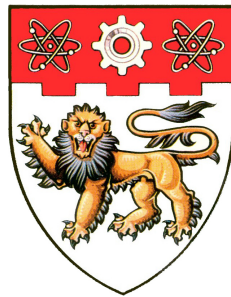


Real-time Pedestrian Detection and Tracking On Moving Vehicle



Nguyen Xuan Tuong
Department of Computer Engineering
Nanyang Technological University

This thesis is submitted for the requirement of

Industrial Attachment

May 30, 2010

Acknowledgements

I would like to thank Professor Alois Knoll, Technical University of Munich and Nanyang Technological University for giving me an excellent research opportunity in real-world research projects with several professional researchers.

I wish to express my gratitude to my supervisor, Dr. Christian Buckl, for accepting me to work at his research institutes and providing me an interesting project to work with during my Industrial Attachment.

My work would never have been possible without the regular advice and encouragement provided freely by my direct supervisor, Mr. Thomas Muller. I would also like to thank all of my colleagues for their valuable assistance and support.

Abstract

Pedestrian monitoring is becoming more and more important nowadays in a variety of areas like traffic control, security monitoring, pedestrian flow analysis. Current intelligent car needs a safety feature that is able to detect and track the pedestrian to avoid collision ahead of time. Although there is currently quite a lot of pedestrian tracking system developed by several research institutes, very few of them can cope to the challenging requirement of moving vehicle and multiple-target tracking in real time. The purpose of my project is to develop a robust real time pedestrian detection and tracking algorithm for a new generation of electronic car.

Contents

Nomenclature	iv
1 Introduction	1
2 Pedestrian Detection on A Single Image	3
2.1 Background Information	3
2.2 Haar Cascade Classifier	6
2.2.1 Algorithm Overview	6
2.2.2 Haar Cascade Training	7
2.2.3 Detection and Result	8
2.3 Histogram of Oriented Gradient Classifier	9
2.3.1 Algorithm Overview	9
2.3.2 Detection and Result	10
2.4 Detection Result and Further Discussion	11
3 Detection In Two Images And Depth Estimation	13
3.1 Single View Geometry	13
3.1.1 Camera Model	13
3.1.2 Image Distortion	16
3.1.3 Single Camera Calibration	18
3.2 Two View Geometry	18
3.2.1 Two Camera Calibration	18
3.2.2 Epipolar Geometry	19
3.2.3 Image Rectification	20
3.2.4 Depth Estimation	22
3.2.4.1 DLT Algorithm	22
3.2.4.2 Correspondence Problem	24
3.2.4.3 Optimal 3D Pose Estimation	28
3.3 Result and Discussion	29

4	Real-time Multi-Target Tracking	30
4.1	Background Information	30
4.2	Recursive Bayesian Filtering	31
4.3	2D Tracking on a Single Image	32
4.3.1	Bayesian Filtering Distribution	32
4.3.2	Algorithm Implementation	33
4.3.3	Result and Further Discussion	34
4.4	3D Tracking	37
4.4.1	Algorithm Overview	37
4.4.2	Further Discussion	37
5	Conclusion	38
	References	40

Chapter 1

Introduction

People detection and tracking perhaps is not a new problem in computer vision. In fact, it attracts so much attentions from several people and there are a lot of researches available in this area. This classic problem may simply be people detection on static image or people detection in film or image sequence. The purpose of people detection is finding the presence of people in an image and if possible detecting the exact location of people in the image. In addition to people detection problem, people tracking problem intends to track the same people from frame in frame. The most common people tracking problem, perhaps, is camera surveillance problem.

Although there are a lot of researches in this area, almost of them focus on the case of static camera and static background. For the case of static camera, a simple background subtraction algorithm might be enough to detect moving target. However, in the case of moving camera, simple background subtraction technique is inapplicable because of the movement of the whole scene.

Moving-camera makes tracking problem become much harder than the case of static camera. Movement of cameras makes prediction of the location of target in the next frame become much harder, especially in the case of non-constant motion-velocity of camera.

For the multi-camera tracking, we even need more effort to track the same target on not only one image but several images at the same time. In this problem, we need to find an algorithm to map target from one image to the same target on the other images. Tracking in several images might be an advantage compared to tracking on a single image because in this case we can extract the depth information of an object in 3D space and use this information for calculation. However, tracking in two or more image needs more effort and careful calculation.

Multi-target tracking, in addition, is another challenge. We need to track not only one but several target at the same time. Occlusion might happen. For example, target might disappear on one frame but reappear on the next ones.

For this problem, multi-thread can be used. Each target can be tracked by one thread. However, the most challenging requirement perhaps is real-time tracking requirement. To achieve this requirement we need to consider the computation time on several cases that might happen. Therefore, choosing the best algorithm for each task is essential. In order to do so, a lot of algorithms must be tested on different data sets to compare the result.

In this report, we will investigate the possibility of implementing a Real-time Multi-Pedestrian Detection and Tracking on Moving Vehicle. The purpose is building a safety-system for a new generation of electronic car that is able to detect pedestrians intelligently and track them from frame to frame. The requirement of this project can be summarized as follow:

- Detect location of people on two images taken from two cameras mounted in front of an electronic car.
- Determine the distance from people to the cameras.
- Continuously monitor the motion of pedestrian by reliable real-time tracking on both cameras.

This is really challenging problem and it requires a lot of time and effort to complete. In the context of this report, we will introduce about the algorithm approaches that might be possible to get the job done. Detailed explanation for every parts of this project is impossible in the context of this internship report. Instead of going into detail in every parts of the algorithm, some tasks will only be introduced briefly, some explanation will be skipped. However, the idea and direction on how to implement the whole system should be as obvious and clear as possible.

In this thesis, I divide the application implementation into 3 chapters. The first chapter introduce about how to implement a detector to detect people on static image. Because background subtraction doesn't work in the case of moving camera, object detection on static image is the most promising choice to detect interested target. The second chapter will introduce about two-view geometry. The purpose is to detect the location of people in 3D space and derive the distance from people to front of the car. In the last chapter we will discuss about how to implement a real-time tracker to track multiple targets in real-time. In each chapter, we will discuss about background, available approaches, motivation idea, implementation and result will also be presented briefly at the end of each chapter. The testing procedure is based on the ETHZ data set [Ess et al. \(2008\)](#). It is great that they provided data set for result comparison.

Chapter 2

Pedestrian Detection on A Single Image

Detection is always the first step for any kind of tracking problem. We cannot track any target without knowing about where the interested target first located. In other words, detection step simply detect the interested target to sample the model of target for tracking process. The quality of tracking, as are result, depends strictly on the quality in detection step. If detection step detects a false target, the tracking will track the unnecessary target in the next frame and wasting resources at the same time. To avoid this, the detector must be as robust as possible. Real-time object detector is not compulsory in this situation because real-time tracking can carry all of hard-work. So in this chapter, we will simply discuss about how to detect the location of a person on a single image, and then in the next chapter, we will try to map the target to another image. The last chapter will described about tracking step to track targets simultaneously in both images. Lets start by looking at some background information about object detection first before investigating two promising method for pedestrian detection problem which are *Haar Cascade Classifier* and *Histogram of Oriented Gradient Detector*.

2.1 Background Information

In this section, we will briefly introduce about the object detection field and some available approaches.

Perhaps, *Single feature classifier* is the simplest classifier for object detection. The classifier just uses a single feature such as color, shadow, corner, texture, symmetry for object detection in a particular context. For example, if you want to detect a car on the road context, you just need to look for an object with

2.1 Background Information

symmetry shape. Similarly, if you want to detect a light of the car, you just need to look for the area with more brightness than its neighbor. A little bit more complex classifiers are classifiers that use multiple features for object classification. In this case, a particular context can be used to restrict the search area. For example, if you can detect a computer, most probably you can find a keyboard somewhere there. Although this approach is very easy to implement, it cannot deal with complex situation. Therefore, the usage of those approaches is very limited.

