Monocular Obstacle Detection for Moving Vehicles

by

Jeffrey Ryan Lalonde

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Abstract

This thesis presents a 3D reconstruction approach to the detection of static obstacles from a single rear view parking camera. Corner features are tracked to estimate the vehicle's motion and to perform multiview triangulation in order to reconstruct the scene. We model the camera motion as planar motion and use the knowledge of the camera pose to efficiently solve motion parameters. Based on the observed motion, we selected snapshots from which the scene is reconstructed. These snapshots guarantee a sufficient baseline between the images and result in more robust scene modeling. Multiview triangulation of a feature is performed only if the feature obeys the epipolar constraint. Triangulated features are semantically labelled according to their 3D location. Obstacle features are spatially clustered to reduce false detections. Finally, the distance to the nearest obstacle cluster is reported to the driver.

Acknowledgements

I would like to thank my supervisor, Dr. Robert Laganière, for his tremendous support and guidance over the last two and a half years. Robert has given me numerous exciting research opportunities, in Ottawa and abroad, for which I am most grateful. Thanks to Luc Martel of CogniVue for believing in me and backing my approach to obstacle detection. Thanks to both Robert and Luc for giving me creative freedom in the development of this algorithm, which made the process most stimulating and rewarding. Thanks to Diego Macrini for his generous help with technical issues, and for the many useful discussions.

Special thanks to my parents and family for their continuing love and support. I dedicate this work to them.

Contents

List of Tables

List of Figures

List of Symbols

$\mathbf{X} = [X\ Y\ Z]^{\mathrm{T}}$	3D Vector	18
$\mathbf{x} = [x\ y\ 1]^{\mathrm{T}}$	Homogeneous 3D vector	20
$\mathbf{p} = [u\ v\ 1]^{\mathrm{T}}$	Homogeneous pixel coordinate	22
R	Rotation matrix	19
$\mathbf{T}$	3D translation vector	19
$\{\mathrm{R}, \mathbf{T}\}$	Rigid motion between two cameras	25
$\{\mathrm{R}_p, \mathbf{T}_p\}$	Planar motion	26
$\pi(\cdot)$	Perspective projection	20
h	Distance from camera center to the ground	27

Chapter 1

Introduction

This thesis presents a 3D reconstruction approach to the detection of static obstacles from a single rear view parking camera. The objective was to design a system that would alert the driver to the presence and location of an obstacle as part of a parking assistance application.

1.1 Motivation

Today there is a huge demand for computer vision tasks. On one hand, computer vision has seen tremendous progress in recent years. Many individual disciplines have advanced to a state where algorithms are becoming applicable for real-world tasks [6]. On the other, the cost (and size) of high quality cameras and designated image processing hardware has dropped to a point where smart camera manufacturers are able to make substantial profit margins on their products. These condition have fostered an unprecedented demand for commercial vision systems.

In particular, passenger vehicles are increasingly fitted with various sensors to improve safety and aid in navigation [21]. Most common among these sensors are rear view parking cameras. Significant value can be added to these cameras by providing vision-based parking assistance via obstacle detection.

Many obstacle detection methods have been proposed, but none offer a clear advantage over the others. Moreover, the proposed methods are designed for forward looking cameras with the detection of vehicles or pedestrians in mind. These scenarios differ significantly from parking situations in the range of vehicle speed, the types of expected obstacles and the proximity of these obstacles to the camera. As a result, there is a need for vision-based obstacle detection methods specific to parking scenarios.

Traditional parking assistance systems use radar or ultrasonic range-finders to detect obstacles. These have proven reliable for detecting large metal objects, even in poor visual conditions. However, these sensors tend to miss small non-metal objects because they lack sufficient spatial resolution [36]. Parking cameras are an attractive safety feature because, in addition to being cheaper than range-finders, they display the rear visual field to the driver. The advantage to a vision-based obstacle detector, then, is a combined camera and range sensor for a single, low-cost, piece of hardware.

Vision-based systems either use a stereo camera rig or a single camera. Well calibrated stereo systems can achieve impressive results, but at a severe cost. Namely, the baselines for these systems are usually greater than 1m to enhance depth resolution. They require high quality CCD cameras with long focal-length lenses for adequate spatial resolution. Also, the best results have been achieved with three cameras, rather than just two. These requirements restrict the adoption of stereo systems to market products [32].

Monocular systems operate at a reduced cost and require little maintenance, but lack the depth perception of stereo systems. Monocular systems can be divided into three categories; (1) appearance-based methods, (2) motion-based methods, and (3) reconstruction-based methods.

Appearance-based methods work under the assumption that obstacles appear differently from the driving surface. Typically, detection is based on colour, texture or shape cues and motion is ignored. These methods have low computational cost but are easily fooled in everyday scenarios.

Motion-based methods compare two or more images of the video stream and detect obstacles by virtue of their motion is differing from the ground's. These methods are more robust than appearance-based methods since they detect on the basis of 3D structure, rather than appearance.

Reconstruction-based methods work by explicitly reconstructing the 3D scene from motion. Obstacles are detected from image features that lie above the ground plane in the 3D model. Reconstruction-based methods are more complex and computationally expensive than the previous two categories, but offer a direct estimate of the obstacle's location in space. Therefore, in order to add range-finding capability to a rear parking camera, we have developed a reconstruction-based approach to obstacle detection.

1.2 Problem Description

The problem is the following. We want to design a vision-based collision avoidance system for vehicles parking in reverse. Video of the scene directly behind the vehicle is captured by a rear-mounted video camera, as shown in Figure 1.1.

The camera is mounted behind the vehicle, just above the rear license plate. It is tilted toward the ground to provide a good view of the ground directly behind the vehicle. The goal of the system is to detect static obstacles using only the visual input the rear-mounted camera and to alert the driver of any obstacles in the collision path of the vehicle.

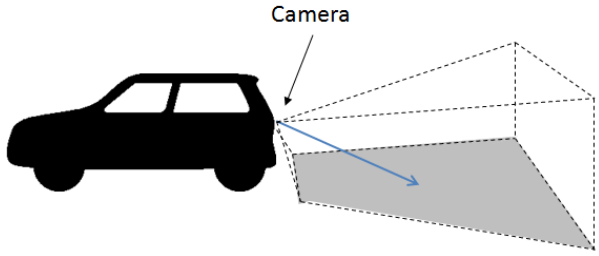


Figure 1.1: A rear-mounted parking camera setup. The camera is located just above the rear license plate and tilted toward the ground. The blue arrow indicates the camera's optical axis, the dotted lines indicate the camer's field of view and the gray area is the ground seen by the camera.

1.3 A Real World Consumer Product

This work was done in collaboration with CogniVue Corporation in Gatineau Quebec. CogniVue produces programmable System-on-Chip products, delivering industry-leading programmable video and imaging solutions for smart camera in the automotive and video monitoring industries. Because our algorithm would be ported to a dedicated image processing chip, it had to be as simple as possible and could not rely too heavily on complex computer vision libraries. Of course, it had to run in real-time and perform sufficiently to be released as a consumer product. This work has resulted in a U.S. patent application [16].

1.4 Overall OD System Architecture

We assume a calibrated camera and that the camera pose wrt. the vehicle is known and constant.

The front end of our system is feature tracking. We detect corner features and track them as they move across the image. We have designed our own system for the creation and deletion of tracked features to enhance OD robustness. Namely, remove erratically moving features and features that merge due to tracking error.

The vehicle motion is entirely estimated from the tracked features. We use the knowledge of the camera orientation wrt. the ground to model the motion as planar rigid motion. This allows us to estimate the motion from only two point correspondences. Outliers to the ground motion are removed via a RANSAC process. We use the knowledge of camera height to assign an absolute scale to camera displacement.

Based on the observed motion, we selected keyframes or *snapshots* from which the scene is reconstructed. These snapshots are meant to maintain a constant and sufficient baseline between the images, which increases the accuracy of reconstruction. As a result, the rate at which snapshots are taken depend on the speed of the vehicle.

