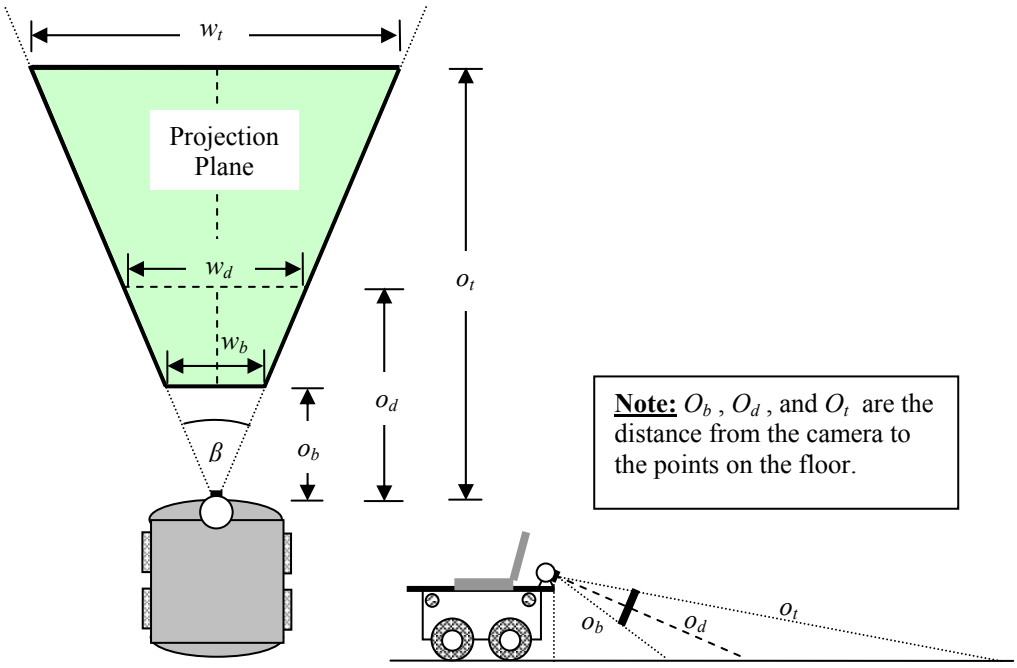


Figure 4.3: Top view of camera setup (floor region projected to the image plane is shaded).

The distance from the center of the camera to the bottom projected line, o_b, middle projected line, o_d, and top projected line, o_t, can be calculated from Eq (4.5), Eq (4.6), and Eq (4.7), respectively.

$$o_b = \frac{b}{\sin\omega} \quad \text{or} \quad o_b = \sqrt{h^2 + b^2} \qquad \text{Eq (4.5)}$$

$$o_d = \frac{d}{\sin\theta} \quad \text{or} \quad o_b = \sqrt{h^2 + d^2} \qquad \text{Eq (4.6)}$$

$$o_t = \frac{t}{\sin(\omega + \lambda)} \quad \text{or} \quad o_b = \sqrt{h^2 + t^2} \qquad \text{Eq (4.7).}$$

The width of the bottom (w_b), middle (w_d), and top (w_t) projected lines can be obtained from Eq (4.8), Eq (4.9), and Eq (4.10), respectively, where β is the camera's horizontal viewing angle.

$$w_b = 2 \times \left(o_b \ \tan\left(\frac{\beta}{2} \right) \right) \quad \text{Eq (4.8)}$$

$$w_d = 2 \times \left(o_d \ \tan\left(\frac{\beta}{2} \right) \right) \quad \text{Eq (4.9)}$$

$$w_t = 2 \times \left(o_t \ \tan\left(\frac{\beta}{2} \right) \right) \quad \text{Eq (4.10).}$$

Vanishing Point on Virtual Image Plane

Based on the Perspective model [14], while the observer is moving forward, every point on the image plane tends to expand around the point where the direction of the observer is projected onto the image plane, called the *Focus Of Expansion* (FOE) or the *vanishing point*. On the other hand, if the observer is moving backward, the vanishing point will be at the same location but every point will, rather, converge to the vanishing point, called the *Focus Of Contraction* (FOC). Therefore, with all the setup and parameters given above, while the robot is moving forward or backward, the vanishing point on the image plane can be computed from Eq (4.11). (*Note that the location of the vanishing point is made based on the ideal assumption of a noiseless signal.*)

$$v_y = f \tan(\alpha) \qquad \text{Eq (4.11)}$$

and $v_x = 0$.

Since it is assumed that the robot is performing a pure translation (i.e., moving straight ahead or behind), the ideal vanishing point should then be located at 0 on the X axis ($v_x = 0$). We note that v_y is given in centimeters from the origin along the Y axis on the virtual image plane. Consequently, this value is mapped to the point where it should appear on the image plane. An example of the ideal vanishing point caused by the robot translation is given in Figure 4.4.

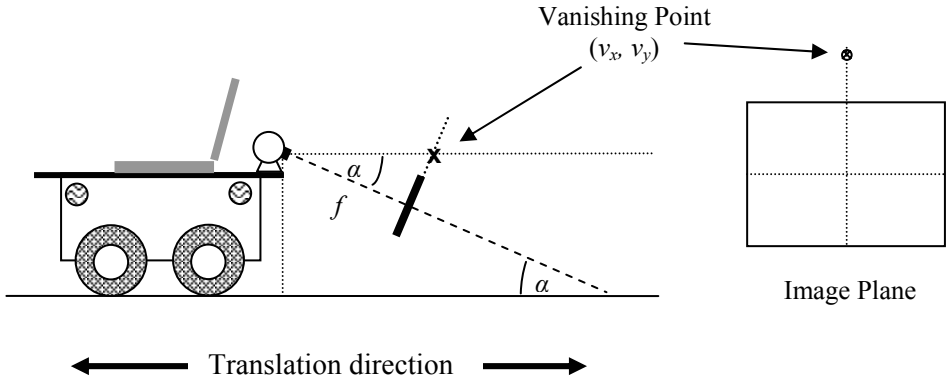


Figure 4.4: Vanishing point location.

Table 4.2 lists the calculated parameters from the camera setup used in this system. Note, however, that these values are approximate. Error may be caused by inaccurate measurement or camera distortion.

Table 4.2: Essential parameters in the robot setup.

Height of camera from the floor (h)	31 cm
Pitch Angle (θ)	72.78 deg
Distance from robot to the point on the center of the image plane (d)	100 cm
Distance from robot to the bottom line of the image plane (b)	52.17 cm
Distance from robot to the top line of the image plane (t)	476.79 cm
The width of the projection area at the bottom of the image plane (w_b)	38.85 cm
The width of the projection area at the middle of the image plane (w_d)	67.02 cm
The width of the projection area at the top of the image plane (w_t)	305.89 cm
Vanishing point [v_x v_y]	[0 15.5] cm*

** The mapping ratio at f is 1mm:1px. Therefore, v_y is approximately 155 pixels above the origin. Since the screen dimension is 320x240, v_y is then located about 35 pixels above the top line of the screen.*

Computer Unit

The Pioneer robot used in this system does not have a built-in computer installed. Instead, an external laptop computer is used by strapping it on top of the robot. Since the robot is shared by a number of staff in the lab, using external laptop computers is found to be more effective than having a built-in computer. Firstly, in case of computer upgrade needed, it is much easier to replace a new external computer than the built-in. A built-in computer is comparatively more expensive and may need upgrades yearly. Secondly, the use of the built-in computer is limited solely to the pioneer robot. It is not convenient to employ it in other tasks. Lastly, it is more troublesome to deploy a new application into the built-in computer. By using a personal laptop computer, one can develop an application or robot simulation on his/her own computer. Once, everything is ready to operate on the real robot, this can be done simply by connecting the laptop computer to the robot via the serial port (RS-323) connectors.

In this work, the software systems are developed mainly on a COMPAQ Presario V2000. The computer runs on an Intel(R) Pentium(R) M processor at 1.86 GHz with 1GBs of RAM. Since this work heavily requires access to numerous software sourcecodes, a Linux operating system is chosen due to the fact that many opensource projects are provided for the Linux platform. Because, this computer model does not have a serial port installed, a Keyspan USB Serial Port is used.

The software systems developed from the COMPAQ computer were ported to a DELL Precision M70 laptop computer during the experiments. The DELL laptop runs on an Intel(R) Pentium(R) M processor at 2.13 GHz with 1GBs of RAM. The main reason for transferring the applications to the DELL computer was not due to performance issues. Rather, it is because the DELL computer is equipped with a large battery pack that can last almost 4 hours from a full charge. It is very helpful in extending the operational times during the experiments. Nevertheless, both computers are running Fedora Core 4 Linux [71].

This chapter generally discussed the hardware specification and the equipment setup. The next chapter will focus specifically on the software issues in developing the system. It will describe the software design and the algorithms employed to develop the system.

CHAPTER V

SOFTWARE SYSTEM DESIGN AND IMPLEMENTATION

In this chapter, a software system design based on the hardware systems and specifications given in the previous chapter is discussed. An opensource software package for an MPEG encoder that is capable of performing in real-time is needed. The chapter starts with how to create a motion field out of the MPEG encoder's motion vectors. The approaches to visual odometry, outlier elimination, and the precipice detection will be given, respectively.

Creating Motion Field from MPEG Encoder

An equivalent optical flow field, the motion vectors field (MV Field), can be constructed from the MPEG motion vectors. The motion estimation module in the MPEG encoder calculates the movement for each macroblock and yields a motion vector representing the motion of that particular macroblock. For a picture encoded with the P-frame type, a macroblock without a motion vector may indicate either no motion or no matched region found, while a picture with the I-frame encoding will not produce any motion vectors at all. Consequently, the encoded frame type becomes necessary for the motion analysis as well. Thus, the motion estimation module must be modified in such a way that the (x, y) components of the motion vectors and the encoded frame type are accessible. Then, the analysis of the robot's egomotion can be proceeded.

Approaches to Visual Odometry

Once the MV field has been created, the robot's egomotion can be determined. The motions under consideration include translation (FORWARD and BACKWARD), rotation (TURN LEFT and TURN RIGHT), and no motion (STOP). In this work, only a single motion will be calculated, one at a time, assuming that no two simultaneous motions occur during the operation in a particular frame. Once the MV field has been obtained, the travel distance or the turning degree will be computed after the erroneous motion vectors have been filtered out. No operation is performed if the egomotion has been categorized as STOP. A flowchart of this operation is given below in Figure 5.1.

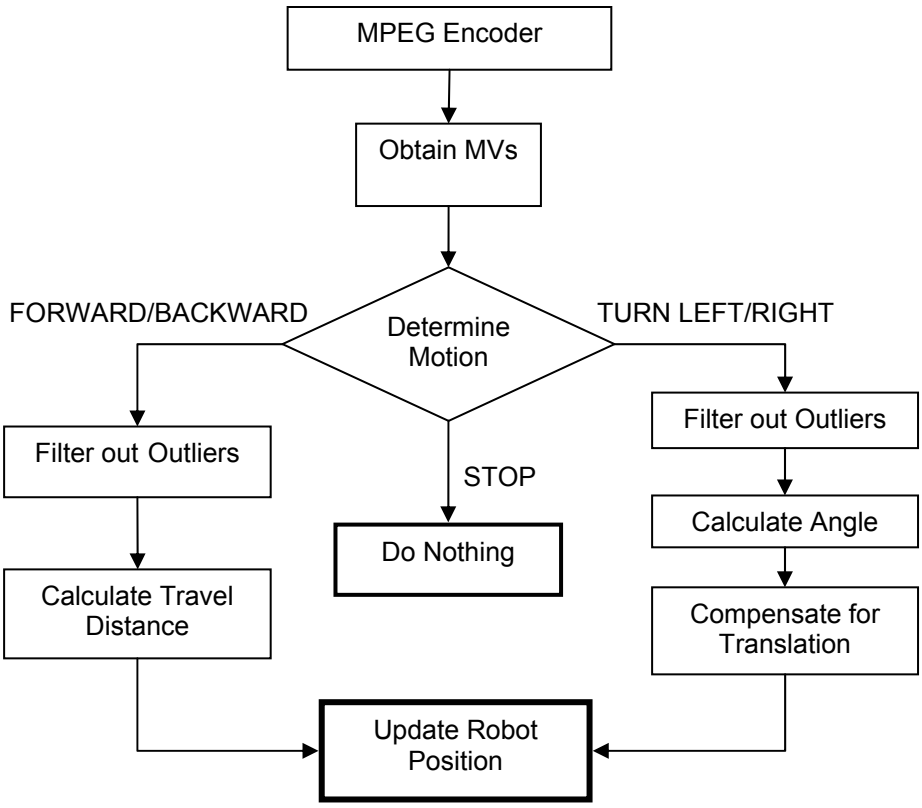


Figure 5.1: Flowchart of real-time visual odometry.

How to Determine Robot Motion

Based on the perspective model, the distance in the 3D environment is inversely proportional to the distance in the image plane. From the camera setup explained earlier, when the robot performs forward or backward translation, the motion vectors in the upper portion of the image plane will have less effect than those in the bottom. Therefore, only motion vectors in the lower portion, which also are less sensitive to noise, will be used in the motion evaluation. In this work, all motion vectors in the evaluated area will be projected onto the sides of the $2V_f$ x $2V_f$ virtual square shown in Figure 5.2. The motion vectors that project onto the left and right sides of the virtual square will be used to estimate the robot rotation LEFT and RIGHT respectively. It is important to note that the top side is used for both the robot translation FORWARD and BACKWARD and the bottom side is not utilized for BACKWARD. This is because translation in the backward direction will cause the motion vectors to diverge from the vanishing point, FOC, as explained earlier. Therefore, a motion vector having a negative y component will be flipped into the opposite direction so that its projection will point towards the vanishing point. Then, it is tested to see if it projects onto the top side of the square. If it does, this motion vector belongs to the motion BACKWARD. Otherwise, this motion vector is part of either the motion LEFT or RIGHT, according to its forward projection.

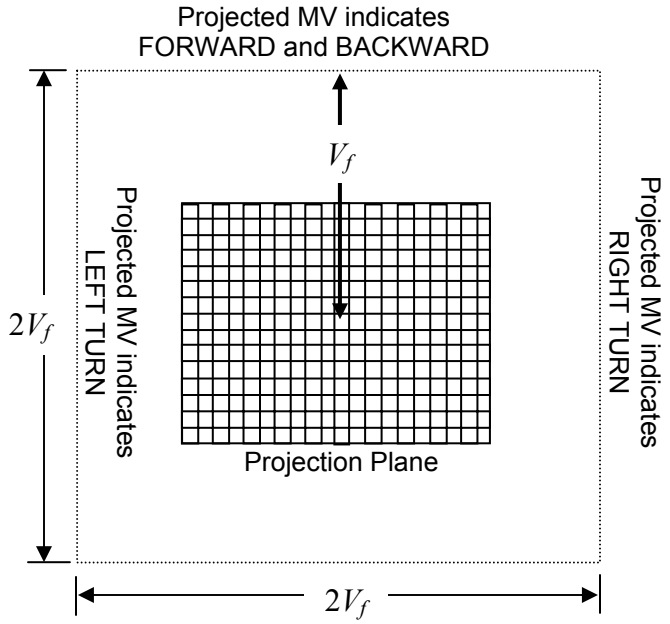


Figure 5.2: $2V_f$ x $2V_f$ virtual square used in classifying robot motion.

After the projection process is completed, the system employs a simple technique of majority vote to finalize the robot motion. If the number of valid motion vectors for the winning motion is below a threshold value, the robot motion will automatically be evaluated as STOP.

Outlier Rejection

Once the robot motion has been determined, outlier rejection can proceed. This process filters out erroneous motion vectors and keeps all legitimate ones for the motion calculation. Figure 5.3 shows the outliner rejection for the robot translation FORWARD and BACKWARD. The process for motion FORWARD starts with calculating the

average heading value of the motion vectors accounted for the forward translation. *(Note that this system does not use this value to compensate for the real heading value due to the fact that the motion vectors produced by the MPEG encoder are too noisy to give an accurate value. It is left to future investigation to obtain a more reliable real heading direction estimate.)* Subsequently, the lower bound and upper bound values of the intermediate threshold values are established for this particular frame. These thresholds may be changed by the user. The values used in this work are given in the next chapter. As a result, motion vectors having projections that lie outside the threshold bounds are rejected. The process of filtering motion vectors for the motion BACKWARD is performed in a similar manner except that the flipped projections of the motion vectors are used instead.