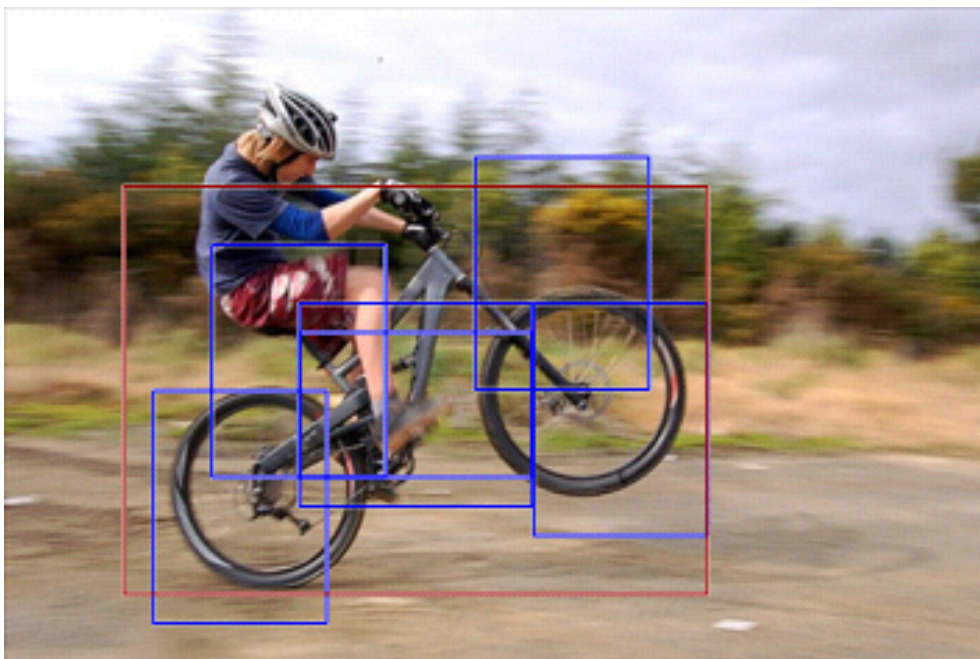


Figure 2.1: Part-Based Object Detection

A bit similar method to single feature object detection is *template-based* object detection and a bit more advanced method is *part-based* object detection or sparse-classifier Felzenszwalb *et al.* (2009) as the one shown in figure 2.1. Template based approach use a single template to classify object. If you want to detect a car, then looking for the most distinguished feature such as car light bulbs, or number plate. Because those features are pretty the same for every object in interested class (in this case interested-class is car), template approach can be used to detect the location this feature in the image. However, this approach can not be used if the interested class contain object with various possible shape appearance like in pedestrian detection case. In the case of car detection, multi-template detection can be used and the relationship between those parts can be estimated to yield

2.1 Background Information

a better result. They are all in the same category of “*Part-Based*” classifier.

Color histogram can also be used to detect object. For example in the case of skin detection, we just need to know the color distribution of sample skin to build RGB or HSV sample color histogram. Then we slide a patch cross image. Each time we calculate the color histogram model of this patch and compare to the sample skin color histogram to get the distance. This distance will be skin probability of the pixel that is at center of the patch. Using this way we can construct back-projection probability image and then locate peak location of skin in the image. Again, the usage of this approach is limited to some small areas.

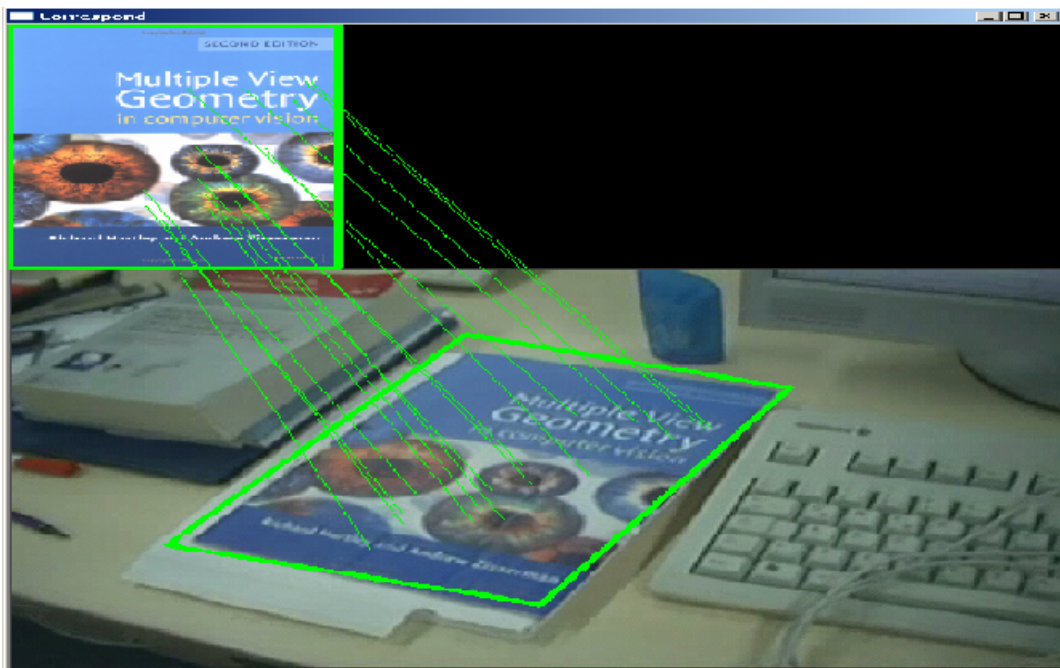


Figure 2.2: Book detection using SIFT algorithm

Another approach is using key-point detection method such as *SIFT* (*Scale-invariant feature transform*) detector Lowe (1999). This method performs well if the interested object class contains objects with similar shape, texture and appearance. SIFT algorithm basically try to find the point that is invariant to scale, rotation or brightness. We call those points as interested points. An example which this approach might be useful is to detect a particular book. This book is served as a model and the SIFT algorithm run on this sample image to detect interested points. SIFT then runs on test image and find interested feature on that image. After that, it tries to map each point in the book model to the interested point in test image to find the location of the target. However,

it cannot be applied in the pedestrian detection case.

The most promising way to build a general-purpose detector is using a large number of simple features instead of using some complex templates. Those features are extracted from positive image in training data set to build a classifier upon. Because the number of feature is very large, advanced machine learning method must be used here to fasten training process. For our problem of pedestrian detection, *Haar Cascade* detector and *Histogram of Oriented Gradient*, perhaps, are two most promising approaches. Let investigate two of them in the next section.

2.2 Haar Cascade Classifier

The usage of Haar-like feature in object detection is proposed first time by Paul Viola and Michael Jones in [Viola & Jones \(2001\)](#). This method then attracted so much attention from the others and a lot of people extended this method for a various field of object detection. In the context of this report, we only introduce briefly about this method. For more information, please take a look at a plenty of publications available. Some of them will be referred in this section.

2.2.1 Algorithm Overview

The feature used by this algorithm is simply composed by two or three black and white rectangle. At the beginning, Viola-Jones used only 4 basic Haar-like features to train the classifier for face detection. After that, the others have extended the Haar-like feature set to make the classifier better. For example, Rainer Lienhart and Jochen Maydt in [Lienhart & Maydt \(2002\)](#) rotated basic Haar-like features to enrich the feature set.

The value of Haar-like feature is the difference between sums of pixel gray level within black and white area. Because the number of feature is very large, the algorithm used the concept of “Integral Image” to make calculation much faster on a large set of Haar-like feature. “Integral Image” is also used in “SURF” key-point detection algorithm that will be described in Chapter 2.

The number of feature in Haar-like feature set is very large, for example, in 24x24 sub-window, the number of feature is more than 117000 [Lienhart & Maydt \(2002\)](#), more than the number of pixel. Therefore, machine learning algorithm such as Ada Boost is used to choose the best small set of Haar-like features for classifier. In the paper of Viola-Jones, they also introduce the concept about cascade classifier to reduce the computation times. Cascade classifier is something like decision tree. At each state, it is trained to detect interested object and eliminate un-interested background area.