We perform multiview triangulation between the current snapshot and all previous ones. For each snapshot pair, triangulation of a feature is only attempted if the feature has sufficient disparity in the image and if it moves along its epipolar line. As a result, moving obstacles and mistracked features are removed from the reconstruction process. In other words, only features moving according to the epipolar geometry are triangulated and associated with a 3D location.

After 3D reconstruction, each triangulated feature is given a semantic label based on its 3D location. Features near the ground plane are labelled “ground”. Those directly behind the vehicle are labelled “obstacle”. These labels, apart from their obvious use as detection flags, assist in subsequent camera motion estimates.

After reconstructed and labelling, the obstacle features are spatially clustered to reduce false detections and increase the accuracy of the distance-to-obstacle estimate. Features are clustered according to their distance from the vehicle in order to cluster large surfaces perpendicular to the ground.

The snapshots are triggered only after a sufficient displacement is observed. As a result, scene reconstruction does not occur every frame. To achieve a smooth frame by frame update of the 3D scene, we update the 3D location of features assuming their height is constant.

The distance to the nearest obstacle cluster is reported to the user.

1.5 Contributions

We have developped of a complete single-view driver-assistant system for the detection of obstacles that is both efficienct and reliable. Efficiency has been demonstrated through the

collaboration with Cognivue who have successfully implement the algorithm on its hardware architecture. Reliability has been demonstrated by extensive testing on real-world data using actual car mounted back up camera.

1.6 Thesis Organization

Chapter 2 reviews literature directly related to this thesis. We begin by reviewing methods of visual odometry, as motion estimation is central to 3D reconstruction techniques. We then provide an overview of visual obstacle detection, with a focus on monocular methods.

Chapter 3 covers fundamental concepts required for the understanding of subsequent chapters. These topics include rigid body motion, the pinhole camera model, and lens and sensor modeling.

Chapter 4 describes our motion estimation algorithm. The planar motion model is introduced. Then the estimation of its parameters is discussed.

Chapter 5 discusses 3D reconstruction from multiple views. First, epipolar geometry is presented; The epipolar identities are introduced and their relation to the epipolar constraint is discussed. The triangulation of image features is discussed and the necessary conditions for reliable triangulation is developed. The chapter concludes with a description of our triangulation algorithm.

Chapter 6 presents our algorithm in detail, drawing on concepts and sub-algorithms developed in Chapters 4 and 5. An overview of the algorithm is presented. Then each component is presented in detail.

Chapter 7 discusses the experimental setup and performance of our algorithm. The systematic optimization of system parameters is discussed. Then performance on selected videos is presented. The chapter concludes with an assessment of global performance for a set of test videos.

Finally, Chapter 8 summarizes our algorithm and results and describes possible future work.

Chapter 2

Literature Review

In this chapter we present a review of related work in obstacle detection from video. Since our own method relies on a robust estimate of vehicle motion from the visual data, we also review relevant work on visual odometry.

2.1 Visual Odometry

Measuring vehicle motion is crucial for any monocular 3D structure-based obstacle detection method. Triangulating image features accurately requires not only good feature tracking, but also good motion estimation. Motion estimation of the ground is particularly challenging because of the scarcity of strong features on its surface. As a result, many works have been published on the specific problem of estimating motion from vehicle-mounted cameras; A topic commonly referred to as *visual odometry*.

There are two main approaches to visual odometry; (1) those that estimate motion by observing the ground only and (2) those that estimate motion by observing all objects in the scene.

These approaches typically combine a motion model of the ground plane with a projective model to obtain a parametric model of the ground's optical flow. The optical flow parameters are fit by observing motion on the image, and these are related to the parameters of the motion model. Such approaches can be further divided by the method used to fit the optical flow model of the ground. Some track ground features and use outlier-robust methods to estimate the ground flow parameters [29]. Others use the optical flow over the ground surface [32]. Others use whole-image matching in the so-called *direct methods* [31, 10, 2, 21]. These ground-observation-only approaches have the advantage of being relatively simple to compute.

However, they are only suited for scenes in which the roadway is large and unobstructed.

Approaches that use all objects in the scene estimate motion by 3D scene reconstruction [13, 25]. The advantage to such methods is that they are able to cope low ground texture with scenes cluttered with static objects. The disadvantage is that these methods require costly bundle adjustment processes for sufficient robustness.

We describe these methods in greater detail and highlight exemplary works in the following sections.

2.1.1 Direct Methods for Visual Odometry

The direct method for visual odometry is meant to accommodate the fact that typical driving surfaces have few feature strong features. In particular, driving surfaces typically consist of self-similar, speckly texture with a high degree of ambiguity between image features [21]. So, it is argued, robust motion estimation cannot be achieved by matching individual features. Instead, whole-image registration is used to estimate ground motion. Robustness is increased by dividing the image into small patches and evaluating the confidence that each patch adheres to the motion model. Then a global image registration is performed, with each patch weighted by its confidence. This way, each pixel contributes a measurement to a global probability function for the parameters of the motion model. The advantage of the direct method is that it avoids feature tracking and computing optical flow.

Stein *et al*. 2000

As an example, Stein *et al*. [31] employ the direct method to fit a 3-parameter constrained planar motion model. The confidence in a given image patch is the ratio between the best fit metric using the previous motion estimate and the best fit using the best local motion estimate. The best fit is computed as $\exp(s/\sigma^2)$, where s is the smallest sum of squared difference (SSD) value for a window surrounding the image patch, and σ^2 is the variance of the image noise.

The basic algorithm is as follows. Given two consecutive images and an initial motion guess based on the previous motion estimate, for each image patch, do

1. warp patch in image 2 towards image 1 using the previous motion parameters
2. Compute the SSD for a 15×15 region surrounding the patch. Let the smallest SSD value be s_1.
3. For a small space of motions about the initial guess, find the best motion for each patch.

4. Warp each patch again, this time according to the best local motion.

5. Again, compute the SSD for a 15×15 region surrounding the patch. Let the smallest SSD value be s_2.

6. The confidence in the patch is the ratio between the best fit of the previous motion estimate ($\exp(s_1/\sigma^2)$) and the best fit of the local motion estimate ($\exp(s_2/\sigma^2)$):
   confidence $= \frac{\exp(s_1/\sigma^2)}{\exp(s_2/\sigma^2)}$

7. Search for the global motion that maximizes the weighted sum of all patch fits using gradient descent.

The result is an efficient whole-image registration technique that makes use of the previous motion estimate to speed up convergence. Although the authors provide evidence the algorithm provides accurate rotation estimates, they provide no ground-truth comparison for their distance estimates. Also, they do not discuss the tolerance of their motion estimation in the presence of significant ground occlusion.

Lovegrove *et al*. 2011

A more recent application of the direct method was proposed by Lovegrove *et al*. [21]. This work is of particular interest to us since it is a visual odometry method for a single rear parking camera. Instead of evaluating the fit of local patches, as above, they find the best homography between successive *entire* image pairs. Doing so, they assume the ground is completely unoccluded. This is a reasonable assumption since the algorithm is designed for when the vehicle is moving forward and so the area behind the car should be unobstructed. The best homography from one image to the other is found by minimizing the sum of squared pixel error between the homography-transform of the image (the synthetic image) and the second image (the true image). This homography, which ultimately encodes the vehicle's motion, is found iteratively using the *Efficient Second order Minimization* algorithm [24].

This algorithm provides robust and accurate camera motion estimation, given the ground is unoccluded. However, as obstacles begin to fill the image and occlude the ground, the minimization process fails to find the proper motion parameters. Of course, in our application we *expect* obstacles to occlude the ground as the vehicle approaches, and it is precisely these situations in which we need the motion estimation to work best. For this reason, we choose not to employ the method of Lovegrove *et al*. .

Scaramuzza *et al*. 2009

Scaramuzza *et al*. [29] exploit the fact that wheeled vehicles possess an instantaneous center of rotation to compute the epipolar geometry from a single feature correspondence. By using the knowledge of the vehicle speed they are able to recover the turning angle with a 1-point RANSAC. This results in an optimal motion estimation scheme in the sense that outlier removal procedures, such as RANSAC, converge with the smallest possible number of iteration. Of course, this can only be used if the vehicle speed is measured by some external sensor.