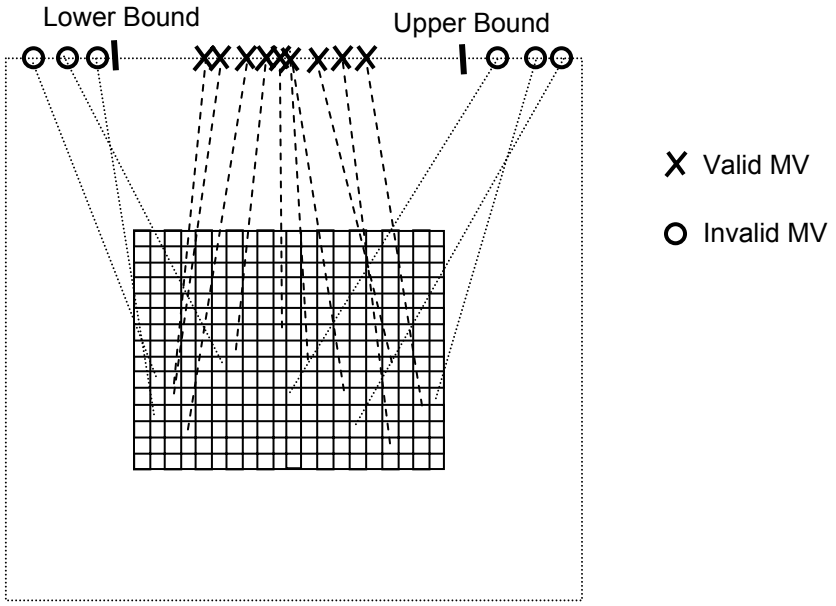


Figure 5.3: Method of rejecting outliers for motion FORWARD and BACKWARD.

The process of eliminating outliers for the motions LEFT and RIGHT, on the other hand, is performed slightly different from that of FORWARD and BACKWARD. Since the direction of the motion vectors belonging to the robot rotation LEFT and RIGHT are actually produced relative to the X axis of the image plane, all motion vectors have to be translated to the X axis of the image plane coordinate before the calculation of the average heading value. Afterward, the process proceeds in a similar fashion to that of FORWARD/BACKWARD. Figure 5.4 and Figure 5.5 visually describe the method of rejecting outliers for the robot motions LEFT and RIGHT respectively.

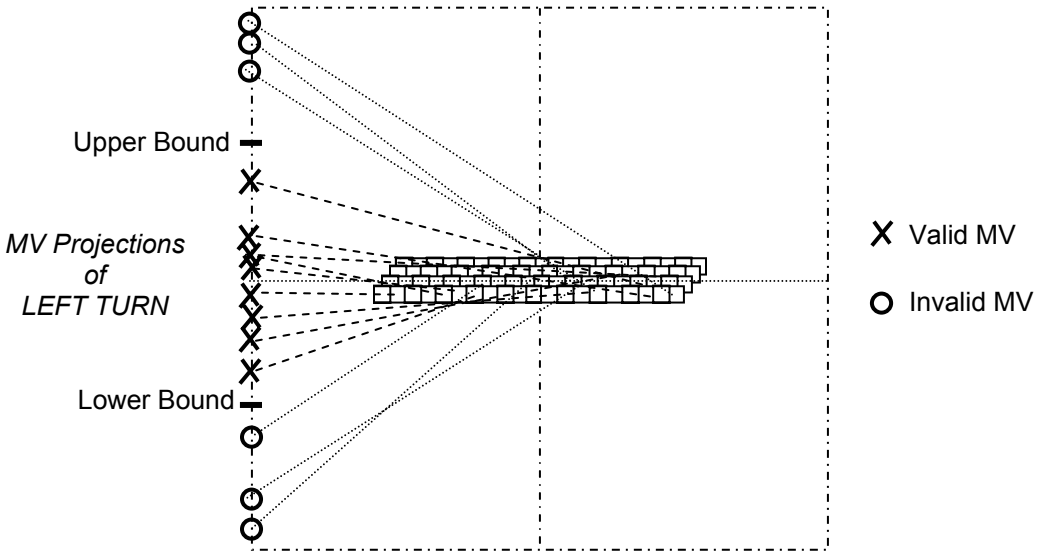


Figure 5.4: Method of rejecting outliers for motion LEFT.

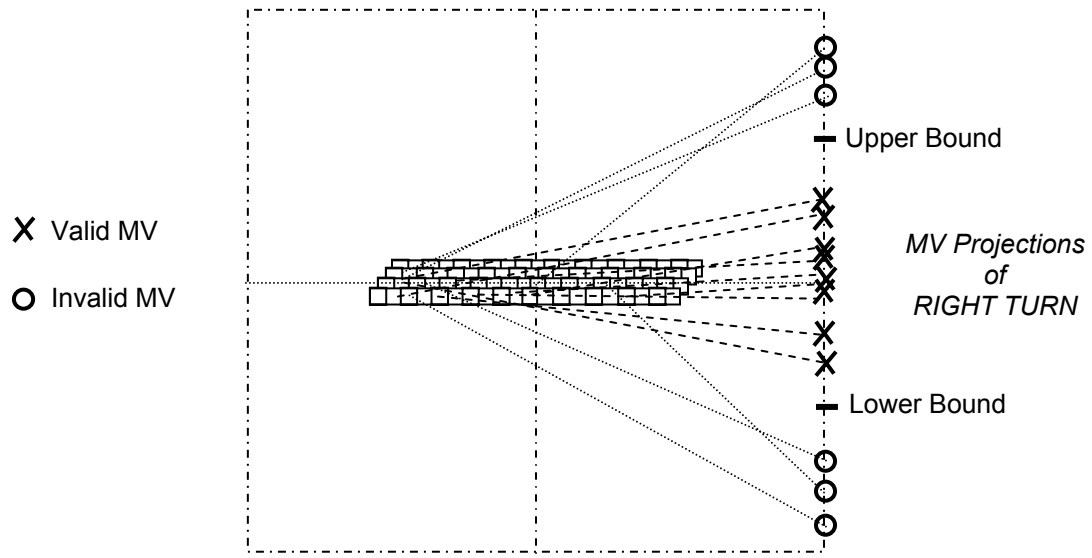


Figure 5.5: Method of rejecting outliers for motion RIGHT.

Calculation of Robot Translation

Since the robot travels on a flat surface with a fixed camera, the mapping of points on the 2D image plane onto the 3D environment becomes static and can be pre-calculated. Thus, the real-time performance of the visual odometry can be improved tremendously by the help of a lookup table. All the y components on the image plane will be pre-computed and stored in a table. Because the motion estimation of MPEG produces motion vectors at sub-pixel resolution, the lookup table of the y components on the image plane is created at the level of sub-pixels as well. Figure 5.6 displays the mapping of line y_i on the image plane onto the distance Y_i on the floor whereas Eq (5.1) explains how to calculate the distance for the lookup table.

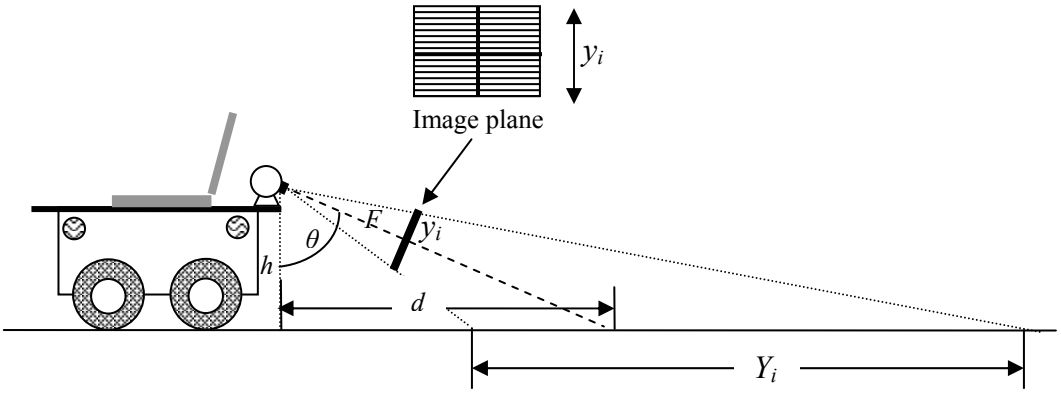


Figure 5.6: Mapping of line y_i on the image plane onto the distance Y_i on the floor.

$$Y_i = h \cdot \tan(\theta + \beta_i) \qquad \text{Eq (5.1).}$$

where $\beta_i = \tan^{-1}\left(\frac{y_i}{F}\right)$ at which y_i is given in subpixel units ranging from the top to the bottom line with the coordinate origin located at the center of the image plane.

Once the lookup table has been created, an intermediate travel distance for each frame can be obtained simply by calculating the difference of the distance, from the lookup table, between the y location of the macroblock and the y component of the motion vector associated with it. The FORWARD motion gives a positive distance while the BACKWARD motion gives a negative value. In each frame, only valid macroblocks are used in the calculation. The macroblocks in the same row will be passed to a filter,

either mean or median, and the output value will represent the intermediate travel distance for that row. Finally, the resulting values for all rows will be sent to the filter, mean or median, again to compute the final intermediate travel distance.

Calculation of Robot Rotation

When the robot performs a rotation, only the *x* component of the motion vectors is used to calculate the intermediate angle. As discussed earlier, since points in the environment at different distances yield the same effect on the image plane, all motion vectors can be used in computing the turn angle. However, the motion vectors in the upper part may be corrupted by other moving objects, at least more so than those in the bottom part. Thus, only valid motion vectors in the lower part will be used in the calculation as well.

With the camera mounted in front of the robot in lieu of being placed on the center of rotation, the robot rotation will simultaneously cause a translation along the X axis as well. As a result, the *x* components of the motion vectors caused by the robot rotation become larger than the value given by the real robot rotation as can be seen in Figure 5.7. Thus, the translation associated with the robot rotation must be excluded. Equations 5.2, 5.3, 5.4, and 5.5 explain how to obtain the pure robot rotation.

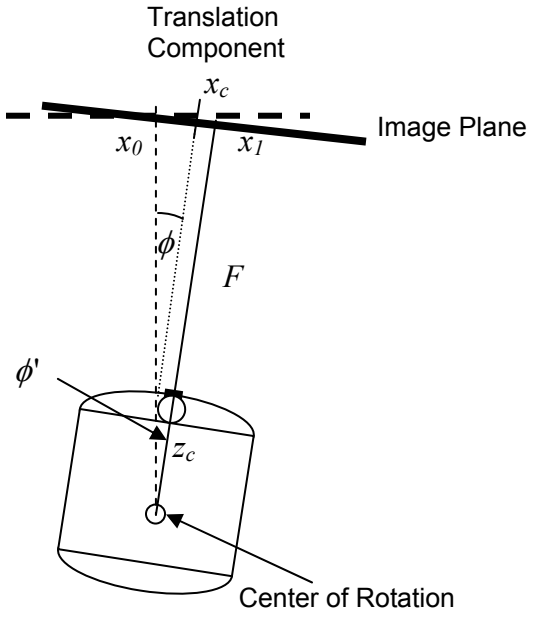


Figure 5.7: Robot rotation.

The angle with the effects of translation is given by

$$\phi' = \tan^{-1} \frac{(x_0 - x_1)}{(F + z_c)} \qquad \text{Eq (5.2)}$$

where ϕ' = angle with translation,

x_0 = point before rotation (x component of motion vector)

x_1 = point after rotation, which is the center of the macroblock,

x_c = camera translation occurring during the robot rotation,

F = focal length,

z_c = length from the center of camera to the center of the robot rotation.

The translation component, x_c, can be obtained from:

$$x_c = z_c \tan(\phi') \quad \text{Eq (5.3).}$$

or it can easily be obtained from:

$$x_c = z_c \frac{(x_0 - x_1)}{(F + z_c)} \quad \text{Eq (5.4).}$$

Therefore, the pure robot rotation, ϕ, can be calculated by:

$$\phi = \tan^{-1} \frac{(x_0 - x_c)}{F} \quad \text{Eq (5.5).}$$

During the calibration process, it has been discovered that the center of rotation is not located in the center of the robot. The approximate z_c is given about 34 cm away from the center of the camera. The resulting x_c, based on the equations above, contributes about 40 percents to the x component of the motion vectors. However, in order to reduce the computational time, the intermediate angle for each valid macroblock is computed by mapping the motion vectors, x, to the pre-calculated angle. Given the horizontal viewing angle of 35.5 degrees with the screen width of 320 pixels, 1 pixel is assumed to be equal to 0.1109375 degree. Since the motion vectors are in sub-pixel resolution, they need to be

divided by 2. In addition, the resulting angle is a combination of robot rotation and translation. The translation component has to be eliminated. Then, the final intermediate turn angle for each frame is the result of the mean or median filter applied in the same manner as in the calculation of robot translation, FORWARD and BACKWARD.

Approach to Precipice Detection

In using the camera to detect a precipice, the same setup used for visual odometry is employed. This also supports the idea of using a single camera to perform multiple tasks. Thus, the motion vector field fed to the visual odometry system will be used here. In this scenario, because the motion vectors obtained from the MPEG encoder are considered to be noisy and may not represent the real motion, the more robust motion vectors in the lower portion of the screen are then divided into three rows as depicted in Figure 5.8. Each row contains 5 patches. Each patch is 4x3 macroblocks. However, the patches in the bottom row are 4x2 macroblocks. This has two reasons: one is that the number of rows in the active portion is not a factor of 3 and other is that the motion vectors in the bottom rows seem to be less noisy than those in the upper part.

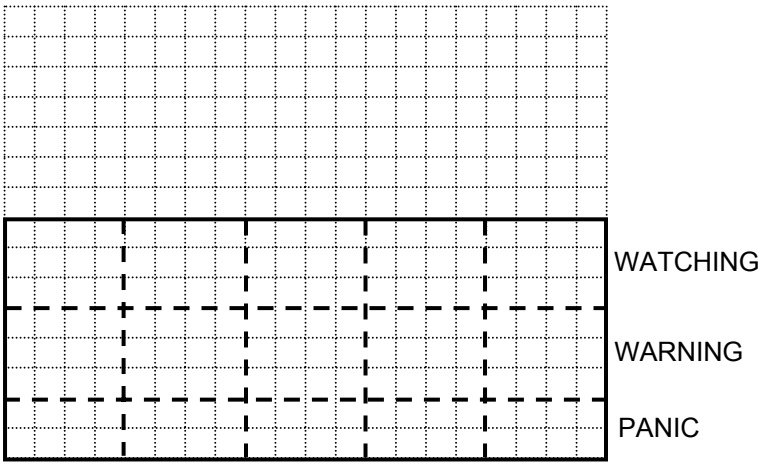


Figure 5.8: Motion vectors are grouped into patches.

From the figure, each row is labeled as WATCHING, WARNING, and PANIC, respectively. The WATCHING row implies that the robot is likely to be approaching a precipice. The WARNING row implies that the robot is about to reach a precipice. The PANIC row indicates the precipice in very near and the robot should stop.

The process of detecting a precipice will be performed on the robot motion FORWARD only. Assuming that there is currently no precipice in the detected range of the robot (about half a meter in front of the robot), the process of detecting a precipice is explained as follows:

- After retrieving the filtered motion field, each patch computes the mean/median value. (*Only macroblocks with motion vectors not equal to zero are used. Macroblocks with no motion vector will be disregarded*.) The resulting value of

each patch will be quantized to 0 or 1. In this case, if the resulting value is less than half of the average value of the bottom row, then this patch yields 0, otherwise 1.

- If more than half of the number of patches in a particular row is zero, which is three in this case, it will be marked as invalid. (*This can be done by using a median filter if the number of patches in a row is odd.*)
- The detection process starts on the upper row, WATCHING, by observing the row status over the past 3 frames. If it has been invalid for 3 consecutive frames, it will be flagged with WATCHING alert.
- The WARNING alert is perform in the same fashion as that in the WATCHING except that the WARNING alert will be flagged if the WATCING is also flagged.
- At a current frame, if the WARNING row indicates a detected precipice, the PANIC alert will be flagged once the number of valid patches in the PANIC row is less than half.