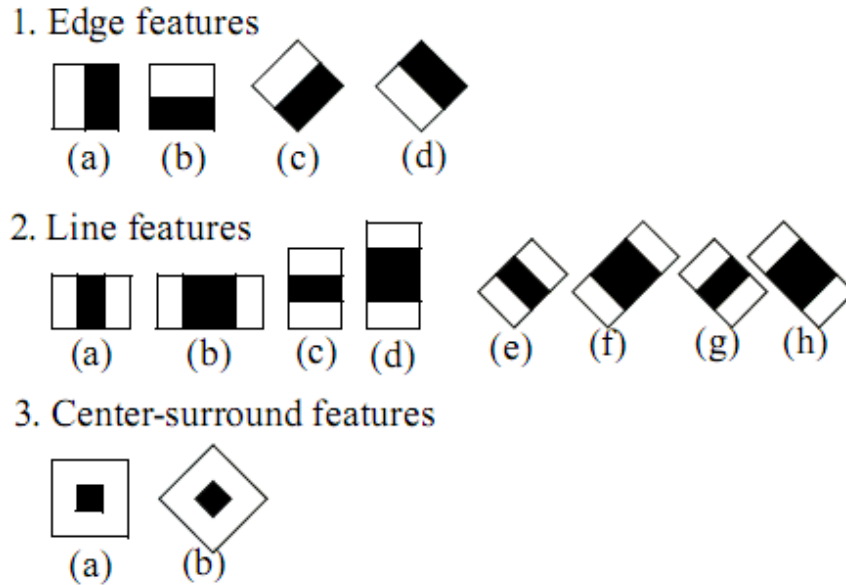


Figure 2.3: Extended Set of Haar-like Feature

2.2.2 Haar Cascade Training

Training classifier is state of art. The great number of feature gives good detection-rate but also increases the computation time. Although training classifier is hard and some kind of “state of art”, OpenCV does almost all of the hard-work for us. It has the tool to help us label the positive and negative image. It also provides the tool for training classifier. For example, haar-training utility can help us train the classifier with minimum effort. The only problem is long-time training. It might take a week for some complicated object detection and large feature set. In the following, we will talk about the command line of haar-training utility that make haar-training become much easier. Please bear in mind that at this point, we have already labeled positive image and negative image using trainingsamples utility of OpenCV or manual operation. Those command lines are already in the OpenCV documentation manual so in the following we only introduced briefly some of them.

-data <dir > data directory which trained classifier is store.

-vec<positive image > file name of positive image

-bg <bg file > background description file

-npos <nping > number of positive image

-nneg <ngimg > number of negative image

-w width of sample image

-h height of sample image

There is already some of GUI tool that make training haar-cascade file even much easier with some few clicks. However, using those tools in research is not recommended and therefore it is not in the context of this report.

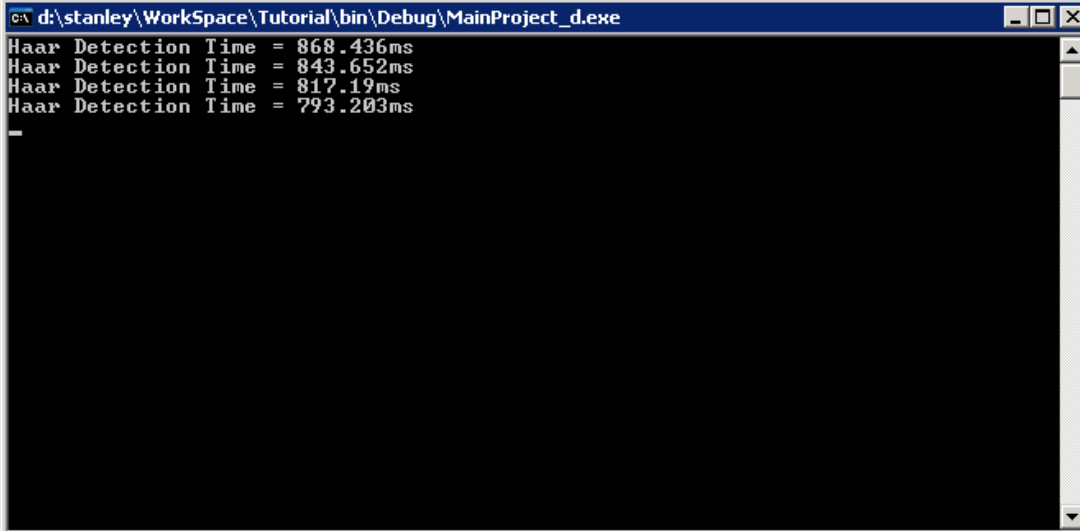
2.2.3 Detection and Result

The result of Haar-cascade detection algorithm running on AMD Athlon 64 2.41GHz processor with 1.00 GB of RAM is shown in the following figure.



Figure 2.4: Haar-Cascade algorithm running on random images of ETHZ data set

2.3 Histogram of Oriented Gradient Classifier



```
cmd d:\stanley\WorkSpace\Tutorial\bin\Debug\MainProject_d.exe
Haar Detection Time = 868.436ms
Haar Detection Time = 843.652ms
Haar Detection Time = 817.19ms
Haar Detection Time = 793.203ms
-
```

Figure 2.5: Haar-Cascade computation time

2.3 Histogram of Oriented Gradient Classifier

Histogram of Oriented Gradient descriptor is introduced first time by Navneet Dalal and Bill Triggs in [Dalal \(2006\)](#). This method aims to detect people in static image. After that, it is extended to detect a wide class of object such as car, motorbike, bicycle . In the problem of people detection in static image, HOG outperforms a lot of existing feature-based classifier and it currently is one of the best methods to solve the people detection problem. This method then is modified to make it much faster to compute and more accurate in detection.

2.3.1 Algorithm Overview

HOG uses the distribution of intensity gradients or edge direction to describe the shape of interested object for classification. In other words, shape of object can be characterized well by the edge direction. In the original HOG algorithm, Dalah divides image into cell and computes histogram of gradient directions or edge orientations over the pixels of the cell. The descriptor then is the combination of these histograms. These histograms can be contrast-normalized first to give a better result, especially in the case of changing illumination. The contrast-normalized step may be done by calculating the intensity across a bigger area and use this value to normalize cells inside. The last step is using HOG descriptor and feed it to machine learning algorithm such as Support Vector Machine or Ada Boost algorithm.

2.3 Histogram of Oriented Gradient Classifier

Although the original algorithm perform very well on people training data set, there is one big improvement of HOG classifier after that when Qiang Zhu and his collaborators introduce a new way to reduce the computation time at IEEE Conference on Computer Vision and Pattern Recognition 2006. In their paper, they introduce the concept of cascade of rejecters that is very similar to haar-cascade in face detection algorithm in the previous section. The principle is the same. Instead of using the uniform size block, they increase feature space by introducing block with various size. The first stage in detection process, they use bigger block size to eliminate the un-interested background. Then in later stage, the small block size is used. They also used the concept of “Integral Image” to fasten the process of calculating a large set of HOG features. The Adaboost algorithm is used for data training just like in Haar-Cascade Face Detection algorithm.

2.3.2 Detection and Result

The result of HOG detection algorithm running on AMD Athlon 64 2.41GHz processor with 1.00 GB of RAM is shown in the following figure.



Figure 2.6: HOG algorithm running on random images of ETHZ data set

2.4 Detection Result and Further Discussion

```
cmd: d:\stanley\WorkSpace\Tutorial\bin\Debug\MainProject_d.exe
HOG Detection Time = 4345.39ms
HOG Detection Time = 4339.99ms
HOG Detection Time = 4321.22ms
HOG Detection Time = 4311.26ms
```

Figure 2.7: HOG computation time

2.4 Detection Result and Further Discussion

First we take a look at some more screen shot of detection algorithm running on ETHZ data set.



Figure 2.8: Result from Haar-Cascade and HOG algorithm

2.4 Detection Result and Further Discussion

As we can see, the detection algorithm work very well on the data set. The pedestrian normally is detected within 3-4 frames. HOG detection algorithm seem outperform Haar Cascade algorithm about detection rate. However, it takes more time to compute. In our application, the result from HOG and Haar Cascade algorithm is filtered so that we don't have any duplicate detection result. The combination of two algorithm help us got high detection rate but also require more computation time. Therefore, the tradeoff of using two algorithms in comparison with using a single one must be considered in the real-work application. In this report, we have already investigated both methods.

Chapter 3

Detection In Two Images And Depth Estimation

In the first chapter, we have investigated two detection algorithms to detect pedestrian ahead of time. After each interested target is detected by that detector on a single image, we will need to map it on another one so that we can track the same target on two images simultaneously. Furthermore we can only extract the depth information from each detected target to the camera if we can detect the same person on two images to run the correspondence algorithm to find a good pair of correspondent point. This pair of correspondent point will be reconstructed to obtain 3D information. This is one of our main interests in this project. So we will start first by investigate the camera model in single view geometry.

3.1 Single View Geometry

In single view geometry, we only need to care about the case in which we have a single camera. We will discuss about the working principle of a camera and how to get projection of a point in 3D space to image coordinate system first. Later in this report we will investigate the system with multi-camera setting and the opposite process of reconstructing 3D information from two correspondence points in two images taken from two cameras with available calibration parameter.