Similar to [29], we optimize the motion estimation by constraining the vehicle motion. However, since our system has no external measurement of the vehicle's speed, we cannot employ the 1-point RANSAC method. Instead we use a general planar motion model (Chapter 4) that results in a 2-point RANSAC method.

2.1.2 Visual Odometry from Optical Flow

Suzuki and Kanade 1999

Suzuki and Kanade [32] combine a velocity sensor and yaw rate sensor with a CCD camera to estimate camera motion from optical flow. Assuming a planar surface, they express the optical flow of ground pixels in terms of the rigid camera motion and the video frame rate. They assume a small change in the camera's pitch angle, roll angle, and camera height wrt. the ground while the vehicle is moving. This allows for a simplified expression of the optical flow parameters of the road from which the vehicle motion is more easily solved. They use an Extended Kalman Filter (EKF) to reduce measurement noise and facilitate solving the non-linear system. They model the vehicle dynamics as a second-order oscillatory motion.

To calculate the optical flow, they apply a Laplacian Gaussian Filter to the image, to enhance image features. Then, they compute the optical flow using a Sum of Absolute Difference (SAD) template matching. The parameters of the ground flow optical flow model are found using the Least Mean Square (LMS) method.

The ground flow parameters, along with the reading from the velocity and yaw rate sensors, are fed into the Extended Kalman Filter to obtain a minimum variance estimate of the vehicle's motion and orientation wrt. the ground.

They show accurate and stable results compared to a ground truth from external sensors. However, they used a high quality CCD camera mounted high on the vehicle's roof rack for a better view of the road, and used external sensors as input. Since none of these aiding factors are amenable to our application, and their contribution to the success of the above algorithm

was not assessed, we have not employed the method of Suzuki and Kanade.

2.1.3 Visual Odometry from 3D Reconstruction

Kitt *et al*. 2011

Kitt *et al*. [13] propose an approach to monocular visual odometry that uses knowledge of the camera mount and assumes a locally planar driving surface to correct drift in scale. They select Harris corners [8] using the *Good Features to Track* algorithm [30] and track these features across consecutive frames. These tracks are used to recover camera pose in two distinct phases; (1) Pose initialization and (2) pose estimation.

For pose initialization, three keyframes $\{K_1, K_2, K_3\}$ are selected and their corresponding feature tracks are used to recover camera pose and triangulate features. First, the 8-point algorithm [9] is used to estimate the pose between K_1 and K_3. Second, this pose estimate is used to triangulate the features, provided they lie near their respective epipolar lines in both images. Third, the pose between K_1 and K_2 is estimated using the triangulated feature points in an optimized PnP algorithm [18]. Finally, a bundle adjustment is performed over the three keyframes to obtain the best pose and triangulation estimate.

For pose estimation, they use the triangulated feature points, along with their current track observations, to update the pose of the camera w.r.t. to the first keyframe K_1

New keyframes are selected when the number of triangulated points drops below a threshold. When a new keyframe is added, a bundle adjustment is performed over the most recent 10 keyframes, and the most recent 5 pose estimates. Pose estimation then ensues until the next keyframe is added.

The scale drift in the translation estimate that may occur with such a reconstruction scheme is compensated for in the following manner. They use the knowledge of the camera height and orientation w.r.t. the ground to produce a bird's eye view of the ground for consecutive frames. By matching large (50×50) pixel blocks of this of this view from one image to the next, they are able to robustly track ground features. This tracking, combined with the camera motion estimate, allows the ground features to be triangulated and thus the camera height to be estimated. This camera height estimate is compared with the true camera height to give the corrective scaling for motion and structure.

Because the speed of a backing vehicle can range from nearly stationary up to 20km/h, we employ the concept of keyframes in our algorithm (we call them *snapshots*). Also similar to [13], we use the known height of the camera to set an absolute scale to the camera motion and scene reconstruction.

Nister 2004

Nister *et al*. [25] propose a system that estimates motion a single moving camera (as well as a stereo rig) in real-time based on video input. Their motion estimation scheme is completely general in that they make no prior assumptions about the camera motion, nor the type of terrain. Thus, they have been able to successfully apply their method to aircraft-mounted, vehicle-mounted and hand-held cameras. Their algorithm is as follows.

At each frame, Harris corners [8] are detected using a novel MMX implementation that optimizes cache performance. Non-max suppression in a 5×5 neighbourhood is used to identify actual feature points. Rather than using absolute thresholds on corner response, they set a limit on the number of detected features in a given local region. In particular, they break the image into 10 by 10 buckets and allow 100 features per bucket.

Tracking is achieved by matching detected features from frame to frame, rather than detecting once and tracking a window of pixels surrounding the feature. In order to keep the matching possibilities to a manageable size, a feature in one image is matched to every feature within a fixed distance in the other image. This fixed distance is typically 10% of the image size. Normalized correlation over an 11×11 window is used to evaluate potential matches. Accepted matches must pass the *mutual consistency check*. That is, feature a in one image is matched to feature b in the other image only if a is the best match for b and b is the best match for a.

Camera motion is estimated by a bundle adjustment scheme that iterates over time. The basic steps are the following.

1. Track features for a certain number of frames. Estimate the relative poses between three frames using the 5-point algorithm [26].

2. Triangulate the tracked features using the first and last observation on each track [28]. If this is not the first time through the loop, put the present reconstruction in the coordinate system of the previous one.

3. Track for an additional number of frames. Compute camera pose w.r.t. the known 3D points using the 3-point algorithm [7]

4. Re-triangulate the features, again using the first and last observation on each track. Repeat step 3 a certain number of times.

5. Repeat step 1.

In addition, a maximum track history is set to avoid long-term error propagation. That is, the track observation at the past frame corresponding to the track history maximum will be treated as the "first" observation. Also, the model fitting in steps 1, 2 and 3 are made robust with a modified RANSAC scheme developed by the author, and then iteratively refined.

Although the authors show impressive results, they only show results for the stereo version of the algorithm and give only the qualitative statement that it outperforms the monocular version. So the performance of the monocular algorithm is unclear. Further, the generality of their motion estimate necessitates the rather cumbersome bundle adjustment scheme shown in steps 1 through 5. This requires the implementation of three separate, highly specialized reconstruction algorithms [26, 28, 7]. Such an undertaking carries two major risks; (1) There is too great a risk of burying the novice implementor in technical computing details, and; (2) Porting such a scheme to the dedicated hardware system is not likely realizable within the project time frame. For these reasons, we choose not to employ the method of Nister *et al.* .

2.2 Obstacle Detection

In this section we review visual obstacle detection methods. We outline various methodologies for detection and focus on methods that use scene structure for detection, since this is how our algorithm works.

Obstacle detection methods can be largely divided into two categories; Those using stereo-vision and those using a single camera.

Stereo methods make use of the disparity field to infer the depth of image features. This allows for detection of the ground plane (if it is unknown) and the detection of obstacles as any feature off the ground plane.

Monocular methods form two broad classes; Appearance-based methods and Motion-based methods. Appearance-based methods use colour and shape cues to differentiate image regions belonging to the ground from regions belonging to obstacles. Motion-based methods, on the other hand, rely on image motion and use a variety of ways to detect and even locate obstacles. We new discuss these methods in greater detail.

2.2.1 Stereo Methods

As our method uses as single camera, we provide only a brief review of stereo methods for obstacle detection.

Stereo methods all use scene structure for detection in one way or another, since 3D reconstruction is the primary function of a stereo rig. Stereo-vision systems have the advantage of instantaneous depth perception. This means that no motion is required between the vehicle and the scene in order to detect obstacles based on 3D structure. However stereo methods require two cameras and a proper calibration between them in order to function. As a result, stereo OD systems are more expensive (both computationally and financially) and require more maintenance than monocular systems.

Jenkin and Jepson 1994

Jenkin and Jepson [12] proposed an OD method using a calibrated stereo rig. They use expectation maximization to fit a Gaussian mixture model for the disparity field obtained from the stereo image pair. Then, the probability of each pixel belonging to an obstacle is computed from the ownership probabilities of the mixture model. In an indoor environment, this system was shown to detect obstacles (textbooks, in fact) of very low height.