However, the status of the invalid WATCHING and WARNING rows can be recovered if its status becomes valid for 3 consecutive frames. From the algorithm above, the detection of the PANIC state is treated differently. Instead of observing its status over 3 consecutive frames, the PANIC status will be checked in every frame. The WATCHING state is not used in this case because the robot may be approaching a trench, which has a short drop-off, about a foot long, with a normal surface ahead. In this case, the WATCHING row may be recovered by the time the trench has been discovered in the

PANIC row. This precaution procedure is very important in order to protect the robot from damage.

The approaches to the visual odometry and precipice detection given here rely extensively on the MPEG motion estimation. The equivalent optical field is created from the motion vectors in the P-frame. This technique is performed only in the software without any assistance from external hardware at all. The system is designed in accordance with the availability of an opensource software project. Thus, the Linux operating system seems to be a good fit in creating the software systems. The implementation of the systems given in this chapter will be explained in the next chapter.

CHAPTER VI

SOFTWARE IMPLEMENTATION

The main objective of this chapter is to describe the software implementations of the real-time MPEG visual odometry and precipice detection. The software systems are written in C++ under the Fedora Linux [71] environment. The major reason for selecting the Linux operating system is mainly because of the support from the opensouce community. The availability of the sourcecode, in particular, makes it possible to customize the software application as needed. Figure 6.1 depicts the overall system in this work, which generally consists of major subsystems including Video Frame Grabber, RGB2YUV converter, MPEG Encoder, Visual Odometry, and Precipice Detection.

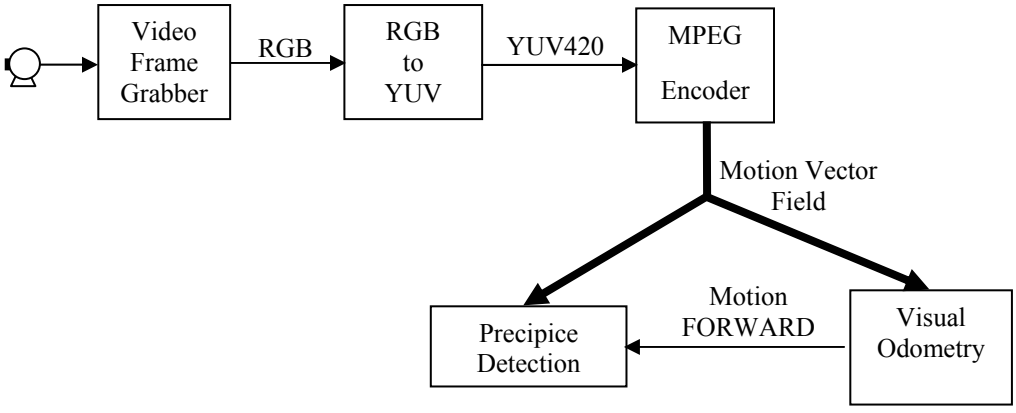


Figure 6.1: Overall system.

In this system, a USB Logitech Quickcam Express is attached to the Pioneer mobile robot. Its generic driver for Linux distributions can be downloaded from the *sourceforge projects* website [70]. The driver requires a kernel version 2.2.18 (or higher), kernel 2.4.x, or kernel 2.6.x with Video for Linux (V4L) support. It also provides infrastructure interface functions such as grab image, image quality adjustment, adaptive brightness/contrast adjustment, etc. The Video Frame Grabber is responsible for capturing an image sequence into an RGB buffer. The RGB2YUV module then converts the RGB picture format in to a YUV420 format so that it can be fed into the MPEG encoder.

As described earlier, an MPEG encoder generates motion vectors in a P-frame based on the motion of each macroblock relative to the image contents in the previous I- or P-frame. The MPEG motion estimation especially calculates the motion for each macroblock and packs the motion vectors together with other information in the MPEG data stream. Unfortunately, the MPEG encoder does not export these motion vectors to the external module outside the MPEG encoder system. With help from the opensouce *FFMPEG* project [72], the retrieval of these motion vectors becomes possible. In this case, the sourcecode of the MPEG encoder is modified by inserting three integer pointers into the *AVMPEGContext* Struct so that the (x, y) components of the motion vectors and the encoded picture type in the motion estimation module can be accessible. Then, an equivalent optical flow field can be constructed. Eventually, this motion vector field can be distributed to the visual odometry and precipice detection systems.

In order to reduce the complexity of the software implementation, the two applications, the Visual Odometry using MPEG Technology and the Real-time Precipice

Detection, are created separately. As a result of doing so, the data analysis from the experiments will be pertinent to each application since the data collected from one application will not interfere with the other. How both applications are constructed will be explained in the following sections.

Visual Odometry

The main functionalities of the Visual Odometry system are evaluating the robot motion from the motion vectors field at a particular frame, rejecting outliers based on the motion detected, and calculating the robot translation or rotation according to the motion detected. A simplified version of an interface for the class CVOdometry is given in Figure 6.2.

```
Class CVOdometry
{
        //Initialize VOdometry
        initVOdometry(f, h, d);
        initMacroblockLocation();
        initDistanceLookupTable();
        resetVisualOdometry();

        //Methods for processing robot motion
        evaluateMotion(mvx, mvy);
        filterMVs(); //Outlier Rejection
        calculateMotion();

        //Retrieve travel distance
        double getIntermediateDistance();
        double getAccumulatedDistance();
        //Retrieve Heading Angle
        double getIntermediateAngle();
        double getAccumulatedAngle();
};
```

Figure 6.2: A simplified interface class of CVOdometry.

The class methods are divided into three main parts: initialization, motion processing, and data retrieval. The initialization part takes the parameters in accordance with camera specifications, mounted angle and position, to compute the vanishing point on the image plane. For this initialization, three parameters are needed: f (camera focal length), h (the height measured from the floor to the center of the camera), and d the horizontal length measured from the camera to the point where the optical axis intersects with the floor. Subsequently, the camera tilt angle and vanishing point can be obtained from the calculation. These values are, in turn, used to create a lookup table in the initDistanceLookupTable(). The initMacroblockLocation() method calculates the (x, y) location of all macroblocks while the resetOdometry() method simply resets all intermediate and accumulated values.

The pivotal part of the class CVOdometry is the motion processing unit. The main functionality of this unit is to evaluate the motion vector field obtained from the MPEG encoder. The output of this operation indicates one of the given robot motions: FORWARD, BACKWARD, TURN LEFT, TURN RIGHT, and STOP. The resulting robot motion, then, defines which outlier rejection algorithm to be used in the filterMVs() method to eliminate erroneous motion vectors. Finally, the filtered motion vector field is used to compute the robot translation or rotation as an intermediate value for a particular frame as well as accumulated values for the entire sequence. The last part of this class provides interfaces to access the intermediate and accumulated distance values and angles.

Application for Real-Time Visual Odometry

Figure 6.3 shows the Graphical User Interface (GUI) of the real-time visual odometry application. The application was written in C++ and the GUI part utilizes the GTK-MM library [73], which is a library for C++ interface.

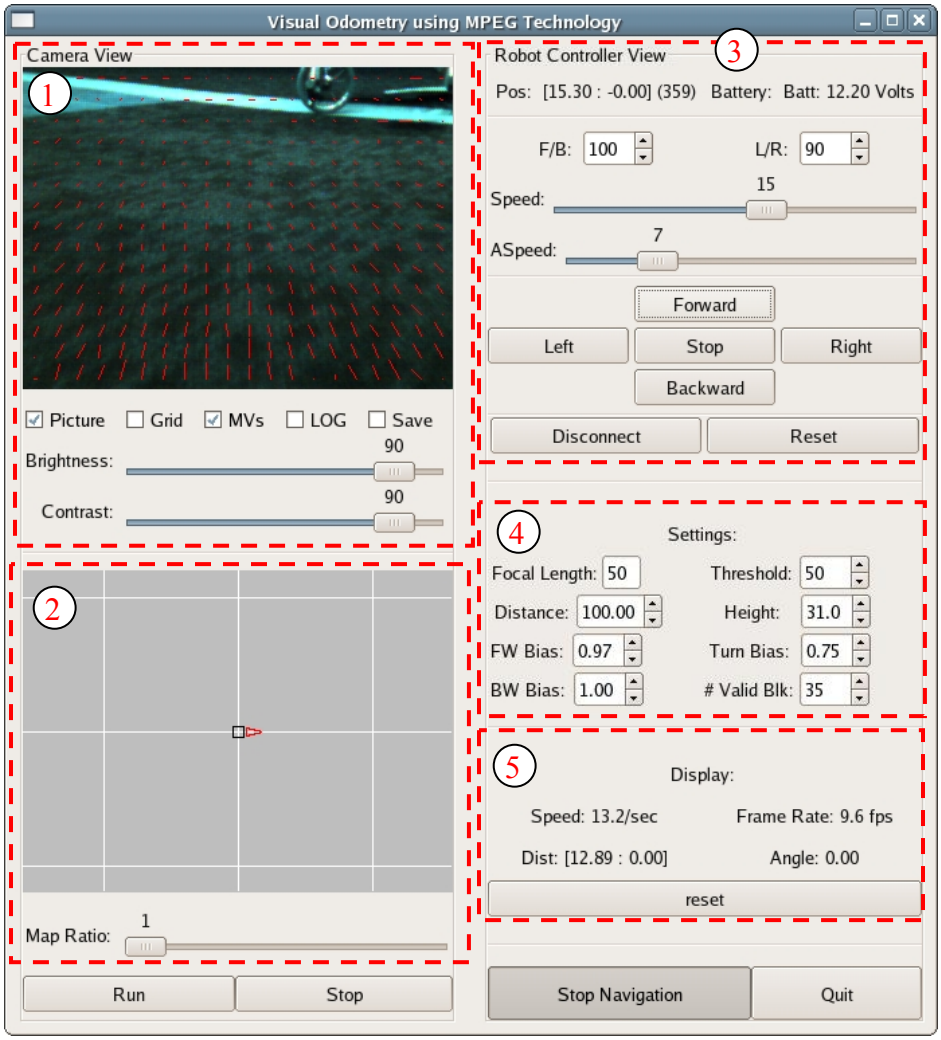


Figure 6.3: A GUI of the real-time visual odometry application.

The application is comprised of 5 main portions: 1.) camera view, 2.) map view, 3.) robot controller view, 4.) setting view, and 5.) display view. The camera view is capable of drawing the image sequence, grid (macroblock layout), and motion vectors. It also provides brightness and contrast adjustable slider bars. The map view graphically displays the robot localization in the Cartesian coordinate relative to the robot origin. Each grid is 100x100 cm^2 in dimension. The adjustable slide bar is responsible for changing the map ratio. Thirdly, the robot controller view provides a user interface to interact with the Pioneer 2AT robot. In addition, it displays the robot position, and reads the wheel encoders and the battery level in numeric format. The setting view requires the user to enter all parameters necessary in creating the distance lookup table and calculating robot motion, which has been described above. Lastly, the display view shows the calculated outputs of the visual odometry, robot translation speed, and video frame rate in numeric format.

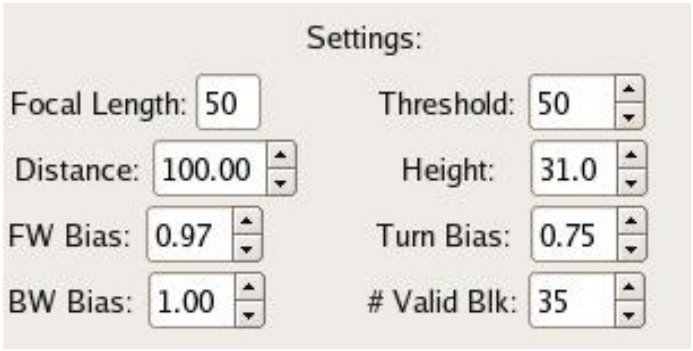


Figure 6.4: Setting view.

The setting view in Figure 6.4 is one of the most significant parts of the application. The accuracy of the visual odometry computation is dependent heavily upon the parameters given to the system. As previously described, the focal length, distance, and height are used to create the distance lookup table. It is where the mapping of the 2D points on the image plane to the 3D points on the floor takes place. The calculation of the robot motion FORWARD and BACKWARD employs these pre-calculated values to produce the accumulated travel distance. The threshold entry box is the value of the lower and upper bound used to eliminate outliers. The threshold value is given in pixel units at which the lower bound is the left value from the average projection value computed from all potential motion vectors, while the upper bound indicates the value to the right. The forward bias and backward bias are the calibrated values for the robot translation FORWARD and BACKWARD, respectively. The turn bias, on the other hand, is the calibrated value for the robot rotation LEFT and RIGHT. The number of valid block is the lowest count of the filtered motion vectors after the determination of the robot motion at a given frame. If the counted number is less than the threshold value, the robot will be considered not moving. The numbers given in Figure 6.4 are the optimal values used in the experiments.

Application for Real-Time Precipice Detection

The GUI of the real-time precipice detection is shown in Figure 6.5. The window is divided into two parts: camera view and detection view. In this application, the robot velocity and the frame rate adjustable slide bars are added in the camera view so that the

experiments can be performed more conveniently. In the detection view, the only main components used during the experiments are the detection status and the status bar (three vertical bars next to the graphs). The top bar indicates the WATCHING status if a potential precipice is detected about a meter in front of the robot. The middle bar indicates the WARNING status when the potential precipice approaches the robot at approximately 70-80 centimeters in front of the robot. The bottom bar indicates the PANIC status and will be flagged when the potential precipice is confirmed. In this scenario, the robot will be stopped immediately to protect it from potential damage.

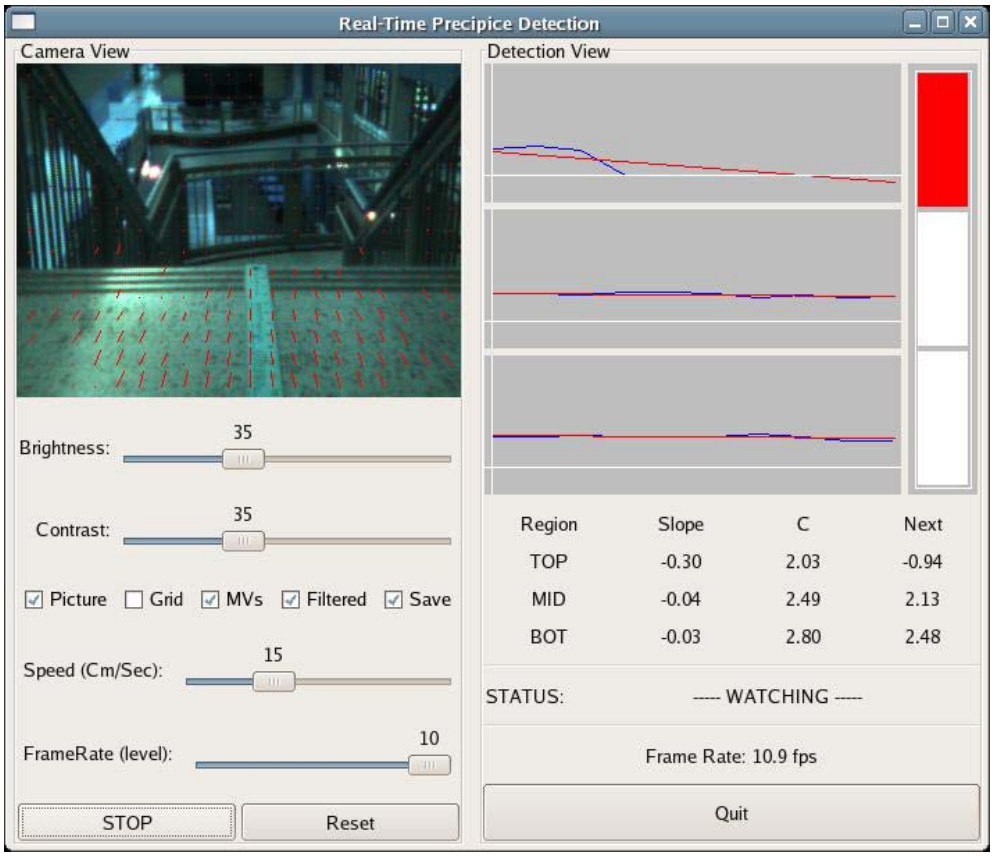


Figure 6.5: A GUI of the precipice detection.

From the figure, the graph plotters and data table are supplementary. Each graph plotter draws the history of the robot velocity calculated from filtered motion vectors for each region over the past 10 frames. The values are then plotted together with the fitting line computed from those values using the numerical method Linear Regression technique by [74]. The equations are given in Eq (6.1) and Eq (6.2), respectively.