3.1.1 Camera Model

The most simple camera model is pinhole camera. In the perfect pinhole camera model, the light is assumed as entering from the space with only a single ray entering from a particular point. In other words, each point in space give only

3.1 Single View Geometry

one corresponding point in the projective plane or image surface.

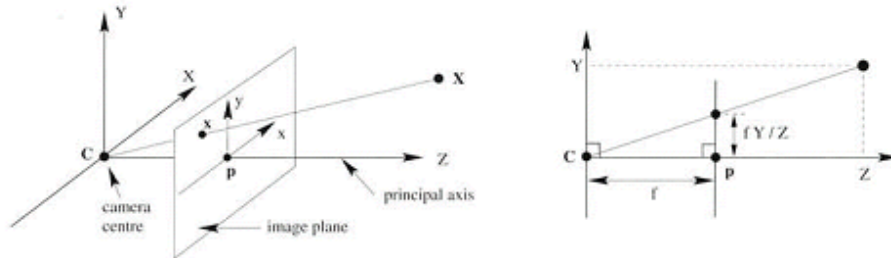


Figure 3.1: Pinhole Camera Geometry

In figure 3.1 Hartley & Zisserman (2004), the image plane is in front of camera center or center of projection. In reality, the order of image plane and camera center is opposite. Image plane or projective plane is normally behind the camera center. Because of that reason, the image of projection of an object is upside-down compared to the original one. However to make it simple, in this case, we swapped the position of image plane and pin-hole plane so that the image plane is in the front of camera center as shown in figure 3.1. The object now is right-side up and the mathematical formula therefore is much easier to construct.

By projecting 3D-point X in real-world coordinate system to image plane, we get 2D-point x in image coordinate system. If we now project both x , X to the CYZ plane in $C\vec{X}$ direction we have the right image as shown in figure 3.1. This projection doesn't alter Y , y coordinates of two corresponding points X , x compared to the original ones because the direction of this projection is $C\vec{X}$ which is perpendicular to the CYZ plane. This means if we denote x' , X' are projections of x and X to the CYZ plane, both x' and X' are 2D points in CYZ coordinate system and y -coordinates of x' , X' are the same as the original x , X correspondingly.

If we denote $X = (X, Y, Z)$ in *world coordinate system* and $x = (x, y)^T$ in *image coordinate system* then $X' = (Y, Z)^T$ and $x' = (y, p)^T$ in *CYZ coordinate systems* (Figure 2.1). p is principle point or coordinate of image center in *world coordinate system*.

Using the basic triangulation formula, we have

$$y = fY/Z$$

. Similarly, we have

$$x = fX/Z.$$

3.1 Single View Geometry

So pinhole camera model give us a mapping from 3-dimension space R^3 to 2-dimension space R^2 as the following formula.

$$(X, Y, Z)^T \rightarrow (fX/Z, fY/Z)^T.$$

Now we will put a little of linear algebra into here to combine two equations into a single equation using matrix multiplication. We have the following mapping from $R^3 \rightarrow R^2$:

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

In *homogeneous coordinate system* this equation can be re-written as follow:

$$x = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} X \tag{3.1}$$

The equation(3.1) is for the perfect pinhole camera model that we rarely see in the real-world situation. Normally we have to deal with imperfect camera model case and two kinds of following errors usually happen.

Firstly, equation(3.1) assumed implicitly that pixel is a square. This assumption is not realistic in the imperfect camera model. Pixel normally is a rectangle not square. Therefore, instead of using a single focal length parameter f for both x, y coordinates in the image coordinate system, we introduce two new parameters f_x, f_y which are focal lengths in two different coordinates of the image coordinate system. Equation(3.1) can be re-written as follow:

$$x = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} X \tag{3.2}$$

Secondly, equation(3.1) has another assumption that the principal point where principal axis meets image plane is exactly at image center. In the imperfect camera model, there usually have an offset c_x, c_y which are possible displacements of image coordinate system center or principal point away from the optical axis. To cope with this error, equation(3.2) must be re-written as follow:

$$x = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} X \tag{3.3}$$

Although we have corrected two kinds of common error that might happen in the imperfect camera model, there are still many kinds of other errors such as image distortion or skew parameter. They will be explained in the next section. At this point, we admit the following formula:

$$x = K[I|O]X \text{ where intrinsic matrix } K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.4)$$

$K[I|O]$ is camera projection matrix and denoted as P .

Equation(3.4) assumes that the object coordinate system is at the same position as camera coordinate system. If we want to consider the case of a table with its own coordinate system, we need to introduce two more parameters: R , T which are rotation and translation parts correspondingly. Those two parameters relate two coordinate systems: camera coordinate system and object coordinate system. Equation (3.4) can be re-written as follow:

$$x = K[R|T]X \text{ where } R, T \text{ is camera external parameters} \quad (3.5)$$

We now denote projection matrix P as $K[R|t]$. P is 3×4 matrix.

3.1.2 Image Distortion

Image distortion is because of imperfect lens. The main reason is because of manufacturing. There are two main kinds of lens distortions: *radial distortion* and *tangential distortion*. Radial distortion arises because of shape of the lens, whereas tangential distortion arises because of assembly process.

Both kinds of above lens distortion can be corrected by using OpenCV or Matlab Calibration Tool Box. However, in this section we will discuss how we can build an algorithm ourselves to automatically correct distorted image. This algorithm will be used together with image rectification algorithm that will be explained in the next section to create re-map matrices. We will use remap matrices to warp-perspective left and right image to get undistorted-rectified image pair.

Let look at radial distortion first. For this kind of distortion, the distortion is zero at the distortion center and increases toward the periphery. The following is picture of radial distortion [Bradski & Kaehler \(2008\)](#)

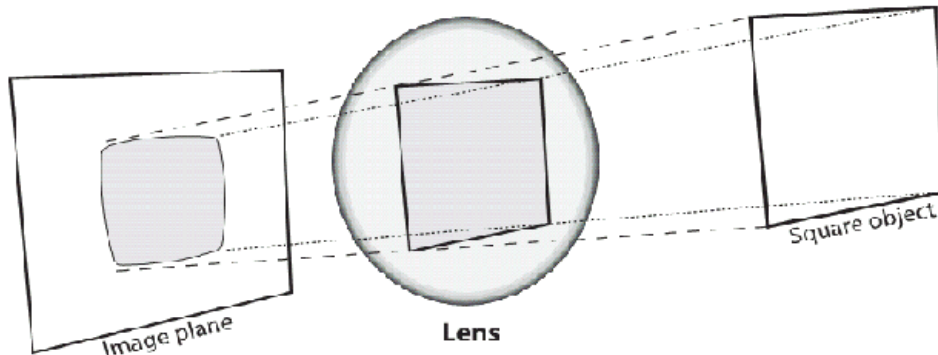


Figure 3.2: Radial Distortion

The second common distortion is tangential distortion. This distortion arise due to the lens is not exactly parallel to the imaging plane. OpenCV and Matlab Calibration ToolBox both use 5×1 matrix to represent distortion coefficients. This matrix contains k_1, k_2, p_1, p_2, k_3 in this order. For simple, we will investigate the algorithm to correct radial distortion with two distortion coefficients k_1, k_2 . It's totally similar for the case of tangential distortion.

There are many ways to correct image distortion. Image distortion is assumed as equal to zero at distortion center and increase towards periphery. Some approachs basically use image center as distortion center, whereas some of the others use camera center of projection as distortion center. In this section, we will investigate the first one. For the second one when we assume that camera center of projection is distortion center, we need to use inversion of intrinsic matrix K^{-1} to project back point in image to camera screen before making correction step. The corrected point then is multiplied to intrinsic matrix K to get the corrected location in original image.

In general, the coordinate of corrected pixels are given by the following formula:

$$x_{corrected} = x_c + L(r)(x - x_c) \quad y_{corrected} = y_c + L(r)(y - y_c) \quad (3.6)$$

where r is distance from (x, y) to distortion center (x_c, y_c)

If distortion center is basically image center and we consider equation(3.6) in image coordinate system, we can simplify the formula as follow:

$$x_{corrected} = xL(r) \quad y_{corrected} = yL(r) \quad (3.7)$$

There are no a perfect way to estimate function $L(r)$. One good approximation is using Taylor series of r with few term in the expansion: $L(r) = 1 + k_1r + k_2r^2$

Using $L(r)$ we can calculate the corrected coordinates of (x, y) and using *Pixel Interpolation* to get the suitable values from its neighbors. Image Interpolation such as linear interpolation will not be repeated again in this report.