Stereo OD systems are preferred for autonomous navigation on undulated terrain [1] [33] [4]. Talukder *et al*. [33] reconstruct the terrain in front of the vehicle using a stereo rig. They cluster the terrain into locally planar regions and use the relative slopes among the planes to detect obstacles.

2.2.2 Monocular Methods

Monocular methods vary substantially more in detection mechanisms. At the highest level, monocular are either appearance based or motion based.

Appearance-based Methods

Appearance-based methods [34, 19, 11] use colour and shape to detect image regions belonging to obstacles and image motion is largely ignored. For example, Ulrich and Nourbakhsh [34] use only colour information to detect obstacles for the purpose of robot navigation. They classify each pixel as ground or obstacle based on how similar the pixel's colour is to the ground's. To this end, a hue and intensity histogram is built from a region near the bottom of the image assumed to be ground. If a given pixel colour is below a threshold in either of these histograms, it is considered an obstacle. The ground colour histograms are updated over time, based on odometry information coming from the robot.

The strength of appearance-based methods is speed, simplicity and the ability to detect very small obstacles. However, they suffer from underlying assumptions. First, obstacle detection can only occur if obstacles differ in appearance from the ground. Second, obstacles cannot occupy the area near the vehicle assumed to be ground, otherwise their appearance gets incorporated into the ground model. Further, obstacle distance can only be estimated if the obstacle is detected at its base. In other words, the distance to overhanging obstacles cannot be estimated.

All of these assumptions are often violated in a parking scenario. First, many obstacles are of similar colour to the ground (gray or concrete objects). Conversely, patterns painted on the ground, such as parking lines, would be falsely detected. Second, in parking situations obstacles will inevitably occupy the majority of the image. Third, at close range, even obstacles that overhang by a small amount, such as a car bumper, can lead to a false distance estimate that could cause a collision.

Motion-based Methods

Motion-based methods are complimentary to appearance-based methods in that they largely ignore colour and shape but rely heavily on the motion of image features and optical flow. Of the motion-based methods, some rely on external sensors to measure vehicle motion [5, 35] while others estimate vehicle motion directly from the images [37, 20, 14, 17, 36].

Obstacle detection in motion-based methods generally works in the following way. The ground, assumed to be planar, is expected to move a certain way on the image. A parametric model is constructed to capture the essence of this motion. Then, the ground motion is observed, with or without the help of external sensors, and the parameters of the motion model are estimated. Regions of the image that agree with this model are considered ground, and the rest are considered obstacles.

Zhang *et al*. 1997

A simple example of the motion-based method is the work of Zhang *et al*. [37]. They present two monocular methods for obstacle detection where the presence of outliers to the motion model is the detection mechanism. In the first method, the motion of a calibrated camera is described by a linear model, assuming a small angle of rotation between images. In the second method, the homography between two views of an uncalibrated camera is computed. In both cases the presence of outliers are used to detect the presence of an obstacle.

Enkelmann 1991

As a variation, Enkelmann [5] uses optical flow to detect stationary and moving obstacles with a single camera. In this approach, orientation of a single calibrated camera wrt. the ground is known and assumed to be fixed. The vehicle's speed, measured externally, is used to generate an expected optical flow field, assuming planar ground and no obstacles. Thus, when obstacles are present, the calculated optical flow on those regions of the image differ substantially from the expected flow. It is this difference that is used to trigger a detection.

Lourakis 1997

Another variant is given by Lourakis and Orphanoudakis [20]. They detect obstacles by differencing registered images of a camera moving on a planar surface. By corresponding image features across two consecutive images, they compute the ground homography using a *Least Median of Squares* (LMedS) estimator to remove outliers. The computed homography is used to map the ground in the previous image to that in the current image. This leaves a difference between the images wherever an obstacle is present. The difference image is thresholded and a minimum connected component size criterion is used to remove high frequency noise. Any remaining connected components indicates the presence of an obstacle.

Willersinn and Enkelmann 1997

Willersinn and Enkelmann [35] detect moving vehicles by spatially clustering similar optical flows. Specifically, they use the knowledge of the ground plane cluster optical flows along lines parallel to the ground, a technique developed with vehicle detection in mind. Multiple cluster hypotheses are developed and the best one is chosen. Each cluster is considered an object and its position and motion are estimated with a Kalman filter. The object position is obtained under the assumption that it is detected at its base.

Lefaix *et al*. 2002

One of the more elaborate motion methods was proposed by Lefaix *et al*. [17]. They detect both stationary and moving obstacles by motion consistency check over subregions of the image. They use 2D quadratic motion to model the projection of the planar motion of the ground on the image. A multi-resolution technique [27] is used to estimate the parameters of the model and produce an outlier map for each pixel. The outlier map is thresholded and detected pixels are grouped based on location and optical flow. To reduce false detections, detected obstacles are

tracked over successive frames. They estimate the time to collision of the detected obstacles by the rate of change of their bounding boxes.

Although the authors present a reasonable looking time-to-collision VS time plot of a detected vehicle, no ground truth is provided, so the accuracy of their system is uncertain. Further, their grouping of pixels with similar motion would fail to properly segment pedestrians that are near the camera, as the difference in motion of the various body parts would become appreciable. In fact, no results for close-range detection/time-to-collision estimation is presented. For these reasons, we did not pursue this particular method.

The strength of motion-based methods is the ability to detect obstacles based on scene structure *without an explicit 3D recovery of the scene*. In other words, these methods provide the robustness of detection from scene structure without the computational cost and complexity of 3D reconstruction methods. However, these methods do not provide a direct estimate of the obstacle distance. For this we must turn to 3D reconstruction methods.

2.2.3 Structure-based Methods

Structure-based methods detect obstacles by an explicit 3D reconstruction of the scene. For this to be possible with a single camera, the camera must be in motion. The camera motion creates a sequence of images of the scene taken from different poses. If the scene is static, then there is no distinction between a set of images taken from a single camera at different times, and a set of images taken from multiple identical cameras taken simultaneously.

The main tasks in most structure-based methods are (1) motion estimation, (2) 3D scene reconstruction, (3) establishing the ground plane, and (4) detecting obstacles as features above the ground plane.

Yamaguchi 2006

Yamaguchi *et al*. [36] propose a method for detecting moving obstacles on roads using a single vehicle-mounted camera. They use the 8-point algorithm [9] to estimate camera motion and use the detection results of previous frames to select features. At each frame, the motion between the current and the previous frames is estimated. To improve the robustness of the motion estimate, they divide the image horizontally into three sections motivated by the following assumptions: (1) The bottom region contains the road, (2) the middle region contains short objects (e.g. vehicles, pedestrians), (3) the top region contains tall objects (e.g. buildings, street signs). Features are selected such that the three regions are sufficiently represented. This scheme is not optimal, since dividing the image into such subregions implies some knowledge

of camera pose wrt. the road, yet no motion constraints reflecting this knowledge are used in the motion estimation. Again, here camera height is used to set an absolute scale to the motion. Regions of the ground are detected by correlating corresponding patches of the current and previous frames, with the patch of the previous frame transformed according to the estimated ground motion. If the correlation is high, the patch is considered part of the ground.

Moving obstacles are indicated by features that are away from their epipolar lines or that have a negative triangulated distance. For robustness of detection, spatio-temporal clustering is performed on detected features.

This literature review was done with the goal of finding the visual odometry and obstacle detection algorithms that give the best performance for the most simplicity. In this sense, no method (in either category) stands out among the others; The more performant systems are invariably the more complex in terms of both software and hardware. Our challenge was to develop and test a monocular obstacle detection algorithm and implement it on a dedicated hardward architecture, within a matter of months. In order to meet this challenge, we have opted for the simpler approach of tracking features. Tracking features provides a single mechanism with which we can do both motion estimation and obstacle detection.