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \qquad \text{Eq (6.1).}$$

where

$A_{1,1} = L$ (Number of observed data)

$A_{1,2} = \sum_1^L x_i$

$Z_1 = \sum_1^L y_i$

$A_{2,1} = \sum_1^L x_i$

$A_{2,2} = \sum_1^L (x_i)^2$

$Z_2 = \sum_1^L x_i y_i$

a = constant term

b = slope of the fitting line.

Therefore the linear regression function of *g(x)* is a representation of Eq (6.2).

$$g(x) = a + bx \qquad \text{Eq (6.2).}$$

The data table below the graph plotters displays the parameter from Eq (2), slope and constant, at a current frame in numeric format. In addition, it computes the predicted velocity, based on the fitting function, of each row for the next frame as well.

Nevertheless, the linear regression part is not used in the data analysis from the experiments at all. It is used primarily as an observation tool in detecting a precipice. This may become very useful in developing a new technique to detect a precipice for future work.

One important issue needed to be discussed here is the operation on the I-frames. In this implementation, the MPEG encoder is set to enforce the coding of the I-frame at every 30 frames. In case if there is no significant change occurring in a given frame, an I-frame will be coded about every 3 seconds, at which one frame is an incident of 100 milliseconds. Hence, the I-frame occurring during the operation will be assumed to the same motion as the previous frame.

In these software implementations, the application performance, however, may not meet the maximum requirements since some decoration components have to be added so as to support the data keeping during the experiments. For example, both applications are equipped with a SAVE function to capture the necessary data for further analysis in the offline mode. During the operation, all necessary data are written to a text file at every frame. Notwithstanding, both applications are able to operate at approximately 10 frames per second, which is considered to be very efficient based on the hardware specifications.

This chapter explained how the applications are built. The applications run on Linux operating system and are mostly opensource. The next chapter will delineate the results from a variety of experiments. The data analysis and discuss will be given in the chapter as well.

CHAPTER VII

EXPERIMENT AND RESULT DISCUSSION

This chapter focuses specifically on the experiments of the real-time visual odometry and precipice detection under different circumstances and environments. The experiments are performed separately. Each section starts by addressing how the experiments are to be conducted. The experimental environment and setup will be described and the resulting data are shown in graphical and numeric formats. A discussion of the results is given at the end of the section.

Real-Time Visual Odometry

The real-time visual odometry system operates on 4 different types of surfaces: lab, carpet, tile, and corridor. Each surface possesses uniquely different properties. The lab surface is moderately textured. The lighting condition is mostly under control in which the reflective interference is minimal. The odometry calibration of all the experiments is undertaken under this environment using the mean filter. The carpet surface is very highly textured without lighting reflection. The tile surface is fairly low textured. The lighting condition is varying, depending primarily on the location and time when the experiments take place. When performing at night or on a closed area, the lighting condition will be dominated by fluorescence and incandescent light, which generate small glare area. When performing in the day time where it is close to exterior

glass windows, the ambient light from the sun will cause a large area of glare on the floor. Finally, the corridor surface is very glossy low textured and highly reflective. It produces a large amount of specular reflection. The experiments on the real-time visual odometry system will be performed on the robot translation and rotation separately. Figure 7.1 displays the surface types used in the experiments.

a) Lab Surface

b) Carpet Surface

c) Tile Surface

d) Corridor Surface

Figure 7.1: All surfaces used in the experiments of robot translation and rotation.

Experiment of Visual Odometry on Robot Translation

In this experiment, the robot performs the translations, forward and backward, for approximately 100 cm on each surface at different speeds ranging from 5 cm/sec to 25 cm/sec. The robot velocity is increased by 5 and each trial is run 10 times at each velocity. The values read from the visual odometry, using the mean filter, wheel encoder, and tape measurement are recorded at the experiment site. The motion vectors from all experiments are saved into text files for further analysis. In this case, the motion vector text files are processed offline to compute the values that would result from the median filter. Since the robot may stop before or after a 100-cm line, the error percentage of the obtained values will be used in the performance comparison. This is to make sure that the values are compared on the same scale. The standard deviation of the error percentage is then used to measure the accuracy consistency of the system, while the mean of the absolute error percentage is used to analyze the magnitude of error. Examples of typical data from the experiments of the robot translation are given in Figure 7.2 and Figure 7.3, respectively. Figure 7.4 and Figure 7.5 display the graphs of the standard deviation of the error percentage. Figure 7.6 and Figure 7.7 display the graphs of the mean of the absolute error percentage. The numeric data of these values are listed in Table 7.1 and Table 7.2, respectively.

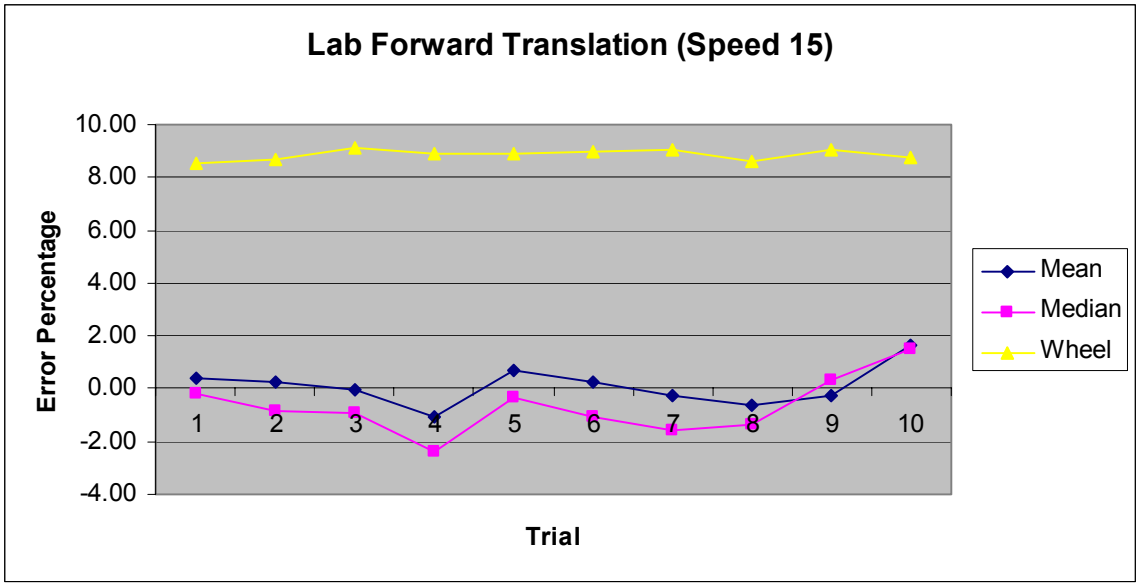


Figure 7.2: Error percentage of forward translation on lab surface.

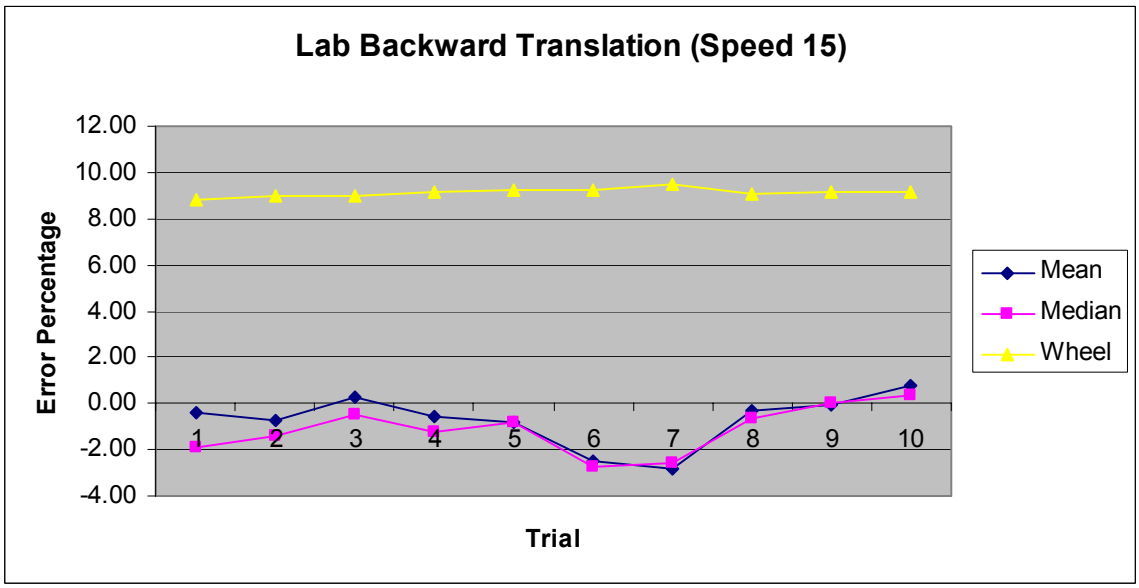


Figure 7.3: Error percentage of backward translation on lab surface.

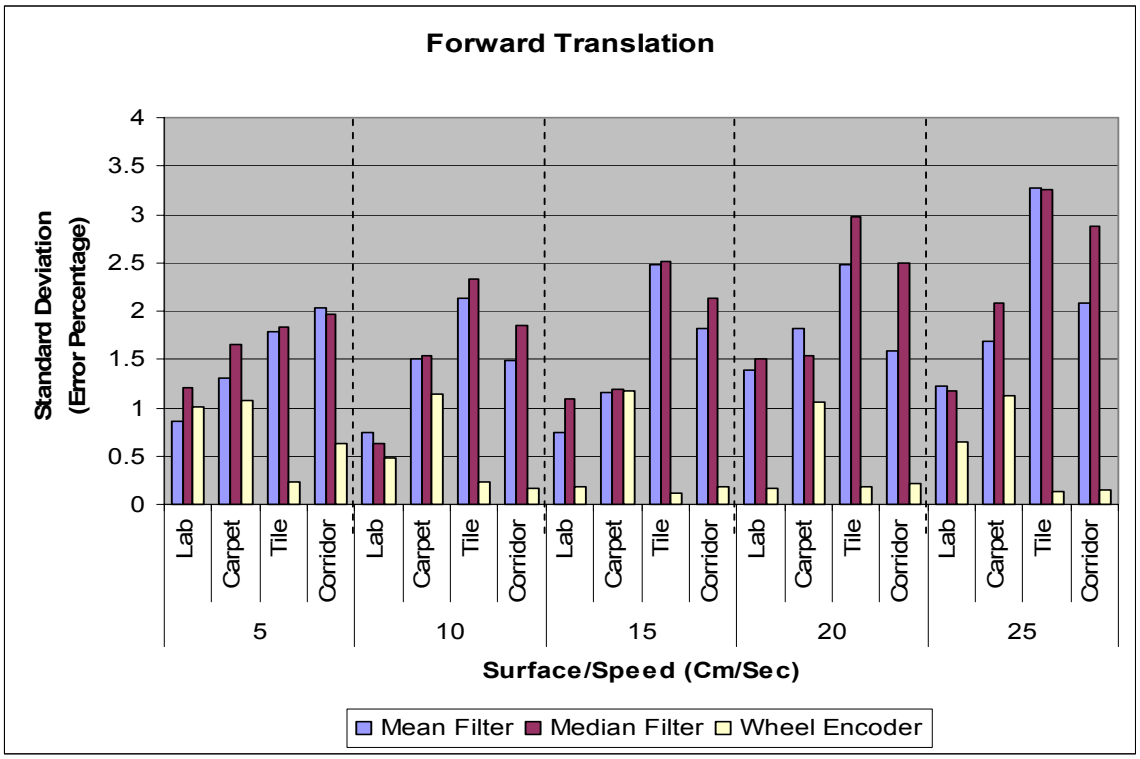


Figure 7.4: Standard deviation of error percentage from forward translation.

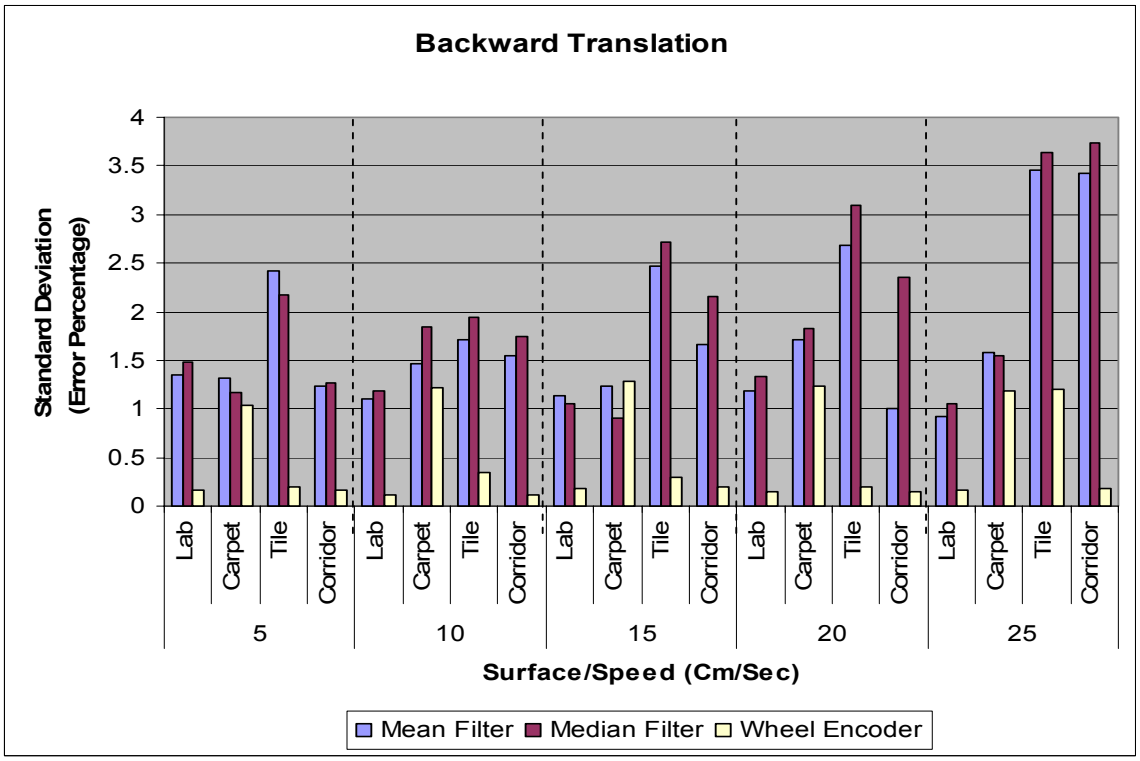


Figure 7.5: Standard deviation of error percentage from backward translation.

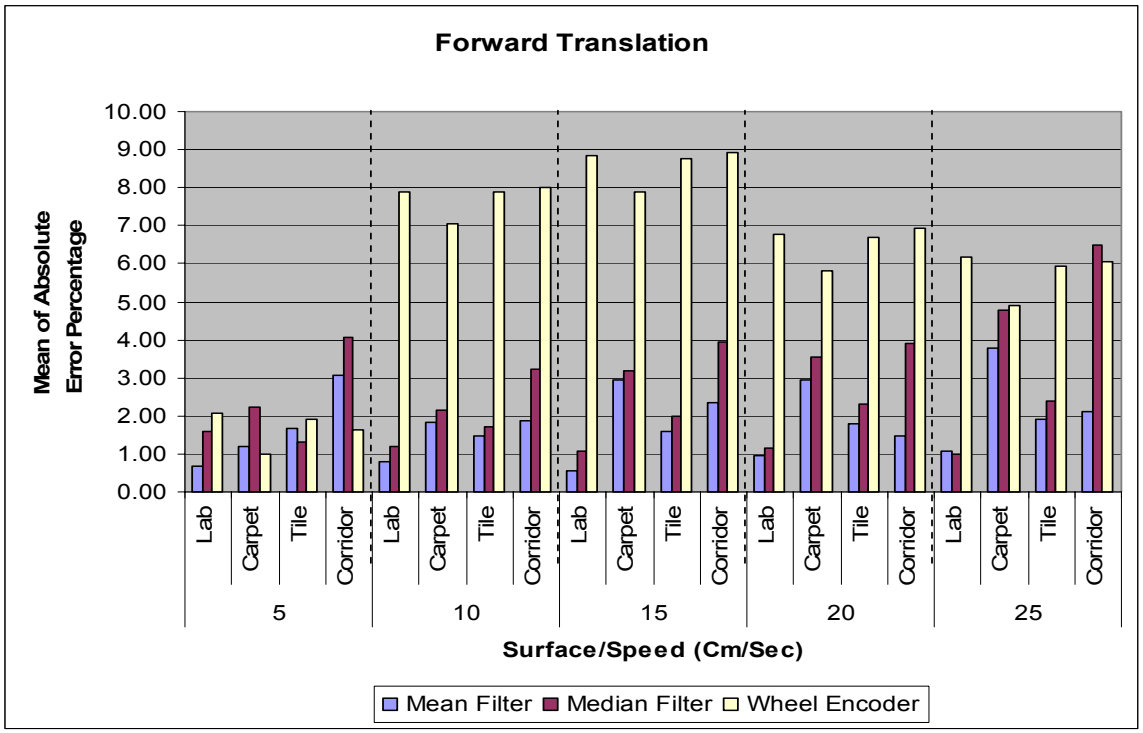


Figure 7.6: Mean of absolute error percentage from forward translation.