3.1.3 Single Camera Calibration

To correct the image distortion, we need to know the distortion parameters and/or intrinsic matrix K . All of those parameters can be achieved by using single camera calibration. Although there are many way to calibrate a camera, there are two most common way to do so. Ones might want to use OpenCV or Matlab Calibration ToolBox for this purpose. Both of them use chessboard to detect the corner and to map the corresponding points of chessboard coordinate system to image coordinate system. Each pair of corresponding point will yield an equation to solve intrinsic matrix K such as equation(3.4). If we have more than enough equations to solve for the 4 or 5 intrinsic parameter (f_x, f_y, c_x, c_y , and/or *skew parameter*), we will need a good estimation that can minimize the error. One of the good estimation methods is described in the publication paper by [Zhang & Zhang](#). Similarly distortion coefficients could be found by using Brown method [Brown \(1971\)](#).

3.2 Two View Geometry

As referred in the single view geometry section, the depth information can only be extracted robustly from two or more images with overlapped sections and available calibration parameters. The difference between single-view geometry and two-view geometry is that we now need to deal with two-camera setting. Because the number of camera is greater than 1, we need to find the relationship between two cameras before making any calculation. This can be achieved by using two camera calibrations.

3.2.1 Two Camera Calibration

Two camera calibration is the same as single camera calibration in finding internal parameters (intrinsic and distortion coefficients). The only difference is that we need to find the relationship between two cameras in two camera calibration. This relationship can be described by introducing two matrices R, T where R is rotation matrix and T is translation matrix. Those matrices relate two camera coordinate systems together. Normally when people calibrate two cameras, they choose world coordinate system is at the same position as one of two camera coordinate system such as left-camera. So the location of this camera will be $(0, 0, 0)$

in world coordinate system. Again, OpenCV and Matlab Calibration Toolbox can deal with this task successfully. In the next section, we will investigate how to use all of the available calibration parameter to rectify image and estimate depth information. But we take a look at epipolar geometry first.

3.2.2 Epipolar Geometry

According to [Hartley & Zisserman \(2004\)](#), epipolar geometry is the intrinsic projective geometry between two views. It is independent of scene structure and only depends on the cameras's internal parameters and relative pose.

Before starting with 3D-Depth estimation problem, we first introduce some terminologies of *Epipolar Geometry*:

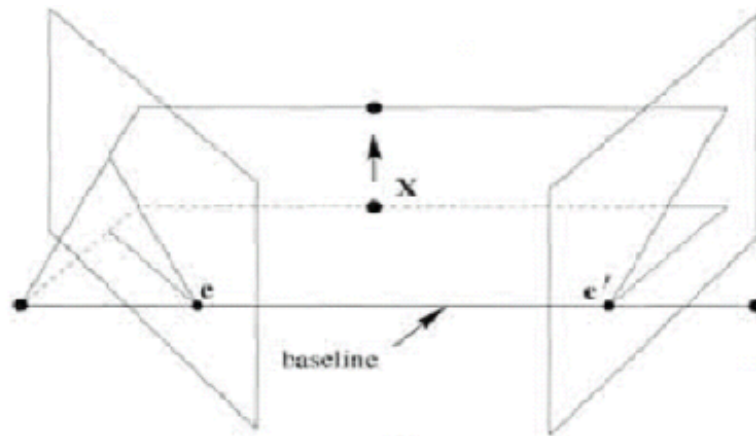


Figure 3.3: Epipolar Geometry

Epipole *Epipole* is the point of intersection of the line joining two camera centers with each image plane. We have two epipoles e and e' in the two-view geometry. Each epipole is on a single image.

Epipolar Plane There is one-parameter family of epipolar planes. *Epipolar Plane* is plane containing the line joining two epipoles or line joining two camera center of projections.

Epipolar line *Epipolar Planes* intersect with each images plane at *epipolar line*.

Fundamental matrix *Fundamental matrix* is the algebraic representation of epipolar geometry. It gives us the relationship between one point in an image

with the corresponding point in another image. For example if $x \leftrightarrow x'$ is a pair of corresponding points on two images, it must follow the epipolar geometry: $x'^T F x = 0$ where F is fundamental matrix.

By understanding the most basic terminologies of epipolar geometry, we now can proceed to the image rectification problem.

3.2.3 Image Rectification

There are two ways to rectify an image pair. The first way is in the case when we have already known every available calibration parameters. The calibration parameters is obtained by using OpenCV or Matlab Calibration Toolbox to find chessboard corners during calibration process as shown in the previous section. The second way is to deal with the case where calibration parameters is not available. In this case, we need to find the calibration ourself by using the correspondent points from two image to calculate the homography matrix. This method is already described in the book of Harley. In our report, we investigate the rectification method where calibration parameters is available from calibration process.

If you want to rectify image from scratch, Matlab calibration toolbox is great tool for you to do so. It also helps you rectify images after chessboard calibration.

In our case, we have already known calibration parameters including intrinsic matrix, extrinsic matrix of two cameras from data set of ETHZ [Ess et al. \(2008\)](#). However, we don't know anything about the rectification matrices. Therefore, we need to calculate the rectification matrices ourself. The rectification process can be summarized in the following steps.

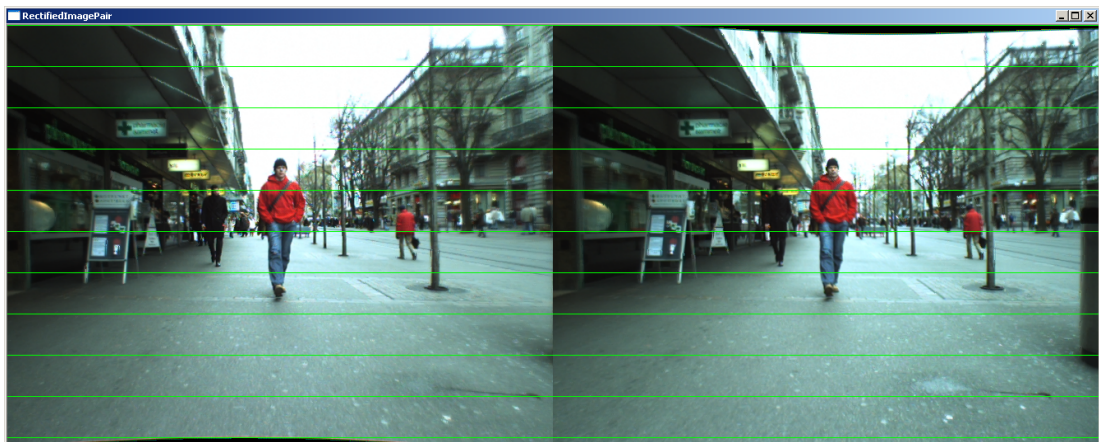
- Calculate Rl, Rr matrix to warp perspective two images using bouget methods.
- Rl, Rr ignore the case of image distortion so we need to take care about how to correct image distortion also.
- Target image is the rectified image we want to reconstruct. For each pixel in this image, we use Rl^{-1} or Rr^{-1} to wrap-back this pixel to the original image. But before immediate interpolation, we need to adjust the coordinate by using the formula in the “distortion correction” section to calculate the correct location of this pixel in original image.
- If pixel coordinate of that point in the original image is non-integer, pixel interpolation such as linear-interpolation might be used to get the RGB value for interpolating target image.

The result of rectification algorithm is shown in the following figure:

3.2 Two View Geometry



(a) Original Image Pair



(b) Image Pair After Rectification

Figure 3.4: Result of Image Rectification Algorithm

3.2.4 Depth Estimation

Depth estimation is one of the main requirements for the safety system. The purpose is monitoring the distance between each detected pedestrian to front of the car so that the braking system can be applied if necessary to stop the car completely and slow it down significantly. For multi-target tracking in real-time system, the algorithm to extract depth information should be fast to compute. We can achieve this by using the available calibration parameters from calibration process. Note that at this point, all the necessary parameters from calibration process have already been calculated. We start first by consider linear triangulation method.