Chapter 3

Preliminary Material

3.1 Rigid Body Motion

A *rigid body* is a body whose shape is constant in time. In other words, the relative position of points comprising the object remains constant. This property of rigid bodies allow us to fully describe their position in space at any given time by a single coordinate system. Consider a rigid body initially at position A, then moved to position B. Let $g : \mathrm{R}^3 \to \mathrm{R}^3$ be the mapping of points on the rigid body from A to B. In other words, if a point p has 3D coordinates $\mathbf{X}_A$ in position A and $\mathbf{X}_B$ in position B, then $\mathbf{X}_B = g(\mathbf{X}_A)$. Because the body is rigid, the distance between any two points, as well as their relative orientation, are preserved as they are displaced from A to B. To put it more formally, g preserves the norm of any vector (distance preservation) and the cross product of any two vectors (orientation preservation).

Definition 2.1 (Rigid Body Displacement). A map $g : \mathrm{R}^3 \to \mathrm{R}^3$ is a rigid body displacement *if it preserves the norm and the cross product of any two vectors*:

1. *norm:* $\|g(\mathbf{X})\| = \|\mathbf{X}\|, \ \forall \mathbf{X} \in \mathrm{R}^3$
2. *cross product:* $g(\mathbf{X}) \times g(\mathbf{Y}) = g(\mathbf{X} \times \mathbf{Y}), \ \forall \mathbf{X}, \mathbf{Y} \in \mathrm{R}^3$

It is shown in [23] that there are two types of transformations in R^3, that when composed together, make up the entire set of rigid body displacements; rotations and translations.

3.1.1 Rotations in R^3

A rotation $r : \mathrm{R}^3 \to \mathrm{R}^3$ is a distance and orientation preserving transformation for which there exists a line passing through the origin which remains invariant. This invariant line is called the *axis of rotation*. In this section we establish a notation for describing rotations.

Let us define a coordinate system S_0 with the orthonormal basis $B_0 = \{\mathbf{X}_0, \mathbf{Y}_0, \mathbf{Z}_0\}$. The unit basis vectors $\mathbf{X}_0$, $\mathbf{Y}_0$ and $\mathbf{Z}_0$, define the orientation of the x, y, and z axes of S_0. Let us define a second coordinate system S_1 whose origin coincides with that of S_0 and that has been rotated such that its basis vectors are given by $B_1 = \{\mathbf{X}_{01}, \mathbf{Y}_{01}, \mathbf{Z}_{01}\}$. Here, the subscript "01" is used to denote the fact that $\mathbf{X}_{01}$ is the x axis of S_1 in the system S_0, and likewise for y and z-axes.

Consider a point p with coordinates $\mathbf{P}_1 = [X_1\ Y_1\ Z_1]^{\mathrm{T}}$ in S_1. Its corresponding coordinates $\mathbf{P}_0 = [X_0\ Y_0\ Z_0]^{\mathrm{T}}$ in S_0 are given by a linear combination of corresponding coordinates and basis vectors in S_1;

$$\mathbf{P}_0 = X_1\mathbf{X}_{01} + Y_1\mathbf{Y}_{01} + Z_1\mathbf{Z}_{01}$$

This expression can be conveniently written as a matrix multiplication;

$$\mathbf{P}_0 = \mathrm{R}\mathbf{P}_1$$

where R_{01} is the 3×3 matrix whose columns are the basis vectors of S_1, *i.e.*,

$$\mathrm{R} = [\mathbf{X}_{01}\ \mathbf{Y}_{01}\ \mathbf{Z}_{01}]$$

We call R a *rotation matrix* since it is constructed from the rotated orthonormal basis B_1. Here, the subscript "01" indicates that R_{01} transforms coordinates in S_1 to coordinates in S_0.

Since any rotation matrix R is constructed from an orthonormal basis, as was done above, it follows that $\mathrm{R}^{\mathrm{T}} = \mathrm{R}^{-1}$. So the inverse rotation, *i.e.*, the transformation from S_0 to S_1, is given by

$$\mathbf{P}_1 = \mathrm{R}_{10}\mathbf{P}_0 \quad \text{where } \mathrm{R}_{10} \equiv \mathrm{R}_{01}^{\mathrm{T}}$$

3.1.2 A Formulation of Rigid Motion

Let us denote the rigid motion from coordinate system S_a to coordinate system S_b as g_{ba}. The rigid motion g_{ba} is a composition of a rotation R_{ba} and a translation $\mathbf{T}_{ba}$. Recall that the columns of R_{BA} are the basis vectors of S_A expressed in coordinates of S_B. Also note that $\mathbf{t}_{ba}$ is the origin of S_a expressed in coordinates of S_b. Consider a point p with coordinates $\mathbf{X}_a$ in S_a and $\mathbf{X}_b$ in S_b. We then have the following relation.

$$\mathbf{X}_b = g_{ba}(\mathbf{X}_a) = \mathrm{R}_{ba}\mathbf{X}_a + \mathbf{T}_{ba} \tag{3.1}$$

Rearanging the above expression yields the inverse rigid motion

$$\mathbf{X}_a = g_{ab}(\mathbf{X}_b) = \mathrm{R}_{ab}\mathbf{X}_b + \mathbf{T}_{ab} \tag{3.2}$$

where

$$\mathrm{R}_{ab} \equiv \mathrm{R}_{ba}^{T} \quad \text{and} \quad \mathbf{T}_{ab} \equiv -\mathrm{R}_{ba}^{T}\mathbf{T}_{ba}$$

3.2 Camera Modeling

The goal of camera modelling is to understand the projection of a 3D scene onto the camera's image sensor. This understanding allows us to make two useful predictions: (1) Given a 3D point in space, we may predict the location of its image on the sensor, and (2) Given a location on the sensor, we may predict the 3D line in space passing through the 3D point and the camera center. These predictions allow us to relate measurements made by the sensor to objects in the world; a requisite for 3D reconstruction.

The basic camera design is to place a flat image sensor inside a dark compartment, and to have light enter through a small aperture located in front of the sensor. The aperture is fitted with a lens to focus light onto the sensor. An image is formed when light enters the aperture, is deflected by the lens, and strikes the sensor. As it is intuitive to think of image formation as occuring in these three steps, we divide the overall camera model into three corresponding sub-models; (1) perspective projection, (2) lens deflection (or distortion), and (3) sensor geometry. We now describe these models.

3.2.1 Perspective Projection

Perspective projection is the projection of a 3D scene onto a plane in which all light rays pass through a single point. A cameras exhibits perspective projection the limit that its aperture size goes to zero. The concept of perspective projection is formalized in the following definition.

Definition 2.3 (Perspective Projection). For a coordinate system S, let us define a point o called the perspective origin and a 2D plane Π called the projection plane. For any point p, we call the line formed by o and p the projection line of p. A *perspective projection* is the mapping $\pi_{\{S,o,\Pi\}} : \mathrm{R}^3/\{o\} \rightarrow \mathrm{R}^2$ of a point p to the intersection of its projection line and the projection plane.

Consider an idealized camera with an extremely small aperture at the origin and the image plane (sensor) is is the $z = -d$ plane, where d is a positive value. This is the so-called *pinhole camera* and is shown in Figure 3.1. As can be seen, the image of 3D points are inverted on the image plane. More precisely, the image of a 3D point $\mathbf{X} = [X\ T\ Z]^T$ is given by

$$\mathbf{x} = -\frac{d}{Z}\mathbf{X}$$

where $\mathbf{x} = [x, y, 1]^{\mathrm{T}}$ is the homogeneous image coordinate. To do away with the inverting of the image, and to set a standard scale, we define the *ideal image plane* at $Z = 1$. Let us denote

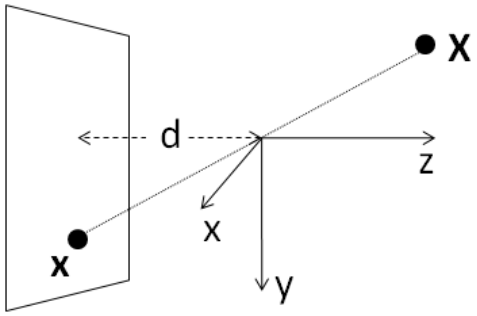


Figure 3.1: The pinhole camera model. The image sensor is normal to the z-axis and at a distance d from the aperture, which is at the origin. All light rays hitting the image sensor must pass through the origin. A point object $\mathbf{X}$ creates the inverted image $\mathbf{x}$.

the projection $\mathbf{x}$ of a 3D point $\mathbf{X}$ onto the ideal image plane by

$$\mathbf{x} = \pi(\mathbf{X}) = \frac{1}{Z}\mathbf{X} \tag{3.3}$$

Although the ideal image plane is not a physical one, it is extremely useful in 3D vision due to its simple projective relation to the world. In fact, the goal of camera calibration is to map pixels to the ideal image plane (and back) to take advantage of this simplicity. As we shall see, the theory of 3D imaging is most conveniently expressed using ideal image coordinates.