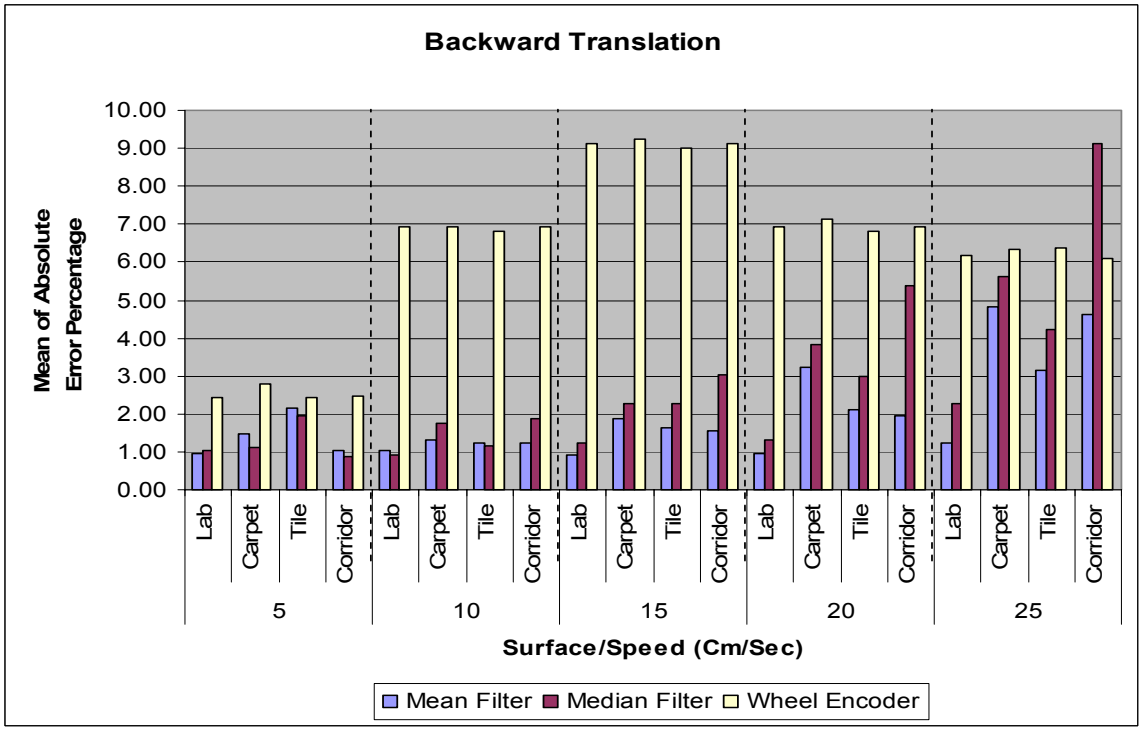


Figure 7.7: Mean of absolute error percentage from backward translation.

Table7.1: **Standard deviation** (mean absolute) of error percentage (forward).

Speed (FW)	Surface	Mean Filter	Median Filter	Wheel Encoder
5	**Lab**	**0.86** (0.66)	**1.21** (1.61)	**1.01** (2.07)
	Carpet	**1.30** (1.19)	**1.66** (2.24)	**1.07** (1.00)
	Tile	**1.78** (1.69)	**1.84** (1.33)	**0.23** (1.91)
	Corridor	**2.03** (3.08)	**1.96** (4.06)	**0.62** (1.65)
10	**Lab**	**0.75** (0.80)	**0.62** (1.20)	**0.48** (7.89)
	Carpet	**1.50** (1.84)	**1.53** (2.16)	**1.14** (7.04)
	Tile	**2.14** (1.47)	**2.33** (1.71)	**0.23** (7.89)
	Corridor	**1.49** (1.87)	**1.85** (3.22)	**0.17** (8.00)
15	**Lab**	**0.74** (0.54)	**1.09** (1.06)	**0.19** (8.85)
	Carpet	**1.15** (2.94)	**1.19** (3.18)	**1.18** (7.88)
	Tile	**2.48** (1.59)	**2.52** (2.01)	**0.12** (8.75)
	Corridor	**1.81** (2.35)	**2.13** (3.95)	**0.19** (8.93)
20	**Lab**	**1.39** (0.96)	**1.51** (1.16)	**0.17** (6.79)
	Carpet	**1.82** (2.94)	**1.53** (3.54)	**1.05** (5.83)
	Tile	**2.48** (1.78)	**2.98** (2.31)	**0.18** (6.70)
	Corridor	**1.59** (1.47)	**2.50** (3.90)	**0.21** (6.94)
25	**Lab**	**1.22** (1.08)	**1.17** (1.00)	**0.65** (6.16)
	Carpet	**1.68** (3.77)	**2.08** (4.78)	**1.13** (4.89)
	Tile	**3.28** (1.93)	**3.26** (2.41)	**0.14** (5.94)
	Corridor	**2.08** (2.13)	**2.88** (6.49)	**0.15** (6.07)

Table 7.2: **Standard deviation** (mean absolute) of error percentage (backward).

Speed (BW)	Surface	Mean Filter	Median Filter	Wheel Encoder
5	**Lab**	**1.35** (0.96)	**1.48** (1.04)	**0.16** (2.44)
	Carpet	**1.32** (1.47)	**1.17** (1.13)	**1.03** (2.77)
	Tile	**2.42** (2.16)	**2.17** (1.95)	**0.19** (2.43)
	Corridor	**1.24** (1.04)	**1.26** (0.88)	**0.16** (2.49)
10	**Lab**	**1.11** (1.03)	**1.19** (0.91)	**0.12** (6.95)
	Carpet	**1.47** (1.30)	**1.84** (1.76)	**1.22** (6.94)
	Tile	**1.72** (1.25)	**1.95** (1.15)	**0.34** (6.81)
	Corridor	**1.55** (1.22)	**1.74** (1.88)	**0.11** (6.93)
15	**Lab**	**1.14** (0.92)	**1.05** (1.22)	**0.18** (9.12)
	Carpet	**1.24** (1.86)	**0.91** (2.27)	**1.29** (9.26)
	Tile	**2.47** (1.62)	**2.71** (2.28)	**0.29** (9.00)
	Corridor	**1.66** (1.56)	**2.16** (3.04)	**0.20** (9.13)
20	**Lab**	**1.18** (0.94)	**1.33** (1.32)	**0.15** (6.93)
	Carpet	**1.71** (3.21)	**1.83** (3.82)	**1.24** (7.12)
	Tile	**2.69** (2.10)	**3.10** (3.00)	**0.20** (6.80)
	Corridor	**1.01** (1.96)	**2.35** (5.38)	**0.15** (6.92)
25	**Lab**	**0.93** (1.25)	**1.05** (2.28)	**0.16** (6.18)
	Carpet	**1.58** (4.81)	**1.55** (5.62)	**1.18** (6.32)
	Tile	**3.46** (3.14)	**3.64** (4.23)	**1.20** (6.38)
	Corridor	**3.42** (4.62)	**3.74** (9.12)	**0.18** (6.10)

From the experiments on robot translations, the wheel encoders have lowest standard deviation, which is around 1 or less. This means that the wheel encoders perform more consistently on all surfaces. The visual odometry seems to work very well on the lab surface as well as carpet at all speeds. It is obvious that the visual odometry does not perform well on the tile surface. The standard deviation becomes higher as the speed increases. This is probably due to a lack of texture on the surface. Unexpectedly, the visual odometry performs more consistently on the corridor surface than the tile. However, similar to the tile surface, the performance deteriorates as the speed increases.

When analyzing the results in terms of error magnitude, the visual odometry performs extremely well on the lab surface at all speeds. On other surfaces, it performs fairly well at low and medium speeds but the error magnitude becomes larger at the higher speeds. In the case of the carpet surface, it is probably because the texture also moves faster in the image plane at higher speed. Since the carpet texture is repeated at an interval distance, the motion estimation may rather select a closer match area for some macrobloks than the real one caused by the robot motion. In the cases of the tile and corridor surfaces, these surfaces are somewhat low textured. As a result, the MPEG encoder seems to produce fewer motion vectors. In addition, some macroblocks may be encoded as intra-block. This might be the reason why the error differences between the mean filter and median filter on the corridor surface are higher at the higher speeds. An intuitive assumption is that the MPEG encoder may produce a sparse motion field, which contains zero or small motion vectors in more than half of each row. However, this is not a surprising issue at all hence this is the expected nature of the MPEG encoder.

The wheel encoders, on the other hand, produce similar average absolute error percentages on all surfaces and the results are separated into two groups. The error magnitude is very small at the very low speed (5 cm/sec). At the speed of 10 cm/sec or higher, the values become higher but stable. As its standard deviation is low, the wheel encoders can perform more accurately with a calibration value for the low speed (5 cm/sec) and another value for higher speeds.

The use of the mean filter and median filter seems to yield comparable results except for the corridor surface at higher speeds. This may be a result from sparse motion vectors as described above. One thing to keep in mind is that this system is calibrated on the mean filter. That is the reason why it seems to have lower error percentage.

What we learn from this experiment is that the robot velocity is limited by the video frame rate. The optimal speed, in general, is approximately 15 cm/sec or slower. The speed of 20 cm/sec seems to be sufficient on certain types of surface in terms of the error magnitude and accuracy consistency. Generally, the error from the visual odometry is proportional to the robot velocity. If the frame rate is low and the speed is high, the MPEG encoder may produce smaller motion vectors (due to the repeated texture pattern or limited search area), or, in the worst case, it may not produce motion vectors at all. On the contrary, if the frame rate is high but the robot velocity is low, MPEG may not produce the motion vectors either. The tiny motion in a macroblock may cause the MPEG to encode only the residual data and leave the motion vector at zero. This leads to a suggestion for future work on implementing an adaptive frame rate for visual odometry. A simple idea is to have the video frame rate vary in inverse proportion to the robot velocity.

Experiment of Visual Odometry on Robot Rotation

In this experiment, the robot performs the rotations, left and right, on the same surfaces as in the translation experiments. Each rotation is performed 10 times on the turns of 90, 180, 270, and 360 degrees, respectively. The angular velocity is fixed at 7 degrees/second for all surfaces, which seems to be the maximum speed for optimal performance, except for the carpet, which is set to 8 degrees/second. This is due to the fact that the carpet is extremely frictional. The robot sometimes cannot complete its rotation. When this happens, it attempts to accomplish its task even though it is stuck in the same position. Essentially, this can severely damage the robot's gear system. However, the robot's turning speed on the carpet surface seems to be slower than 8 degrees/second possibly due to the surface friction.

Similar to the previous experiment, all necessary data are saved in text files for further analysis. The data representations and comparisons are shown in the same fashion as the above experiment. Examples of typical data from the experiments of the robot rotations are given in Figure 7.8 and Figure 7.9, respectively. The graphs of the standard deviation of error percentage are shown in Figure 7.10 and Figure 7.11, and the graphs of the mean of absolute error percentage are shown in Figure 7.12 and Figure 7.13. The numeric data of these values are listed in Table 7.3 and Table 7.4, respectively.

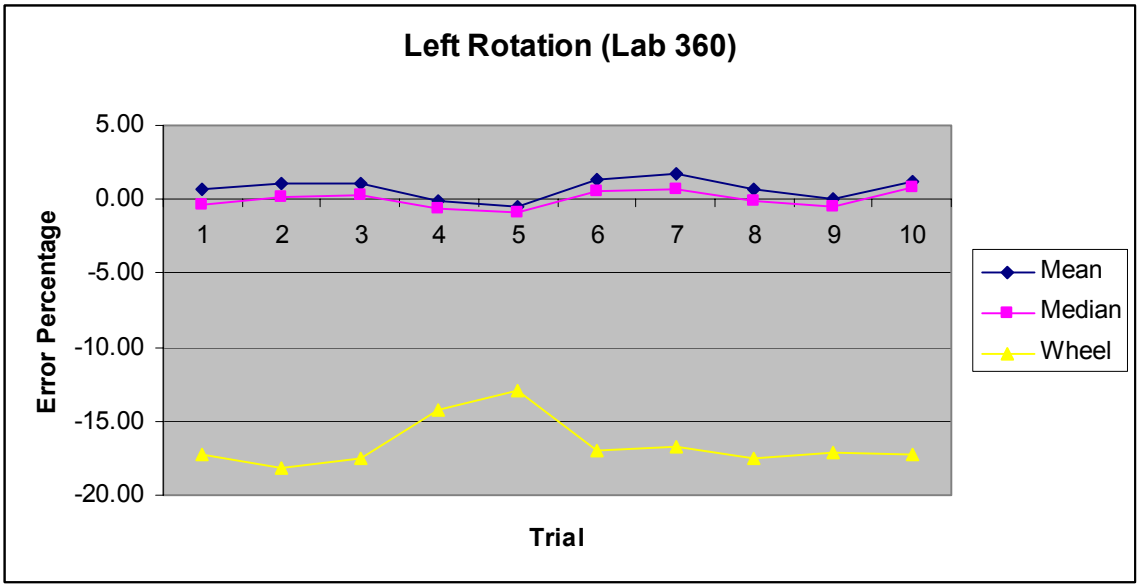


Figure 7.8: Error percentage of left rotation of 360 degrees on lab surface.

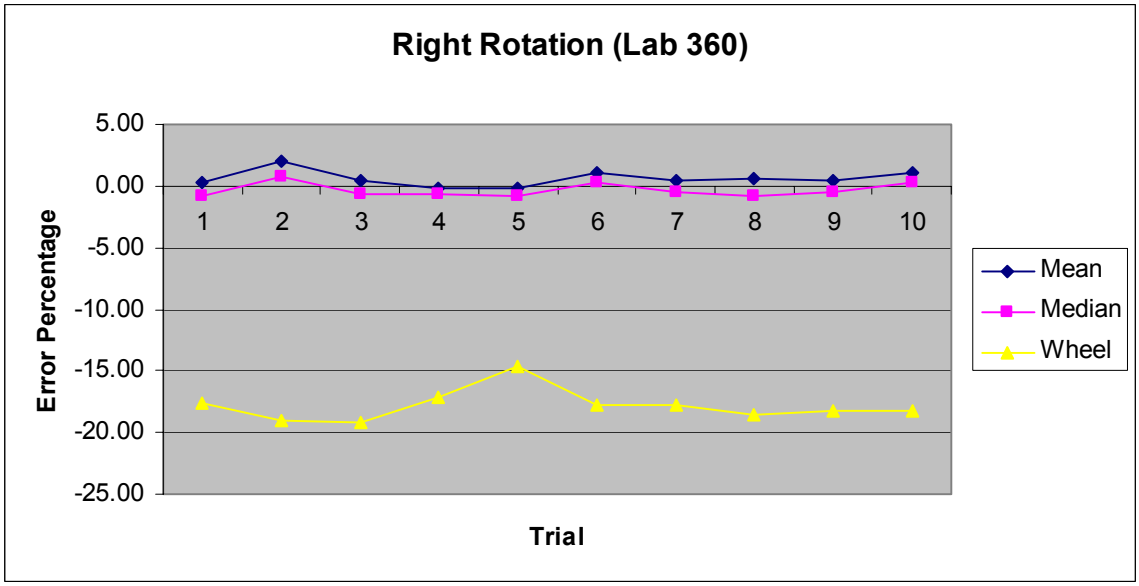


Figure 7.9: Error percentage of right rotation of 360 degrees on lab surface.