3.2.4.1 DLT Algorithm

3D-reconstruction problem can be shown as in the following figure:

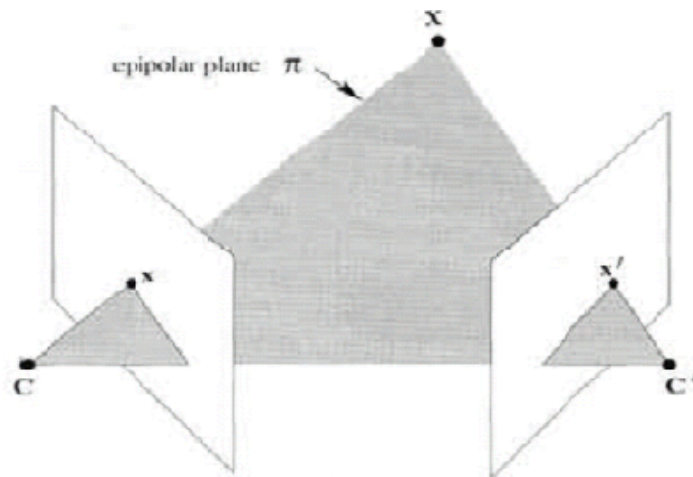


Figure 3.5: 3D-Reconstruction from correspondent points

From figure 3.5 Hartley & Zisserman (2004), we can states the 3D-Reconstruction problem as follow: x, x' are projections of the same unknown 3D-point X in 3D space to two image planes of two cameras with projection matrices P, P' correspondingly. Our task is to find the coordinate of point X in 3D space. This task can be achieved by *linear triangulation* algorithm.

Linear triangulation method or DLT method is already described in Hartley & Zisserman (2004), but like any other books, the author didn't describe in detail about the algorithm and how to implement it. In this section, we will discuss about this algorithm and add some more explanations to make it more understandable. The result will be shown at the end of this chapter.

3.2 Two View Geometry

As shown in figure 3.5, x and x' are projection of X . So we have:

$$x = PX \text{ and } x' = P'X \quad (3.8)$$

We use cross-product to eliminate scale factor in homogeneous coordinate of x, x', X , then both of equation(3.8) can be re-written as follow:

$$x \times (PX) = 0 \text{ and } x' \times (P'X) = 0. \quad (3.9)$$

Lets denote p^{iT} is the i -th row of matrix P . We have:

$$P = \begin{pmatrix} p^{1T} \\ p^{2T} \\ p^{3T} \end{pmatrix}$$

then

$$PX = \begin{pmatrix} p^{1T} X \\ p^{2T} X \\ p^{3T} X \end{pmatrix}$$

By denoting $x = (x, y, 1)^T$ in homogeneous coordinate system, the left equation in (3.9) is equivalent to:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \times \begin{pmatrix} p^{1T} X \\ p^{2T} X \\ p^{3T} X \end{pmatrix} = 0$$

After using cross-product of two 3×1 vectors, we have the following equation system:

$$\begin{aligned} x(p^{3T} X) - (p^{1T} X) &= 0 \\ y(p^{3T} X) - (p^{2T} X) &= 0 \\ x(p^{2T} X) - y(p^{1T} X) &= 0 \end{aligned} \quad (3.10)$$

Note that x, y in equation(3.10) are constants (not points) and the third equation can be obtained from two first equations($y \times$ first equation - $x \times$ second equation). So we have:

$$\begin{aligned} x(p^{3T} X) - (p^{1T} X) &= 0 \\ y(p^{3T} X) - (p^{2T} X) &= 0 \end{aligned}$$

If we do the same thing with $x' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$ then up to this point we can have

an equation system of 4 equations:

$$\begin{aligned} x(p^{3T}X) - (p^{1T}X) &= 0 \\ y(p^{3T}X) - (p^{2T}X) &= 0 \\ x'(p'^{3T}X) - (p'^{1T}X) &= 0 \\ y'(p'^{3T}X) - (p'^{2T}X) &= 0 \end{aligned}$$

or

$$\begin{aligned} (xp^{3T})X - (p^{1T})X &= 0 \\ (yp^{3T})X - (p^{2T})X &= 0 \\ (x'p'^{3T})X - (p'^{1T})X &= 0 \\ (y'p'^{3T})X - (p'^{2T})X &= 0 \end{aligned} \tag{3.11}$$

because x, y, x', y' are constants.

Let denote:

$$A = \begin{pmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ y'p'^{3T} - p'^{2T} \end{pmatrix}$$

then we have:

$$AX = 0 \tag{3.12}$$

Each row of A is 1×4 vector. X is 4×1 vector. So equation(3.12) is in fact a equation system of 4 equations with 3 unknown parameter. Although X is 4×1 vector, it is defined in homogeneous coordinate system so it has only three degree of freedom. One may want to assume the scale factor of X as 1 and choose 3 first equations in this equation system to find three unknown coordinate of X . But that approach is not recommended because it explicitly ignore the last equation in the equation system. If you straightly solve the equation system of 4 equations to find 4 parameters, the only solution you can find is zero solution. This solution is not interested with us. The best way to deal with this situation is using *Singular Value Decomposition* or *SVD* to decompose 4×4 matrix A to U, D, V as follow:

$$A = UDV^T$$

The solution will be the last collumn of V .

3.2.4.2 Correspondence Problem

The quality of triangulation depends on a pair of corresponding points found on two images. If two corresponding points are projection of the same point in space, the triangulation can yield a very accurate result with small error coming from calibration process and imperfect camera model. There are currently many algorithms that allow us to find the corresponding point of a point in an image.

3.2 Two View Geometry

For example, SURF algorithms in [Bay *et al.* \(2006\)](#) find the interested points on two images and create a descriptor for each detected interested point. The descriptor is then used for mapping process. It map each interested point found on left image to a corresponding interested point in right image to create a good pair of corresponding points. Another way to get a good pair of correspondent point without using descriptor is using fundamental matrix F as descriptor. This way is faster because F is calculated beforehand and it doesn't change once two cameras are fixed. A correspondent point in right image of a point in left image must be in the epipolar line in right image of left-image point. In contrast, calculating SURF descriptor for each point is expensive. After target is detected on left image using detector as in Chapter 1, it is then mapped to right image using the algorithm described as in the following:

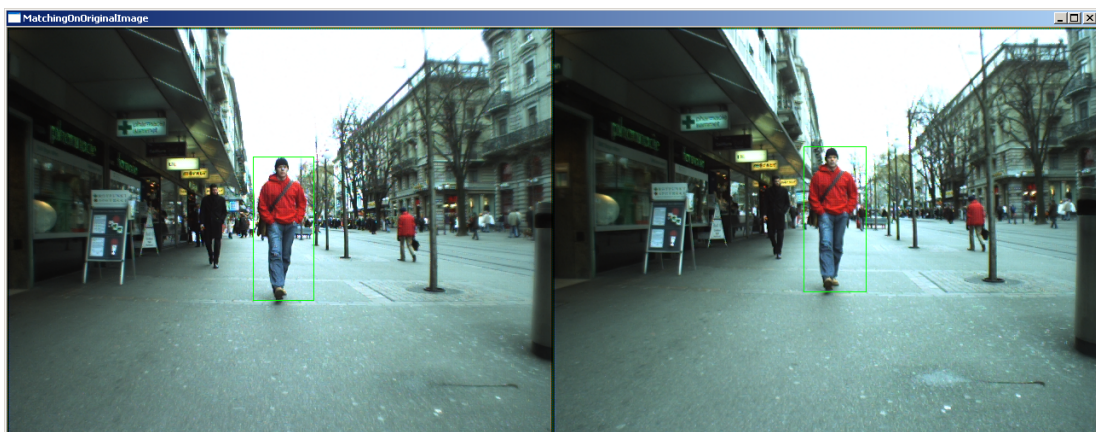
3.2 Two View Geometry



(a) Run detector on original image pair to detect target



(b) Target is mapped to between two rectified image



(c) Target is mapped back to right original image

Figure 3.6: Matching Target Algorithm between two images

3.2 Two View Geometry

After that, SURF detector can be applied to both *Region of Interest(ROI)* in both images to find a good correspondent points. After good pair of correspondent points is found in both images, DLT algorithm can be applied to reconstruct the 3D information:



(a) Point Correspondence found using SURF

```
cmd: d:\stanley\WorkSpace\Tutorial\bin\Debug\MainProject_d.exe
Number of Match Points 8
Distance of Point 0-th to the left camera is 5859.74
Distance of Point 1-th to the left camera is 5842.22
Distance of Point 2-th to the left camera is 5708.17
Distance of Point 3-th to the left camera is 5726.08
Distance of Point 4-th to the left camera is 6099.71
Distance of Point 5-th to the left camera is 5765.78
Distance of Point 6-th to the left camera is 5841.54
Distance of Point 7-th to the left camera is 5919.11
```

(b) Distance from each point found by SURF algorithm

Figure 3.7: 3D Depth Estimation using DLT algorithm on SURF matching points

There are many correspondent points found in two images. DLT algorithm calculates the distance from 3D-point of each pair of correspondent points to origin of world coordinate system. We might basically take average of all of those distances as estimated distance from detected target to the front of the car

or in more advanced way, using *RANSAC* algorithm to eliminate outliers first before take average of the sum of those distances. Note that the distance shown in figure 3.7 is in calibration object coordinate that is determined during the calibration process. If you want to have an idea about the unit of calibration object coordinate system, we can make some calculation as follow.