3.2.2 Lens Modelling

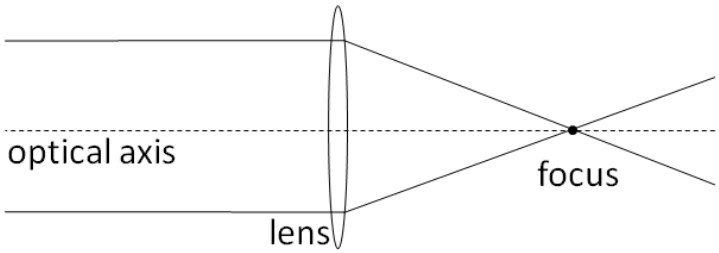


Figure 3.2: The thin lens is characterized by its focal length; The distance between the lens and the focus.

The purpose of a lens is to focus a large area of incident light onto a small sensor. It allows a camera to be compact while admitting large amounts of light. The simplest lens model is the *thin lens*, shown in Figure 3.2. In this idealization, the lens has an optical axis perpendicular to its surface and passing through its center. All incident rays that are parallel to the optical

axis are deflected to a single point on the optical axis and on the opposite side of the lens. This point is called the *focus*. The defining characteristic of the thin lens is the *focal length* f; The distance between the lens and the focus.

Placing a thin lenses at the aperture of a camera bends the incident light and effectively scales the image. This allows us to build cameras of practical size, with small image sensors placed near the aperture. Given a thin lens with focal length f, the ideal image point $\mathbf{x}$ is scaled according to

$$\mathbf{x}' = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x} \tag{3.4}$$

3.2.3 Sensor Modelling

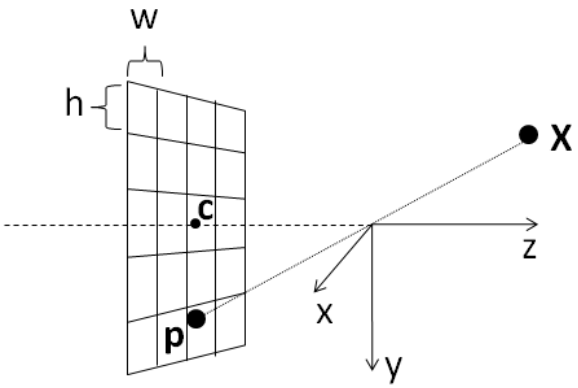


Figure 3.3: The image sensor model. Each pixel has width w and height h. The optical axis passes through the sensor at pixel $\mathbf{c}$. A point object $\mathbf{X}$ traces a ray to pixel $\mathbf{p}$.

The image sensor is typically a rectangular grid of discrete photo-sensitive elements called *pixels*. Each pixel on the sensor is labelled according to its discrete horizontal (x) and vertical (y) position. These coordinates, extended to the continuous domain are called *pixel coordinates*. We denote homogeneous pixel coordinates as $\mathbf{p} = [u, v, 1]^{\mathrm{T}}$.

Consider the image sensor shown in Figure 3.3. The pixel width and height, in meters, are w and h respectively. The optical axis passes through the point $[c_x, c_y, 1]^{\mathrm{T}}$ in pixel coordinates. For simplicity, let the sensor be located a distance 1m from the aperture (at the origin). This way, the sensor lies on the ideal image plane and so the pixel coordinates are given by

$$u = \frac{1}{w}x + c_x \qquad v = \frac{1}{h}y + c_y$$

where x and y are the ideal image coordinates. In homogeneous form we have

$$\mathbf{p} = \begin{bmatrix} \frac{1}{w} & 0 & c_x \\ 0 & \frac{1}{h} & c_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x} \tag{3.5}$$

3.2.4 Putting it All Together

When a thin lens and image sensor are combined, the ray traced from the world point $\mathbf{X}$ is related to the pixel coordinate $\mathbf{p}$ by combining (3.4) and (3.5):

$$\mathbf{p} = \begin{bmatrix} \frac{1}{w} & 0 & c_x \\ 0 & \frac{1}{h} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \frac{f}{w} & 0 & c_x \\ 0 & \frac{f}{h} & c_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}$$

The terms $f_x \equiv \frac{f}{w}$ and $f_y \equiv \frac{f}{h}$ are horizontal and vertical focal lengths, respectively, and are in pixel size units.

Finally, the ideal image coordinate $\mathbf{x}$ is related to the pixel coordinate $\mathbf{p}$ by

$$\mathbf{p} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x} \equiv \mathrm{C}\mathbf{x} \tag{3.6}$$

where C is called the *camera matrix*.

In addition to the linear mapping presented here, there is typically a non-linear component of the mapping due to lens distortion. However, lens distortion modelling is beyond the scope of this thesis and so we refer the reader to [3].

In this chapter we have discussed fundamental concepts required for the discussions in subsequent chapters. We have described rigid body motion and basic camera modeling.

Chapter 4

Motion Estimation

As we shall see in Chapter 5, recovering the depth of objects taken from two views requires knowledge of the motion between those two views. In the absence of external sensors, such as speedometers, accelerometers or GPS, we must estimate the motion from the visual data. In this section we begin by describing the special type of motion experienced by a vehicle-mounted camera. Next, we discuss how the specific properties of this motion allow us to describe and estimate the camera motion more easily than if it were a general rigid motion. Next, we discuss how to estimate the motion in the presence of noise and mis-measurements. We then give a brief description of feature tracking, the method with which we establish correspondences between two images. Finally, we combine these basic concepts in an algorithm for robustly estimating the motion between two images taken from a vehicle-mounted camera.

4.1 Camera Motion as Planar Motion

Any camera is, to any reasonable physical limit, a rigid object. So it is correct and sufficient to describe its motion as *rigid motion* (section 3.1.2).

Consider the two camera reference frames shown in Figure 4.1. The first camera C_1 is located at $\mathbf{T}$ in world coordinates and it is rotated such that such that the unit vectors along its x, y and z axes have world coordinates $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$, respectively. For convenience, and without loss of generality, the second camera C_2 is located at the world origin and its axes are aligned with the world's. The combined rotation and translation of C_1 gives a rigid motion $\{\mathrm{R}, \mathbf{T}\}$ from C_1 to C_2 where the three columns of R are the coordinates of the rotated axes $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$. From (3.2) we know the inverse rigid motion (from C_2 to C_1) is given by $\left\{\mathrm{R}^{\mathrm{T}}, -\mathrm{R}^{\mathrm{T}}\mathbf{T}]\right\}$. In other words, in the first camera frame, C_2 has coordinates $-\mathrm{R}^{\mathrm{T}}\mathbf{T}$, and its

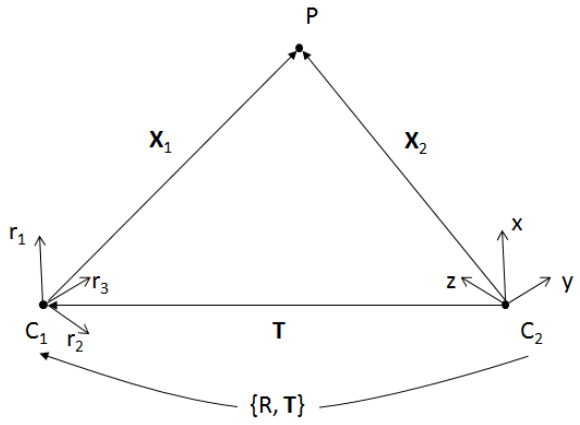


Figure 4.1: Two camera coordinate systems related by a rigid motion $\{\mathrm{R}, \mathbf{T}]\}$ whereby the first camera (C_1) has coordinates $\mathbf{T}$ in the second camera (C_2) frame, and its axes (r_1, r_2, and r_3) are the columns of the rotation matrix R.

axes are the columns of R^{T}.