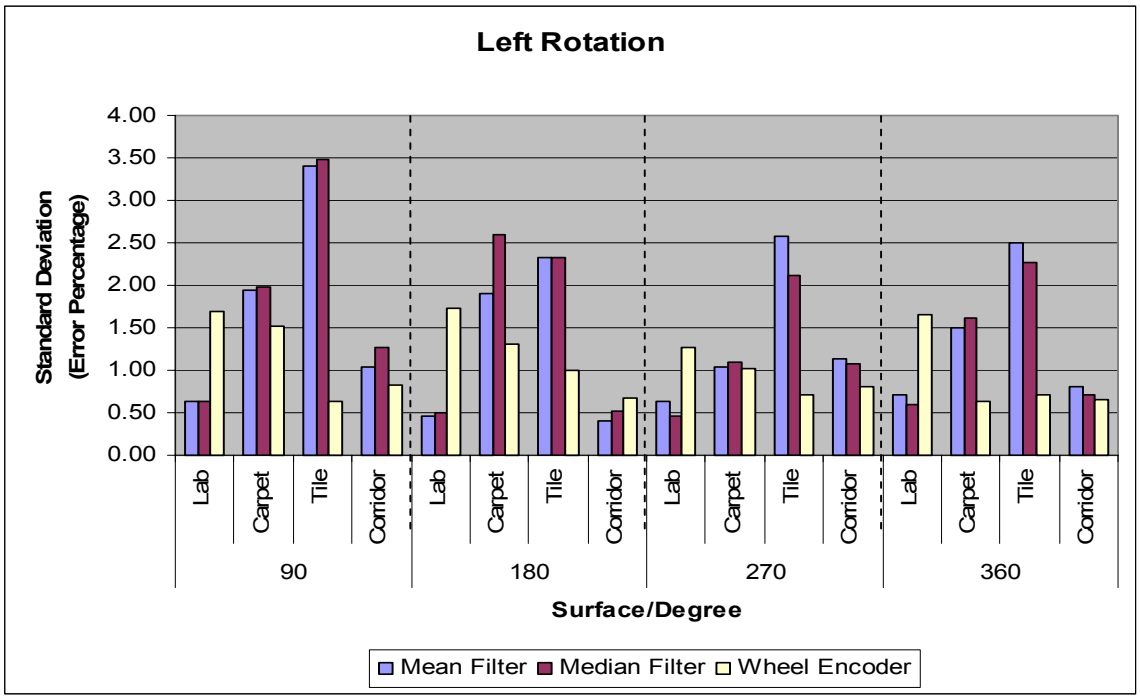


Figure 7.10: Standard deviation of error percentage from left rotation.

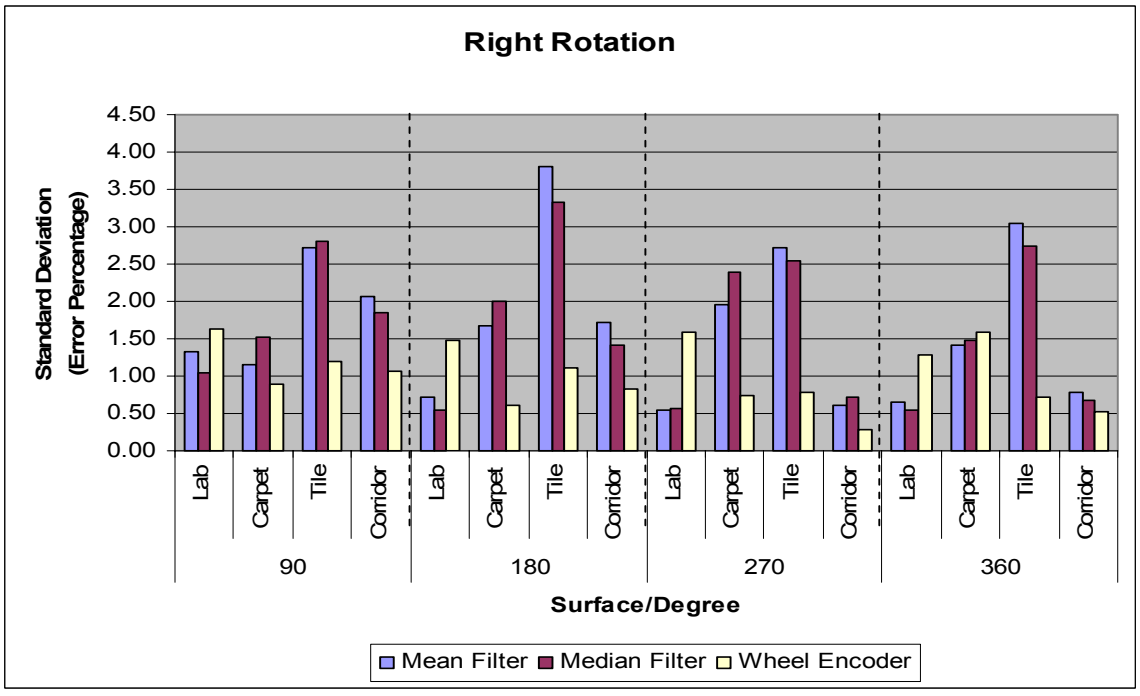


Figure 7.11: Standard deviation of error percentage from right rotation.

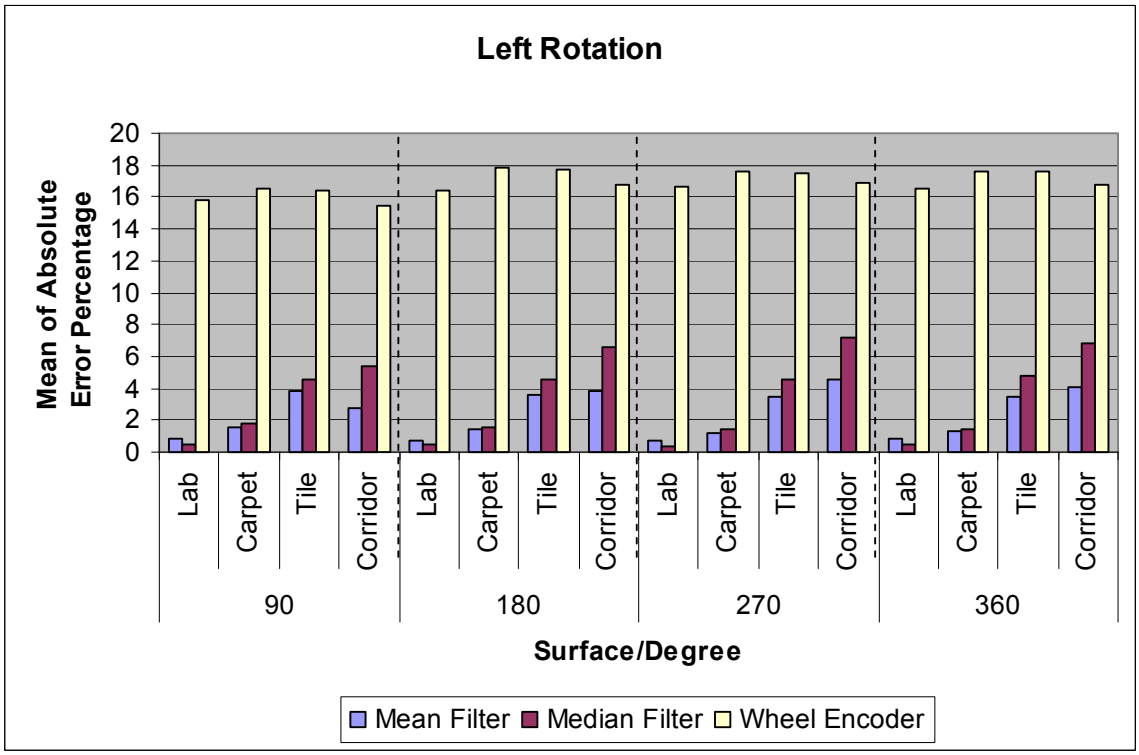


Figure 7.12: Mean of absolute error percentage from left rotation.

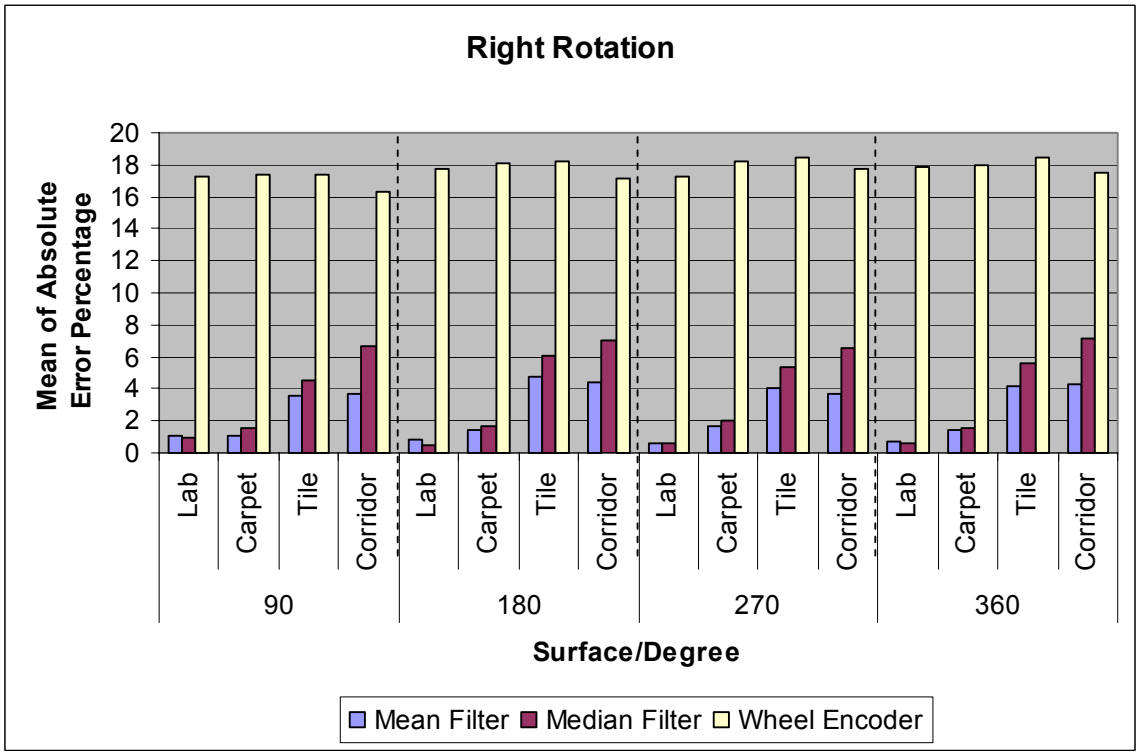


Figure 7.13: Mean of absolute error percentage from right rotation.

Table 7.3: **Standard deviation** (mean absolute) of error percentage (left rotation).

Degree L	Surface	Mean Filter	Median Filter	Wheel Encoder
90	**Lab**	**0.63** (0.79)	**0.64** (0.49)	**1.70** (15.80)
	Carpet	**1.95** (1.51)	**1.98** (1.74)	**1.52** (16.58)
	Tile	**3.40** (3.83)	**3.48** (4.53)	**0.63** (16.46)
	Corridor	**1.04** (2.71)	**1.26** (5.36)	**0.82** (15.45)
180	**Lab**	**0.46** (0.70)	**0.50** (0.42)	**1.74** (16.39)
	Carpet	**1.90** (1.38)	**2.60** (1.54)	**1.31** (17.82)
	Tile	**2.33** (3.59)	**2.32** (4.54)	**1.00** (17.67)
	Corridor	**0.41** (3.83)	**0.51** (6.60)	**0.67** (16.78)
270	**Lab**	**0.64** (0.73)	**0.46** (0.35)	**1.27** (16.65)
	Carpet	**1.04** (1.20)	**1.09** (1.44)	**1.01** (17.62)
	Tile	**2.57** (3.43)	**2.12** (4.52)	**0.72** (17.50)
	Corridor	**1.14** (4.51)	**1.08** (7.16)	**0.80** (16.86)
360	**Lab**	**0.72** (0.83)	**0.59** (0.51)	**1.66** (16.57)
	Carpet	**1.50** (1.34)	**1.61** (1.48)	**0.63** (17.64)
	Tile	**2.50** (3.49)	**2.27** (4.74)	**0.71** (17.58)
	Corridor	**0.81** (4.09)	**0.72** (6.77)	**0.65** (16.76)

Table 7.4: **Standard deviation** (mean absolute) of error percentage (right rotation).

Degree R	Surface	Mean Filter	Median Filter	Wheel Encoder
90	**Lab**	**1.32** (1.05)	**1.05** (0.93)	**1.62** (17.22)
	Carpet	**1.15** (1.10)	**1.53** (1.50)	**0.89** (17.40)
	Tile	**2.72** (3.58)	**2.80** (4.54)	**1.20** (17.35)
	Corridor	**2.06** (3.66)	**1.85** (6.68)	**1.06** (16.28)
180	**Lab**	**0.71** (0.79)	**0.54** (0.49)	**1.48** (17.73)
	Carpet	**1.67** (1.39)	**1.99** (1.69)	**0.60** (18.05)
	Tile	**3.81** (4.71)	**3.32** (6.12)	**1.10** (18.26)
	Corridor	**1.71** (4.37)	**1.41** (6.99)	**0.82** (17.13)
270	**Lab**	**0.54** (0.57)	**0.57** (0.54)	**1.59** (17.31)
	Carpet	**1.96** (1.63)	**2.39** (1.99)	**0.74** (18.25)
	Tile	**2.71** (4.02)	**2.55** (5.34)	**0.78** (18.41)
	Corridor	**0.60** (3.69)	**0.71** (6.50)	**0.29** (17.68)
360	**Lab**	**0.65** (0.68)	**0.55** (0.58)	**1.29** (17.82)
	Carpet	**1.42** (1.40)	**1.47** (1.52)	**1.58** (17.98)
	Tile	**3.04** (4.21)	**2.75** (5.60)	**0.72** (18.48)
	Corridor	**0.78** (4.27)	**0.68** (7.13)	**0.52** (17.48)

From the experiments on robot rotations, the performance of visual odometry under the controlled environment in the lab is superior to other surfaces, including the wheel encoder, in terms of error magnitude and accuracy consistency. The experiment on the carpet yields small mean error percentage but provides lower expectation on the accuracy consistency. Possibly due to the floor friction, the rotation motions sometimes become rather jerky, resulting in decreasing of its accuracy consistency. However, the tile surface gives the least accuracy consistency. This is caused by the low texture surface and the large area of glare under certain circumstances of lighting. In addition, the robot rotation may suffer from the imperfect grout joints between tiles. The robot may get stuck between uneven tiles temporarily and then swings out pretty fast after the release. This possibly leads to a higher error percentage as well. An interesting point to note is that the visual odometry on the corridor performs surprisingly well in terms of accuracy consistency, especially at larger degrees. On the corridor surface, the areas that contain bad texture or excessive reflection are stationary. For random rotations at 90 degrees, it may accidentally encounter a bad spot at some trials. Consequently, the results of short rotations may depend primarily on the random trial spot whereas the complete rotation, 360 degrees, will always encounter the good and bad spots equally for all trials.

However, the above experiment employs only the motion vectors in the lower part of the image plane in the angle calculation. An additional experiment has been conducted to investigate whether the motion vectors in the upper part of the image plane could resolve the problem regarding the consistency on the tile and corridor surfaces. First, all motion vectors are used to calculate the rotation angle. Then, only the motion vectors in the upper part are used to calculate the angle. The results from both scenarios do not give

satisfaction as expected. This is probably because of the nature of the surface properties. For these types of surface, the texture at a distance becomes less noticeable. As a result, visual odometry should rely on objects above the surface instead. Since the vertical viewing angle of the camera is small, distant objects become out of its sight.

What we learn from this experiment is that the performance of the robot rotation is dependent primarily on the surface properties. The computation of the rotation angle relies heavily on the surface texture and the surface reflection. It can be concluded from the experiment that surfaces with Lambertian reflection (tiles in this case) may have more impact on the system performance than the specular type (corridor). The video frame rate also has an impact on the computational accuracy. On surfaces that cause a jerky rotation, when the robot gets stuck and swings out after release, the speed may be too fast for the MPEG encoder to capture the real motion, resulting in decreased accuracy as well. This is the same problem that has also been discussed previously in the robot translation.