Suppose d is the distance between two cameras in meter-unit. From the calibration file we got during calibration process as described in the previous section, we got the distance between two camera which is 400-unit of world-coordinate system. Therefore 1 unit of world-coordinate system is equals to $d/400$ meters. The result in figure 3.7 is equal to about $6000d/400$ or $15d$.

Although *SURF* algorithm is much faster than his "old-brother" *SIFT*, the running time of SURF still quite large. Furthermore if we use this algorithm for each frame in the tracking process and for each target in each frame, the total computation time is more than real-time system can afford. There is one way to get rid of that is offload computation to GPU instead of using CPU for computation. However, it is not recommended in this case because we can find another algorithm which is much faster to compute with little less accurate compared to using SURF features.

3.2.4.3 Optimal 3D Pose Estimation

In this section, we will briefly introduce how to obtain the depth information from detected target to the camera in a fast and robust manner. Different from using SURF feature that is slow and troublesome to compute, we just use a rough estimation of two correspondent points of the same target on both images. In the previous section, we have described how to map the same target on left image to right image. After mapping process, we have two rectangles containing the location of the same people in both left and right image. One might want to chose the center of two rectangles as a pair of correspondent points for DLT triangulation. However, because when we warp back target from right-rectified image to right image or from left image to left-rectified image, we have already round up the rectangle. So choosing the center of two rectangles containing people on original image is not a reliable and accurate way to find correspondent points. The best way is choosing centers of two rectangles containing people but in left-rectified image and right-rectified image instead. Those two point then is wrapped back to the original image pair to yield a good correspondent point on original image. Those point is then used for DLT algorithms to calculate the distance. Note that, the distance can be calculated much accurate if we used the optimal triangulation method described in [Hartley & Zisserman \(2004\)](#) and therefore, it's not necessary to repeat it over here.

3.3 Result and Discussion

As we can see through out this chapter, the result from 3D estimation is reliable enough for the purpose of safety feature. Furthermore, if we use the optimal algorithm described in the previous section, we can get a very fast and quite accurate estimation. The distance estimation in 3D is not only used for the distance estimation requirement but also used for the tracker that will be described in the next chapter. At this point, we already have a reliable result from detection step and reliable result in the task of mapping target between two images and distance estimation. The only thing left is finding a real-time tracking algorithm. This task will be explained in the next chapter.

Chapter 4

Real-time Multi-Target Tracking

In the first chapter, we have investigated how to detect interested targets on a single image. Then in the second chapter, we discuss about how to map each detected target on the left image to the right image and estimate properly the distance from each of them to the origin of world coordinate system at left camera. Therefore, at this point of this report, we already have enough information to implement a tracker to track each detected target on both images from frame to frame. We need the tracker because the detector may be able to find the one target in a frame, but may not be able to find the same target on the next frame. Tracker should be less expensive in computation time than detector and it should be executed on real-time. Otherwise, the system will be too slow to apply in real-world situation. Let starts with some background information about tracking first before we proceed to our pedestrian tracking problem.

4.1 Background Information

In this section, we will talk about some basic concepts, definitions about tracking. Understanding the concepts in this part is crucial to understand the more advanced tracking method. Let starts about some assumptions we implicitly make in any general tracker. The assumptions are:

- Camera movement should be smooth. This means the camera should not have immediate changing viewpoint movement.
- Target movement should be smooth also. The tracker cannot run properly if the target changes its speed, acceleration or motion in an arbitrary way. Normally, the motion of target is assumed to be able to model mathematically.

Although some assumptions are made to make tracking problem become easier, it is still largely unsolved. There is no general purpose tracker although there did exist some limited success such as *Open Tracking Library (OpenTL)*. The challenging of the tracking problem continues to remain because of changing appearance, illumination, motion, occlusion during tracking process. Pedestrian tracking has to face with all of those challenges. It needs to deal not only with complex target shape, changing illumination but also with complex motion, scale changing and especially camera movement on real-time. All of them make pedestrian tracking become one of the hardest classic tracking problems.

In the context of this report, we will discuss about sequential parallel tracking as opposite to joint-state batch tracking. This means we use tracker to tracker each process and for multiple target tracking we just use multiple threads. The limitation of sequential tracker is that it cannot take into account the future information of target and correct past-error during tracking process. We will investigate two methods, SIR tracker and MCMC tracker. Both of them are in Recursive Bayesian Filtering category.

4.2 Recursive Bayesian Filtering

Kalman filter and *Particle filter* are two examples of *Recursive Bayesian Filter*. The “recursive model” simply means the tracking cycle we will use in our application. This cycle has the following steps:

Prediction We model the motion of target (or *State* of target in general) in order to predict the location of target.

Measurement After predicting the location of target in an image, we use observation model which may be color histogram, active contour, *SURF* feature, *HOG*... to measure the likelihood (probability) of target in the image.

Correction Correction is after measurement step to correct guessed location to give the best location for detected targets.

State of interested target can be represented by taking into account: location, velocity, acceleration, shape, scale factor... In 2D tracking on a single image, depth information is not available so we only need to care about *State* with shape model which may be rectangle or ellipse or cube... and scale factor and some noise model. In 3D tracking we need to care about location, velocity, acceleration also. Detail will be explained in the next section.

Note that state definition is flexible. If you want to use “*Active Contour*” approach instead of “*Color Based*” approach you can represent state of target

as list of contour point, spline length. . . Or if you want to use “*Articulated and Part-Based Model*” you can model state as set of vertices, location and scale.

In briefly speaking, each kind of “State Representation” yields one kind of tracking method. General tracking library usually based on this representation to make it suitable for a various kind of tracking task.

4.3 2D Tracking on a Single Image

Single Image Tracking refers to tracking process that is implemented in Single View Geometry. This means we have only one image and the depth information is not available. Because depth information is not available, the location of a target is estimated in a single image only. However, different from detecting process that uses expensive exhaustive search over the whole image, tracking process use the advanced method to model *State* of an object over time to restrict search space. In this section, we will discuss about how to implement a particle tracker based on *Sequential Importance Resampling* and *Markov Chain Monte Carlo Resampling*. Let start with Bayesian Filtering Distribution first to have the basic understand about how to implement the algorithm.

4.3.1 Bayesian Filtering Distribution

Bayesian framework is popular in almost every research areas. Many problems in signal process, image processing, or even applied statistic can be stated in the form of “state space” in which there is a transition equation to describe *prior distribution*, observation equation to describes the likelihood of the observation. In this framework, we use *posterior distribution* model $p(x_0, x_1, \dots, x_k | y_0, y_1, \dots, y_k)$ to obtain the information about x_0, x_1, \dots, x_k or more specifically for the case of *filtering distribution*, we use posterior distribution model $p(x_k | y_0, y_1, \dots, y_k)$ to obtain information about state at time k.

In the field of tracking and context of this report, we will investigate the *Bayesian Filtering Distribution* as follow:

$$p(x_t | y_{1:t}) = \alpha p(y_t | x_t) \int_{x_{t-1}} p(x_t | x_{t-1}) p(x_{t-1} | y_{1:t-1}) dx_{t-1} \quad (4.1)$$

Basically formula(4.1) describes how to obtain the target state at time t given the *observation model*, *previous state and transition model*.

Previous state $p(x_{t-1} | y_{1:t-1})$ *Previous state* is the state of object at time $t - 1$.

Transition model $p(x_t | x_{t-1})$ *Transition model* is model of Motion of Object from frame to frame.

Observation model $p(y_t|x_t)$ Suppose target is at a specific location. *Observation model* just specifies the *likelihood* of an object in this location compare to the original state.