Let the point P have coordinates $\mathbf{X}_1$ and $\mathbf{X}_2$ in the respective camera frames. These coordinates are related by the rigid motion between the cameras:

$$\mathbf{X}_2 = \mathrm{R}\mathbf{X}_1 + \mathbf{T} \tag{4.1}$$

It is precisely the rotation matrix R and the translation vector $\mathbf{T}$ that we need to know in order to recover depth of objects.

Now, we have considered two cameras giving two distinct views of a 3D scene. However, taking an image with a camera from C_1, then moving it to C_2 and taking another image is equivalent to having two identical cameras at C_1 and C_2. So the relative position of single, moving camera at different instants is also described by rigid motion.

For a vehicle-mounted camera, there is a special type of motion we can expect. Consider a vehicle moving around on flat ground. As long as the ground is flat and the wheels remain firmly on the ground, we have the following. No matter where the driver directs the vehicle, the vehicle's translation is within the ground plane. In other words, the translation is perpendicular to the ground normal. Also, no matter how the driver turns the wheel, the axis of the vehicle's rotation is in the same direction as the ground normal. As such, the vehicle is restricted move within the ground plane. We call this type of motion *planar*, and its formal definition is as follows.

Definition 4.1 (Planar Rigid Motion). A rigid motion $\{\mathrm{R}, \mathbf{T}\}$ is *planar* if there there exists

a vector $\mathbf{N}$ such that

$$\mathbf{T} \cdot \mathbf{N} = 0 \quad \text{and} \quad (\mathrm{R} - \mathrm{I})\mathbf{N} = \mathbf{0}$$

In that case, we say the motion is *within the plane* defined by the normal $\mathbf{N}$.

Planar motion is most conveniently expressed in a coordinate system in which the normal is aligned with one of the axes. By choosing to align the normal with the z axis, we may represent planar motion as $\{\mathrm{R}_p, \mathbf{T}_p\}$, where

$$\mathrm{R}_p = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_p = \begin{bmatrix} t_x \\ t_y \\ 0 \end{bmatrix}. \tag{4.2}$$

Here θ is the angle of rotation and $[t_x, t_y]^{\mathrm{T}}$ is the translation within the plane.

Now we ask; If we describe the vehicle's motion by $\{\mathrm{R}_p, \mathbf{T}_p\}$, what is the motion in the camera reference frame? In other words, what is $\{\mathrm{R}, \mathbf{T}\}$?

Qualitatively, we know the camera experiences planar motion because the pose of the camera relative to the vehicle is fixed, so the motion of the camera and the vehicle wrt. the ground is the same. For a quantitative answer, we need to define the pose of the camera wrt. the vehicle.

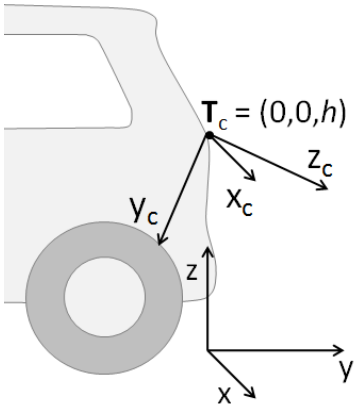


Figure 4.2: The vehicle and camera coordinate systems. The vehicle axes are x, y and z and the camera axes are x_c, y_c and z_c. The location of the camera center in vehicle coordinates is $\mathbf{T}_c$.

Figure 4.2 shows our choice for the vehicle and camera coordinate systems. The vehicle axes are denoted x, y and z while the camera axes are denoted x_c, y_c and z_c.

We locate the origin of the vehicle coordinate system on the ground and centered directly beneath the rear bumper of the vehicle. The vehicle's direction of travel, which we'll simply call its *axis*, is aligned with the y-axis. The z axis is normal to the ground.

The camera pose is defined relative to the vehicle coordinate system. The camera is mounted at the rear of the vehicle and its optical axis points in the z_c direction. We assume the camera is at the center of the rear bumper at a height h off the ground. So the camera center is located at $\mathbf{T}_c = [0\ 0\ h]^{\mathrm{T}}$ in vehicle coordinates. The rotation of the camera wrt. the vehicle is given by the rotation matrix R_c, where the columns of R_c are the coordinates of the camera's x, y and z axes. We assume the camera height and orientation are known via some prior calibration process.

With R_c and $\mathbf{T}_c$ defined as such, the camera coordinates $\mathbf{X}$ and vehicle coordinates $\mathbf{X}'$ of a point are related by

$$\mathbf{X}' = \mathrm{R}_c\mathbf{X} + \mathbf{T}_c \tag{4.3}$$

Now let us relate the planar vehicle motion to the camera motion. Say a point has coordinates $\mathbf{X}'_1$ wrt. the vehicle in the first instant, and $\mathbf{X}'_2$ wrt. the vehicle in the second instant. These are related by

$$\mathbf{X}'_2 = \mathrm{R}_p\mathbf{X}'_1 + \mathbf{T}_p$$

By substituting $\mathbf{X}'_1$ and $\mathbf{X}'_2$ for their expressions in (4.3), we get

$$\begin{aligned} \mathrm{R}_c\mathbf{X}_2 + \mathbf{T}_c &= \mathrm{R}_p\left(\mathrm{R}_c\mathbf{X}_1 + \mathbf{T}_c\right) + \mathbf{T}_p \\ \mathbf{X}_2 &= \mathrm{R}_c^{\mathrm{T}}\mathrm{R}_p\mathrm{R}_c\ \mathbf{X}_1\ +\ \mathrm{R}_c^{\mathrm{T}}\left(\left(\mathrm{R}_p - \mathrm{I}\right)\mathbf{T}_c + \mathbf{T}_p\right) \end{aligned}.$$

Notice that the above expression has the same form as (4.1). So the camera motion $\{\mathrm{R}, \mathbf{T}\}$, in terms of the vehicle's planar motion $\{\mathrm{R}_p, \mathbf{T}_p\}$ and the relative orientation of the camera w.r.t. the vehicle $\{\mathrm{R}_c, \mathbf{T}_c\}$ is

$$\mathrm{R} = \mathrm{R}_c^{\mathrm{T}}\mathrm{R}_p\mathrm{R}_c \qquad \text{and} \qquad \mathbf{T} = \mathrm{R}_c^{\mathrm{T}}\left(\left(\mathrm{R}_p - \mathrm{I}\right)\mathbf{T}_c + \mathbf{T}_p\right) \tag{4.4}$$

4.2 Estimating Motion on a Planar Surface

In the absence of external motion sensors, the motion parameters θ, t_x, t_y are not known and must be inferred by observing the motion of the image. Our strategy for estimating the motion of a vehicle-mounted camera is the following. We make three main assumptions; (1) the ground surface is planar, (2) the wheels of the vehicle remain in contact with the ground so that the camera exhibits planar motion, and (3) the height and orientation of the camera wrt. the ground is known and fixed. We estimate the vehicle's planar motion by tracking image features. Using the knowledge of camera pose, tracked features are projected onto the ground in vehicle coordinates. The planar motion parameters in (4.2) are estimated from the ground-projected

features in a process robust to model outliers. Finally, the vehicle motion is transformed to camera motion by (4.4).