Real-Time Precipice Detection

The experiments on precipice detection are divided into two stages. Firstly, the experiments are performed on a simulated precipice. A large piece of white plain paper without texture is used in this experiment. This is to make sure that the robot can perform safely on very first attempts due to possibly unexpected failures. Figure 7.14 displays the robot detecting a fake precipice.

Figure 7.14: Experiment on simulated precipice.

Precipice Detection by Robot Velocity

In this experiment, the robot detects a simulated precipice located in front it. The detection is performed on the same setup at 10 different speeds, starting from 5 cm/sec to

50 cm/sec. The robot velocity is increased by 5 on each attempt. If the robot can detect the precipice successfully, the distance measured from the front side of the robot (the position where the robot stops) to the precipice will be made. Table 7.5 gives the results from this experiment. A graph of the stop distance by the robot speed is displayed in Figure 7.15.

Table 7.5: Precipice detection by speed.

Speed (cm/sec)	Stop at (cm)	Frame/Sec
5	56.0	6.0
10	51.5	10.8
15	48.7	10.8
20	46.0	10.8
25	43.0	10.8
30	41.0	10.8
35	32.0	10.8
40	35.0	10.8
45	28.5	10.8
50	19.3	10.8

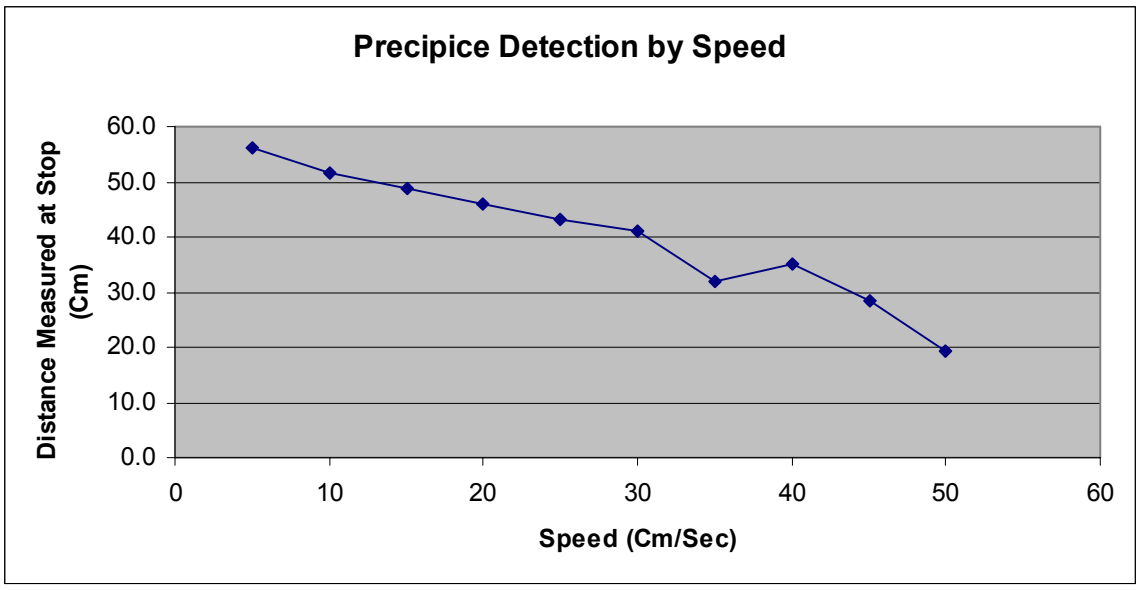


Figure 7.15: Graph displays the result from the precipice detection by speed.

From this experiment, the robot can successfully detect the simulated precipice at all speeds. The graph in Figure 7.15 indicates that the stop distance is inversely proportional to the robot velocity. It is noted that at the speed of 5 cm/sec, the frame rate is set to 6 fps. This is due to the fact that the sizes of the motion vectors are too small at lower speed. Reducing the video frame rate will increase the size of the motion vectors so that the precipice detection operates more efficiently.

Precipice Detection by Approaching Direction (Simulated Precipice)

This experiment is performed on three scenarios. First, the robot attempts to detect the precipice from the front. Then, it tries to detect the precipice when approaching from its left side and right side, respectively. Figure 7.16 shows the robot approaching the precipice from 3 different directions.

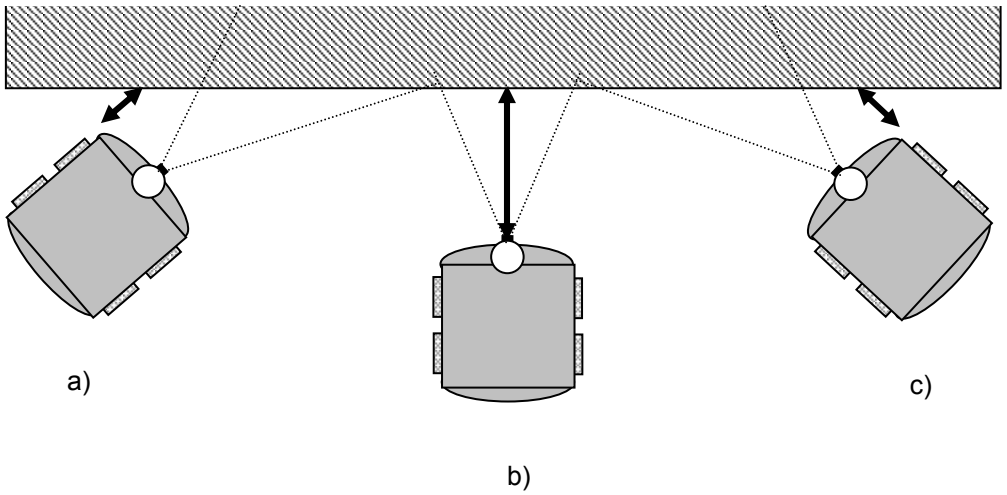


Figure 7.16: Precipice detection from different directions: a) left, b) center, and c) right.

In this experiment, the robot velocity is fixed at 15 cm/sec. Each direction is performed on 10 trials. The approximate angles are 40 degrees for the left approach and 38 degrees for the right approach. When approaching from the front side, the robot direction is perpendicular to the precipice. In this case, the measurement from the position where the robot stops to the precipice is performed in the same fashion as the measurement in the above experiment. In the case of left approach, the measurement is made from the left side of the robot's left wheel to the precipice and vice versa in the case of right approach. Each measurement is shown by the doubled arrowhead lines in Figure 7.16. Table 7.6 provides the results in numeric format while the resulting graphs are shown in Figure 7.17.

Table 7.6: Precipice detection from different directions (simulated precipice).

	Direction of Precipice Approaching		
Trial #	**Left**	**Front**	**Right**
1	23.0	51.0	24.8
2	24.0	50.7	21.8
3	24.0	50.9	27.0
4	20.0	50.5	21.8
5	20.5	51.3	22.3
6	20.5	51.2	22.5
7	23.7	49.0	24.3
8	21.5	51.0	23.8
9	22.6	49.8	21.8
10	20.5	49.5	23.8
Avg	**22.03**	**50.49**	**23.39**
StdDev	**1.61**	**0.79**	**1.69**

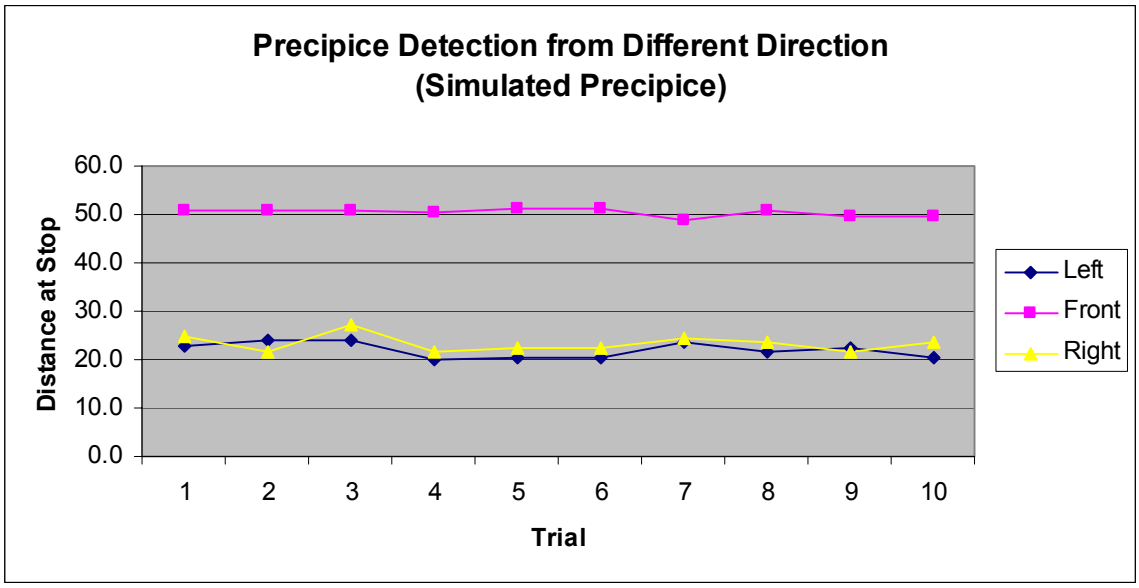


Figure 7.17: Graph of precipice detection from different directions (simulated precipice).

From the experiment, the robot can successfully detect the precipice in all approaching directions. The average measured distance in the Front approach is 50.49 cm. The left and right approaches are 22.03 cm and 23.39 cm, respectively. The standard variation is small, less than 1 for the front and less than 2 for the left and right.

The experiment on the simulated precipice indicates that the real-time precipice detection should perform consistently and be able to successfully detect the real precipice, as well. The next section will show the results from the experiment using the real precipice.

Precipice Detection by Approaching Direction (Stairwell)

The real-time precipice detection is performed to detect the stair steps located next to the mobile lab. The robot is heading for a stairwell and expected to stop before falling down. The experiment is performed in the same manner as the previous

experiment. The average left angle is 50 degrees and the average right angle is 48 degrees. The range of the angle variation for both cases is ± 5 degrees. Figure 7.18 displays the robot at its experimental location. The results are reported in Table 7.7 and Figure 7.19.

Figure 7.18: Mobile robot detecting real precipice.

Table 7.7: Precipice detection from different directions (stairwell).

	Direction of Precipice Approaching		
Trial #	**Left**	**Front**	**Right**
1	20.6	51.2	31.5
2	30.5	49	21.5
3	28.1	50	21.5
4	35.5	53.3	21.7
5	21	49.3	19.8
6	31.1	49.3	21.1
7	34.8	49.1	33.7
8	18	50.2	24.5
9	21	49.8	35.3
10	32.4	50.3	33
Avg	**27.30**	**50.15**	**26.36**
STD. Dev	**6.55**	**1.29**	**6.21**

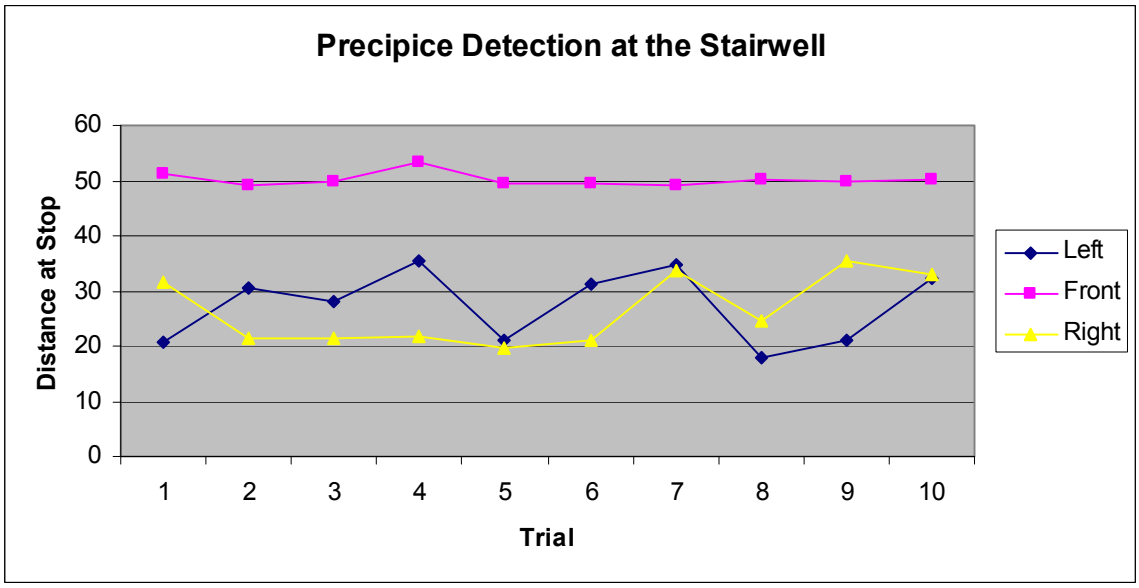


Figure 7.19: Graph of precipice detection from different directions (real precipice).

The experiment on the real precipice is also successful even though the floor surface is somewhat reflective and produces some small glare during the operation. The average distance in the case of the front detection is very close to the value from the simulation. The average values of the side detection are larger than the simulation, perhaps because of the larger angles. The higher standard deviation of the side detection may indicate the variation in the angles performed in the experiments.

From the experiments on the real-time precipice detection, the robot can detect both simulated and real precipices successfully. The camera's tilt angle is set in such a way that the width of the bottom row of the macroblocks in the image plane is, at least, equal to the width of the robot or larger. This is to make sure that the detection range is large enough to protect the robot from falling down in a precipice. As explained earlier, the selected tilt angle is made on the basis of using the same camera setup to perform

multiple sensor operations on a single camera. The results from both applications also substantiate the assumption.

Caveat

From the experiments, it is crucial to keep in mind that the camera's adaptive brightness adjustment must be turned off. A small change in lighting condition can severely impair the MPEG motion estimation. This can cause MPEG to mistakenly detect false motions even in a stationary scene. Consequently, several false motion vectors will be produced or, in the worst case, the entire scene will be encoded as an intra-frame (I Frame). Furthermore, the processing time in performing automatic adjustment is evidently slow. It may take up to 2-3 seconds to stabilize the brightness level. Another aspect that is very important in helping improve the MPEG motion estimation is the adjustment of image quality. It has also been discovered that setting the image quality to 3 or lower (on a scale of 5) can help the MPEG encoder produce a more uniform motion vector field regarding the robot motion. For instance, the visual odometry will perform poorly on the corridor surface unless the image quality is set to its lowest value, 1.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

Typical computer vision applications involve high-level mathematics and advanced programming skills. In spite of efficient coding, the algorithms are mostly time consuming. In this work, we investigate the use of an existing technology to help in implementing a visual system for mobile robot navigation. The main objective is to employ a low-cost and simple technology to reduce complexity to estimate the robot's egomotion from a cheap USB webcam along with an opensource MPEG encoder. With a little effort in modifying the MPEG encoder software, a motion field can be estimated simply from the obtained motion vectors. Such an effort is considerably less time-consuming and easier than estimating a motion field from the conventional methods. Subsequently, this motion field is analyzed in two real-time applications: visual odometry and precipice detection.

Firstly, the visual odometry system was designed to be easy to implement. A set of constraints are applied to the camera setup so that certain known values can be computed once and stored in a lookup table. The outlier rejection exploits the trigonometry method and the nature of the robot movement to eliminate erroneous motion vectors without iteration. Eventually, the calculation of the robot motions can be obtained readily from the lookup table. These basic mathematics methods prove to be very effective in implementing a real-time application.

The experiments of the real-time visual odometry are performed to evaluate the accuracy and consistency of the visual odometry on four different surfaces: lab, carpet, tile, and corridor. The visual odometry performs very well on the surface with high texture and less reflection whereas its performance suffers from the surface with a large area of glare. The results also show that the robot velocity is limited by the video frame rate. Its accuracy deteriorates as the robot runs at very high speeds. The results confirm that the visual odometry based on MPEG encoder is sufficiently accurate for use in a mobile robot without wheel encoders. On the other hand, it may be used in accordance with built-in wheel encoders to improve the performance.

Secondly, the same motion vectors obtained from the MPEG encoder are used to implement a real-time precipice detection. The proposed algorithm divides the motion vectors into three rows of patches, WATCHING, WARNING, and PANIC. The detection procedure is performed by observing the status of each row over the past 3 frames. The experiments of the precipice detection are firstly tested on a simulated precipice to protect the robot from unexpected failures. Later, it is applied to an actual precipice. The results from both experiments are successful. The experiments of detecting a precipice show that the robot can detect a precipice successfully at a high speed up to 50 cm/sec, or even higher during the test. Furthermore, the results report that the proposed algorithm is able to effectively detect a precipice approaching the robot from an angle as well. Finally, despite its simplicity, the results from the experiments validate its reliability and efficiency. It also suggests that the system can be used to detect a precipice in real situations.