It might be a bit tricky to understand now but in the next section, when we talk about how to implement a tracker, it will be much more understandable. At this point, we just bear in mind those concepts.

4.3.2 Algorithm Implementation

In this section, we will go into detail about how to implement particle filter. In the context of this report, we will simplify many parts of the tracker to make it more understandable. But the principle is the same for more advanced tracker. This implementation is independent with *OpenTL*.

In 2D tracking on a single image, the depth information is irrelevant. So the state of an object can be represented by 3 parameters: current x-coordinate, current y-coordinate and scale parameter s . Note that scale parameter s gives us the width and height of the “rectangle” containing interested target in current image (s is scale of new location compare to the original location). Obviously the “state” of target will give us the exact location of target in image. The question is how to estimate state of a target at time t from the previous states. What happen if the previous states are estimated but with error.

In implementation, we just represent each particle with all of three above parameters for previous state, original state, current state and one more parameter: particle weight. Totally each particle has more than 10 parameters. To simplify tracking problem, we just choose HSV color histogram as observation model. For the initial location of target, we calculate histogram at this location and use it as sample histogram model. The distance metric between two histograms in our case is *Battacharyya* distance. You might want to normalize histogram first before making any calculation. The likelihood function is assumed as the above distance metric for simplify. In real-implementation, likelihood function is chosen as a complex function of distance metric.

The number of particle we need for our estimation is next thing to consider. The more particles we use, the more expensive the computation is because at each stage we need to calculate the histogram for this particle and resample particle weight to avoid degeneracy weight after running a period of time. In our application, we just use 200 particles per target.

Prediction Each particle at time t is estimated base on the same particle at time $t - 1$ and the same original particle. More advanced estimation can be done by using the information from time 0 to time $t - 1$. The prediction based on

4.3 2D Tracking on a Single Image

transition model. Most simple transition model is $\Delta_t = \alpha\Delta_{t-1} + \beta w$ where Δ_t is the transition between time t and time 0, w is noise model.

Measurement Each particle is used to calculate the location of target in image. Each location in image then is used to calculate the HSV color histogram. Likelihood function is just Battacharyya distance metric between two color histograms in our case. For simplification, we choose this metric distance as weight of each particle.

Resample The set of particle must be resampled to avoid the degeneracy weight problem. The resample algorithm we use in this section are *Sequential Importance Resampling* (SIR) and *Markov Chain Monte Carlo*(MCMC) Resampling. Those sampling algorithms will not be explained in this report.

Correction Correction can be done by many ways. In the most simple case, just calculate the average of all particles as target location at time t .

4.3.3 Result and Further Discussion

Tracking step in 2D is real-time. Furthermore, multi-target tracking is possible; just imply giving each target one thread so that they are executed independently. Tracking is reliable if target is far from camera. However when the target comes near camera, the scale parameter changes dramatically. So that is why when target come near camera, the detection result is not robust anymore. Some modification must be made to deal with this case. Remember that scale parameter is guessed from the previous one. So when scale parameter changes too fast, the estimation must adapt in this situation also.

In the following, we will show some screen shots from tracking process. The below is just for single-target tracking but multi-target tracking is the same. The first figure is tracking result without displaying particles, whereas the second one is tracking when displaying particles.

4.3 2D Tracking on a Single Image



(a) Tracking result on 1st frame



(b) Tracking result on 7th frame



(c) Tracking result on 13th frame



(d) Tracking result on 19th frame

Figure 4.1: Tracking result using SIR tracking

4.3 2D Tracking on a Single Image



(a) Tracking result on 1st frame with particle (b) Tracking result on 7st frame with particle



(c) Tracking result on 13st frame with particle (d) Tracking result on 19st frame with particle

Figure 4.2: Tracking result using SIR tracking when displaying particles

4.4 3D Tracking

In the previous section, we have discussed about 2D tracking on a single image. 2D tracking can be used to track object on both images by using the mapping technique described in chapter 2. However, that approach is not recommended because 2D tracking is not accurate enough due to the absence of depth factor. A better way is using the depth estimation in chapter 2 as initial pose for the next estimation. Because the principle of 3D tracking is totally the same as 2D tracking, we will not go into detail about this algorithm in this section. Instead, we will briefly introduce about how to achieve this purpose.

4.4.1 Algorithm Overview

In 3D the state of object is described by the width, length, depth, scale and its location is the world coordinate system. With available intrinsic parameter from calibration process, the shape of object can be wrapped to image to give its location on image.

In 3D tracking, we also use 200 particle filters. The transition model is based on velocity and direction of pedestrian, velocity and direction of the car and the frame rate. The next particle is estimated by using this transition model. Measurement and resampling are done in totally the same way as 2D tracking. So we will not talk about this again in this section.

4.4.2 Further Discussion

3D tracking is much better than 2D tracking because the depth information is available. However the result of 3D tracking is currently unavailable at this point of time due to time limitation of this internship. But basically the idea of 3D tracking is similar to 2D tracking. It is more difficult at proposing the transition model base on velocity of both cameras and target.

Chapter 5

Conclusion

At this point of the report, we have already explained how to implement an application to detect and track multiple targets in real-time. The application is still on its way and an extensive investigation is currently in progress to improve the result in multi-directions. The project in general and the requirements in particular still seem quite challenging. Therefore the possibility that the project can be applied in real-world or just stop as an academic thesis is still an question at this point of time. However there are obviously a lot of places for improvement.

In summary, the algorithm is used in this report can be summarized as follow:

- Using Haar Cascade and Histogram of Oriented Gradient Detector to detect people in the left image of image pair.
- Rectify images using available camera calibration parameters to obtain rectified image pair.
- Map detected target from left image to left-rectified image.
- Map target from left-rectified image to right-rectified image.
- Map back target from right-rectified image to right-image to obtain location of the same target that is detected on left image.
- Run DLT algorithms to estimate the initial location of targets.
- Using SIR and MCMC tracker to track target from frame to frame.

We can summarized the result of algorithms that are used in this report as follow:

Detectors Detecting step using Haar Cascade and HOG detector is reliable. However, it is not real-time by the way. The detection can be implemented such that it is much faster in the next version of application. The key points are using cascade of HOG detector, and restricting the search area of Haar-Cascade algorithms. Cascade of HOG detector can be done in the same way like Haar-cascade. At each stage it tries to eliminate as much as non-interested background as possible.

DLT depth estimation DLT estimation is accurate enough for the case of safety-feature but not good enough if you are keen on reconstructing 3D surface of an object. However, in the case of safe-distance estimation, the algorithms is fast and accurate enough.

Tracker Current version of application used only SIR and MCMC tracker on 2D or single image. The result is not reliable enough because motion of camera is current ignored. In the next version, 3D tracking with depth and velocity estimation will be implemented.

All in all, in this report, we have investigated how to implement an “state of art” detector and tracker and explaining several ways to improve the result. This project is still very promising and the approaches described in this report should be useful for further improvement. The project will be developed further, even after the end of this internship, until performance result is available enough.

References

- BAY, H., TUYTELAARS, T. & GOOL, L.V. (2006). Surf: Speeded up robust features. In *In ECCV*, 404–417. [25](#)
- BRADSKI, G. & KAEHLER, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, New York, 2nd edn. [16](#)
- BROWN, D.C. (1971). Close-range camera calibration. *PHOTOGRAMMETRIC ENGINEERING*, **37**, 855–866. [18](#)
- DALAL, N. (2006). *Finding People in Images and Videos*. Ph.D. thesis, Institut polytechnique de Grenoble. [9](#)
- ESS, A., LEIBE, B., SCHINDLER, K., & VAN GOOL, L. (2008). A mobile vision system for robust multi-person tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, IEEE Press. [2](#), [20](#)
- FELZENSZWALB, P., GIRSHICK, R., MCALLESTER, D. & RAMANAN, D. (2009). Object detection with discriminatively trained part based models. Tech. rep. [4](#)
- HARTLEY, R. & ZISSERMAN, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, 2nd edn. [14](#), [19](#), [22](#), [28](#)
- LIENHART, R. & MAYDT, J. (2002). An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002*, 900–903. [6](#)
- LOWE, D.G. (1999). Object recognition from local scale-invariant features. [5](#)
- VIOLA, P. & JONES, M. (2001). Robust real-time object detection. In *International Journal of Computer Vision*. [6](#)
- ZHANG, Z. & ZHANG, Z. (????). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000. [18](#)