Future Work

Based on what we gained from this work, a few examples of future work are given as follows:

Adaptive Frame Rate

The results from the experiments indicate that the video frame rate and the robot velocity are the major factors in creating a motion field. At a particular frame rate, the motion vector size is proportional to the robot velocity. At a high frame rate, when the robot moves very slowly, the motion vectors become very small. In this case, the MPEG encoder may produce sparse motion vectors or, in the worst case, the macroblocks may be coded as intra-block. These small and sparse motion vectors can potentially cause inaccuracy to the visual odometry calculation. In addition, the precipice detection cannot operate on a motion field of tiny motion vectors. This leads to a suggestion of estimating motion field adaptively. The idea is to dynamically adjust the video frame rate according to the robot's speed. As a result, the motion field will be comprised of motion vectors with appropriate size.

Alternative Method to Detecting a Precipice

The algorithm used in the proposed system is simple and proved to be very efficient when operating in real-time. However, from an observation during the experiments of a successful detection, when each detection status, WATCHING, WARNING, and PANIC, is activated, the slope of its fitted line over the past 10 frames will become negative (typically about -0.2). An alternative method to implement

precipice detection is to monitor the slope of the fitted line over a range of times/frames. The activation of the detection status is then dependent on this value.

Dealing with Detecting a Fake Precipice

From the successful experiment on the simulated precipice, it also implies that the robot may mistakenly detect an area of uniform surface, a surface without texture, as a precipice. This is a common problem occurring in visual applications. Inevitably, the area in doubt needs to be captured for additional processing to validate the detection.

This work shows that the use of MPEG motion vectors can tremendously reduce complexity and time in developing a visual system. The procedure to obtain the motion vectors from the encoding software is comparatively simple. The applications are able to operate efficiently under real-time constraints. Therefore, further investigations and implementations for mobile robot navigation are worthwhile and encouraging.

REFERENCES

[1] B.M. Dawant, S.L. Hartmann, J-P. Thirion, F. Maes, D. Vandermeulen, and P. Demaerel, "Automatic 3D segmentation of internal structures of the head in MR images using a combination of similarity and free form transformations: Part I, methods and validation on normal subjects", *IEEE Transactions on Medical Imaging*, Vol.18(10), pp. 909-916, October 1999.

[2] S.L. Hartmann, M.H. Parks, P.R. Martin, and B.M. Dawant, "Automatic 3D segmentation of internal structures of the head in MR images using a combination of similarity and free form transformations: Part II, Validation on Severely Atrophied Brains", *IEEE Transactions on Medical Imaging*, Vol. 18(10), pp. 917-926, October 1999.

[3] B.D. Jeffs and M. Gunsay, "Restoration of blurred star field images by maximally sparse optimization", *IEEE Transactions of Image Processing*, Vol. 2(2), pp. 202-211, April 1993.

[4] O. de Vel and S. Aeberhard, "Line-based face recognition under varying pose", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21(10), pp. 1081–1088, October 1999.

[5] Y.T. Chien, Y.S. Huang, S.W. Jeng, Y.H. Tasi, and H.X. Zhao, "A real-time security surveillance system for personal authentication", *Proceedings: IEEE 37th Annual on Security Technology*, 2003.

[6] H.X. Zhao and Y.S. Huang, "Real-time multiple-person tracking system", *16th International Conference on Pattern Recognition (ICPR'02)*, Vol. 2, pp. 20897, 2002.

[7] B.K.P. Horn and B.G. Schunk, "Determining optical flow", *Artificial Intelligence*, Vol. 17(1-3), pp. 185-203, August 1981.

[8] J. Chang, W. Hu, M. Cheng, and B. Chang, "Digital image translational and rotational motion stabilization using optical flow technique", *IEEE Transactions on Consumer Electronics*, Vol. 48(1), pp. 108-115, February 2002.

[9] S. Erturk, "Image sequence stabilization: Motion vector integration (MVI) versus frame position smoothing (FPS)", *Proceeding 2nd International Symposium Image and Signal Processing and Analysis*, pp. 266–271, 2001.

[10] G. Adiv, "Determining three dimensional motion and structure from optical flow generated by several moving objects", *IEEE Trans. on Pattern Analysis and Machine Intelligence,* Vol. 7(4), pp. 384-401, 1985.

[11] P.J. Burt, J.R. Bergen, R. Hingorani, R.J. Kolczynski, W.A. Lee, A. Hung, J. Lubin, and J. Shvaytser, "Object tracking with a moving camera; an application of dynamic motion analysis", *IEEE Workshop on Visual Motion,* Irvine, CA, 1989.

[12] S. Patwardhan, H.S. Tan, and J. Guldner, "A General Framework for Automatic Steering Control: System Analysis", *Proceedings of the American Control Conference*, Albuquerque, New Mexico, pp. 1598-1602, June 1997.

[13] O. Ramström and H. Christensen, "A method for following unmarked roads", *IEEE on Intelligent Vehicles*, Las Vegas, NV, June 2005, pp. 650-655, 2005.

[14] R. N. Bracewell, "Two-Dimensional Imaging", Perspective Model, pp 43- 48, ISBN 0-13-062621-X, Prentice-Hall, 1995.

[15] W. Burger and B. Bhanu, "Estimating 3-D Egomotion from Perspective Image Sequences", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 12(11) November 1990.

[16] A. Dev, B. Krose, and F. Groen, "Navigation of A Mobile Robot on The Temporal Development of The Optic Flow*", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 558-563, September 1997.

[17] A. Branca, E. Stella, A. Distante, "Mobile robot navigation using egomotion estimates", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 533-537, September 1997.

[18] N. O. Stoffler and Z. Schnepf, "An MPEG-Processor-Based Robot Vision System for Real-Time Detection of Moving Objects by a Moving Observer", *Proceedings of the 14th International Conference on Pattern Recognition*, Vol. 1, pp. 477, 1998.

[19] N. O. Stoffler, T. Burkert, and G. Farber, "Real-Time Obstacle Avoidance Using an MPEG-Processor-Based Optic Flow Sensor", *15th International Conference on Pattern Recognition*, Vol. 4, pp. 161-166, 2000.

[20] J. Cambell, R. Sukthankar, and I. Nourbakhsh, "Techniques for Evaluating Optical Flow for Visual Odometry in Extreme Terrain", *Proceedings of IEEE/RSJ International Conference on Intelligence Robots and Systems*, Vol. 4, pp. 3704-3711, September-October 2004.

[21] J. Cambell, R. Sukthankar, I. Nourbakhsh, and A. Pahwa, "A Robust Visual Odometry and Precipice Detection System Using Consumer-grade Monocular Vision", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 3412-3427, April 2005.

[22] A. Burns and A. Wellings, "Real-Time Systems and Programming Languages", *Chapter1: Introduction to Real-Time Systems*, pp 1-15, ISBN 0-201-40365-X, Second Edition, Addison-Wesley, 1997.

[23] S.M. Smith and J.M. Brady, "ASSET-2: Real-time motion segmentation and shape tracking", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17(8), pp. 814-820, August 1995.

[24] M. Sotelo, F. Rodriguez, L. Magdalena, L. Bergasa, and L. Boquete, "A color vision-based lane tracking system for autonomous driving on unmarked roads," *Autonomous Robots*, Vol. 16(1), pp. 95–116, January 2004.

[25] J. Campbell, A. Pahwa, R. Sukthankar, and I. Nourbakhsh. "Visual Odometry Using Commodity Optical Flow", *Intelligent Systems Demonstrations at AAAI 2004 (The 19th National Conference on Artifical Intelligence)*, San Jose, July 2004.

[26] L. Muratet, S. Doncieux, and J.A. Meyer, "A biomimetic reactive navigation system using the optical flow for a rotary-wing UAV in urban environment", *In Proceedings of the International Session on Robotics*, March 2004.

[27] I.E.G. Richardson, "H.264 and MPEG-4 video compression: Video coding for next-generation multimedia**",** London: John Wiley & Sons, Ltd., 2003**.**

[28] B. Townsend, "PAL Colour Television", *The Syndics of the Cambridge University Press*, 1970.

[29] D.G. Fink, "Color Television Standards: selected papers and records of the National Television System Committee", McGraw-Hill Television Series, 1995.

[30] "Motion estimation", *http://www.cs.cf.ac.uk/Dave/Multimedia/node259.html*.

[31] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors", *IEEE Transactions on Circuits and System for Video Technology*, Vol. 3(2), pp.148-157, April 1993.

[32] E. Linzer, P. Tiwari, M. Zubair, "High performance algorithms for MPEG motion estimation", *IEEE International Conference on Acoustics, Speech, and Signal Processing***,** *(ICASSP-96),* Vol. 4, pp.1934-1937, May 1996**.**

[33] "ISO-11712: Coding of moving pictures and associated audio for digital storage media at up to 1.5 Mbits/sec (MPEG1)", *http://www.chiariglione.org/mpeg/standards/mpeg-1/mpeg-1.htm*.

[34] "ISO-13818: Generic coding of moving pictures and associated audio (MPEG2)", *http://www.mpeg.org/MPEG/dvd.html*.

[35] “MPEG-4 overview”,
http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.

[36] “ITU-T Recommendation H.261: LINE TRANSMISSION OF NON-TELEPHONE SIGNALS”, Video codec for audiovisual services at *p* x 64 kbits,
International Telecommunication Union (ITU), March 1993.

[37] “ITU-T Recommendation H.263: SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS”, Infrastructure of audiovisual services – Coding of moving video, Video coding for low bit rate communication, *International Telecommunication Union (ITU)*, January 2005.

[38] “ITU-T Recommendation H.264: SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS”, Infrastructure of audiovisual services – Coding of moving video, Advanced video coding for generic audiovisual services,
International Telecommunication Union (ITU), March 2005.

[39] R.L. Joshi, T.R. Fischer, and R.H. Bamberger, “Lossy encoding of motion vectors using entropy-constrained vector quantization”, *IEEE Computer Society Proceedings of the 1995 International Conference on Image Processing*, Vol. 3(3), pp. 3109, ISBN:0-8186-7310-9, 1995.

[40] R. Schäfer, T. Wiegand, and H. Schwarz, “The emerging H.264/AVC standard”, *Audio/Video Coding*, Heinrich Hertz Institute, Berlin, Germany.

[41] “Huffman Coding”, National of Institute of Standards and Technology (NIST),
http://www.nist.gov/dads/HTML/huffmanCoding.html.

[42] Y. Takishima, M. Wada, and H. Murakami, “Reversible variable length codes”, *IEEE Transactions on Communications*, Vol. 43(2/3/4), pp. 158/162, February/March/April 1995.

[43] “MPEG pointers & resources”,
http://www.mpeg.org/MPEG/index.html.

[44] “Decoding MPEG video in software”,
http://www.cs.cf.ac.uk/Dave/Multimedia/node263.html.

[45] “MPEG-2 profiles and levels”,
http://viswiz.gmd.de/DVP/Public/deliv/deliv.211/mpeg/pr@lv01.htm.

[46] A.M. Tekalp, “Digital Video Processing”, Prentice Hall Signal Processing Series, ISBN: 0-13-190075-7, 1995.

[47] Y. Gong, G. Proietti, and D. LaRose, "A Robust Image Mosaicing Technique Capable of Creating Integrated Panoramas", *Proceeding: IEEE International Conference on Information Visualization*, pp. 24-29, July 1999.

[48] A. Smolic and T. Sikora, "Long-Term Global Motion Estimation and Its Application for Sprite Coding, Content Description, and Segmentation", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 9(8), pp. 1227 -1242, December 1999.

[49] M. Hebert, "Pixel-Based Range Processing for AutonomouS Driving", *Proceeding: IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994.

[50] J.I. Park, N. Yogi, K. Enami, and K. Aizawa, "Estimation of Camera Parameters from Image Sequence for Model-Based Video Coding", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 4(3), pp. 288-296, June 1994.

[51] J.L. Barron and N.A.Thracker, "Motion Constraint Equation". *Tina Memo No. 2004-012*, Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester,Stopford Building, Oxford Road, Manchester, M13 9PT.

[52] M.J. Black, "Robust Incremental Optical Flow", *Ph.D. Thesis*, Yale University, Department of Computer Science. Research Report YALEU-DCS-RR-923, 1992.

[53] J. Kim and B. Bhanu, "Motion Disparity Analysis using Adaptive Windows", *Honeywell Systems and Research Center*, Tech. Rep., 87SRC38, June 1987.

[54] S.T. Barnard and W.B. Thompson, "Disparity Analysis of Images", *IEEE Transactions on Pattern Analysis for Machine Intelligence*, Vol. PAMI-2 (4), pp. 333-340, July 1980.

[55] R. Wang and T. Huang, "Fast Camera Motion Analysis in MPEG Domain", *Proceedings of IEEE International Conference on Image Processing*, Vol. 3, pp. 691–694, Kobe, Japan, October 1999.

[56] R. Jin, Y. Qi, and A Hauptmann, "A Probabilistic Model for Camera Zoom Detection", *Proceedings of IEEE International Conference on Pattern Recognition*, Vol. 3, pp. 859-862, 2002

[57] J.I. Park, S. Inoue, and Y. Iwadate, "Estimating Camera Parameters from Motion Vectors of Digital Video", *IEEE Second Workshop on Multimedia Signal Processing*, Vol. 7(9), pp. 105-110, CA, USA, December 1998.

[58] J.G. Kim, H.S. Chang, J.W. Kim, and H.M. Kim, "Threshold-Based Camera Motion Characterization of MPEG Video", *ETRI Journal*, Vol. 26, pp. 269-272, June 2004.

[59] T.K. Chiew, P. Hill, D. R. Bull, and C. N. Canagarajah, "Robust Global Motion Estimation Using The Hough Transform for Realtime Video Coding", *Coding Symposium 2004*, University of California Davis, CA, USA, December 2004.

[60] A. Smolic, J.R. Ohm, and T. Sikora, "Object-Based Global Motion Estimation Using A Combined Feature Matching and Optical Flow Approach", *Proceedings of International Workshop on Very Low Bitrate Video Coding*, Urbana, IL, USA, October 1998.

[61] J.I. Park, N. Yogi, and C.W. Lee, "Video Composition Based on Robust Estimation of Camera parameters from Image Sequence", *Proceedings of IEEE International Conference on Image Processing*, Vol. 1, pp. 368-372, November 1994.

[62] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, "Numerical Recipes in C", *The Art of Scientific Computing*, Second Edition, Cambridge University Press, 1992.

[63] C. Morimoto and R. Chellappa, "Fast 3D Stabilization and Mosaic Construction", *Proceedings on IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 660-665, June 1997.

[64] Z. Zhang and O. Faugeras, "3D Dynamic Scene Analysis", *Springer-Verlag*, 1992.

[65] H. Broszio and O. Grau, "Robust Estimation of Camera Parameters Pan, Tilt and Zoom for Integration of Virtual Objects into Video Sequences*", International Workshop on Synthetic-Natural Hybrid Coding and 3D Imaging*, Greece, September 1999.

[66] M. Fischler and R. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", *Communications of the ACM*, Vol. 24(6), pp. 381-395, 1981.

[67] "Open Computer Vision Library", *http://sourceforge.net/projects/opencvlibrary/*.

[68] J.Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker", *OpenCV Documentation*, Intel Corporation, Microprocessor Research Labs, 1999.

[69] MobileRobots Inc., "Robots for Commercial & Research Applications", *http://www.mobilerobots.com/*.

[70] "Linux QuickCam USB Web Camera Driver Project", *http://qce-ga.sourceforge.net/*.

[71] "Fedora Project", *http://fedora.redhat.com*.

[72] "FFMPEG", *http://ffmpeg.mplayerhq.hu*.

[73] “gtkmm - the C++ interface to GTK+”,
http://www.gtkmm.org/.

[74] S. Nakamura, “APPLIED NUMERICAL METHODS WITH SOFTWARE”,
Chapter 8: Curve Fitting to Measured Data, pp. 274-288, ISBN 0-13-041047-0, Prentice Hall, Englewood Cliffs, N.J. 07632.