Say we observe a set of points p_i, $i = 1...N$ on a plane with homogeneous coordinates $\mathbf{x}_1^i$ in the first reference frame and $\mathbf{x}_2^i$ in the second reference frame. The two sets of coordinates are related by the planar motion between the reference frames:

$$\mathbf{x}_2^i = \mathrm{R}_p \mathbf{x}_1^i + \mathbf{T}_p \qquad i = 1...N \tag{4.5}$$

Subtracting $\mathbf{x}_2^i$ from both sides of (4.5) gives an expression equal to the zero vector, which we call a *residual*:

$$\mathbf{0} = \mathrm{R}_p \mathbf{x}_1^i + \mathbf{T}_p - \mathbf{x}_2^i \qquad i = 1...N$$

As such, the true motion parameters can be found by minimizing the sum of squared magnitudes of all the residuals

$$\left\{\hat{\theta}, \hat{t_x}, \hat{t_y}\right\} = \arg \min_{\{\theta, t_x, t_y\}} \sum_i \left\| \mathrm{R}_p \mathbf{x}_1^i + \mathbf{T}_p - \mathbf{x}_2^i \right\|^2 \tag{4.6}$$

Linear Approximation

The residual in the above expression is non-linear with respect to θ and so the solution must be arrived at iteratively. However, we may arrive at an approximate solution by substituting α for $\cos(\theta)$ and β for $\sin(\theta)$. This removes the non-linear constraint $\cos^2(\theta) + \sin^2(\theta) = 1$ and gives the residual

$$\begin{bmatrix} r_x^i \\ r_y^i \end{bmatrix} = \begin{bmatrix} \alpha x_1^i - \beta y_1^i + t_x - x_2^i \\ \beta x_1^i + \alpha y_1^i + t_y - y_2^i \end{bmatrix}$$

and the linear least squares minimization

$$\left\{\bar{\alpha}, \bar{\beta}, \bar{t_x}, \bar{t_y}\right\} = \arg \min_{\{\alpha, \beta, t_x, t_y\}} \sum_i \left[\left(r_x^i\right)^2 + \left(r_y^i\right)^2 \right] \tag{4.7}$$

The expression $R = \sum_i \left[\left(r_x^i\right)^2 + \left(r_y^i\right)^2 \right]$ is minimized when all its the partial derivatives of are zero:

$$\frac{\partial R}{\partial \alpha} = 2 \sum_i \left(r_x^i\right) x_1^i + 2 \sum_i \left(r_y^i\right) y_1^i = 0 \qquad \frac{\partial R}{\partial t_x} = 2 \sum_i r_x^i = 0$$
$$\frac{\partial R}{\partial \beta} = 2 \sum_i \left(r_y^i\right) x_1^i - 2 \sum_i \left(r_x^i\right) y_1^i = 0 \qquad \frac{\partial R}{\partial t_y} = 2 \sum_i r_y^i = 0$$

If we express this system in matrix form $\mathrm{A}[\alpha, \beta, t_x, t_y]^{\mathrm{T}} = \mathrm{B}$, we get

$$\mathrm{A} = \begin{bmatrix} \sum \left({x_1^i}^2 + {y_1^i}^2 \right) & 0 & \sum x_1^i & \sum y_1^i \\ 0 & \sum \left({x_1^i}^2 + {y_1^i}^2 \right) & -\sum y_1^i & \sum x_1^i \\ \sum x_1^i & -\sum y_1^i & N & 0 \\ \sum y_1^i & \sum x_1^i & 0 & N \end{bmatrix} \quad \text{and} \quad \mathrm{B} = \begin{bmatrix} \sum (x_1^i x_2^i + y_1^i y_2^i) \\ \sum (x_1^i y_2^i - y_1^i x_2^i) \\ \sum x_2^i \\ \sum y_2^i \end{bmatrix} \tag{4.8}$$

Then the solution to (4.7) is

$$[\bar{\alpha}, \bar{\beta}, \bar{t_x}, \bar{t_y}]^{\mathrm{T}} = \mathrm{A}^{-1}\mathrm{B} \tag{4.9}$$

where A and B are given by (4.8).

Newton-Gauss Method

The approximate solution can then be refined by an iterative Newton-Gauss method. For a sum of squared normed residuals $R(\mathbf{p}) = \sum_i^N \|\mathbf{r}_i(\mathbf{p})\|^2$, where $\mathbf{p}$ is the set of parameters, the Newton-Gauss method says that we can converge to the optimal parameter set $\hat{\mathbf{p}}$ by iterating

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \left(\mathrm{J}^{\mathrm{T}}\mathrm{J}\right)^{-1} \left(\mathrm{J}^{\mathrm{T}}\mathbf{r}(\mathbf{p}_k)\right)$$

where J is the Jacobian of each residual

$$\mathrm{J}_{ij} = \frac{\partial \mathbf{r}_i}{\partial \mathbf{p}_j},$$

$\mathbf{r}(\mathbf{p})$ is the concatenation of all residual vectors

$$\mathbf{r}(\mathbf{p}) = \left[\mathbf{r}_1(\mathbf{p})^{\mathrm{T}}\ \mathbf{r}_2(\mathbf{p})^{\mathrm{T}}\ ...\ \mathbf{r}_N(\mathbf{p})^{\mathrm{T}}\right]^{\mathrm{T}},$$

and the initial guess $\mathbf{p}_0$ is close enough to $\hat{\mathbf{p}}$.

We use the results of the linear approximation (4.9) for the initial guess. Specifically, we use $\left\{ \text{sign}(\bar{\beta}) \cos^{-1}(\frac{\bar{\alpha}}{\sqrt{\bar{\alpha}^2+\bar{\beta}^2}}), \bar{t_x}, \bar{t_y} \right\}$, where the rescaling of $\bar{\alpha}$ enforces the non-linear constraint $\bar{\alpha}^2 + \bar{\beta}^2 = 1$.

In practice, convergence is reached when the updated parameters differ from the previous ones less than some threshold ϵ, that is, when $\|\mathbf{p}_{k+1} - \mathbf{p}_k\| < \epsilon$.

4.3 The RANSAC Algorithm

If some of the observed points $\mathbf{x}_1^i$ and $\mathbf{x}_2^i$ are corrupted by noise or are mis-measured, the estimated motion found by least squares minimization will be incorrect. In fact, such minimizations can be very sensitive to a small minority of model outliers. We therefore require a method of excluding outliers from the least squares fitting.

One such method is the Random Sample Consensus (RANSAC) algorithm. RANSAC estimates the parameters $\{p_1, ..., p_m\}$ of a mathematical model $\mathbf{y} = f(\mathbf{x} : \{p_1, ..., p_m\})$ from a dataset containing outliers. An *outlier* is a data pair $(\mathbf{x}_i, \mathbf{y}_i)$ that does not agree with the model. We detect outliers as data pairs whose normed residual $\|\mathbf{y}_i - f(\mathbf{x}_i)\|$ (a.k.a. reprojection error) exceeds some threshold ϵ_r. Conversely, any data pair with a normed residual less than ϵ_r is considered an *inlier*.

The basic RANSAC algorithm is as follows.

Algorithm 4.1 (RANSAC).
For a given set of data pairs $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1...N$, a model $\mathbf{y} = f(\mathbf{x} : \{p_1, ..., p_m\})$ and a threshold ϵ_r, this algorithm finds the set of model parameters $\{\hat{p}_1, ..., \hat{p}_m\}$ that maximize the number of inliers to the model.

1. Randomly sample the minimum number of data pairs required to solve the model parameters.
2. Solve the model parameters to obtain the candidate parameters $\{p_1, ..., p_m\}$.
3. Build a list of inliers for the candidate parameters. For each data pair, compute the squared residual $r_i = \|\mathbf{y}_i - f(\mathbf{x}_i : \{p_1, ..., p_m\})\|$. If $r_i < \epsilon_r$, add the i^{th} data pair to the list of inliers.
4. If the current list of inliers is largest found so far, keep it, otherwise discard it.
5. Repeat steps 1-4 M times
6. Solve the model parameters $\{\hat{p}_1, ..., \hat{p}_m\}$ for the largest list of inliers found. Return these parameters.

The remaining issue is: How do we set the maximum number of iterations M? The idea is to set M such that we are fairly confident that after M tries, at least one of the samples (obtained in step 1) has no outliers. In other words, we would like to be confident that we've managed to pick at least one subset of inliers.

Say the chance of picking an inlier from the data is u. Assuming a large dataset, the chance of picking k inliers is u^k. So the chance of picking at least one outlier among k samples is $1 - u^k$. Now, the chance of picking at least one outlier, M times in a row, is $(1 - u^k)^M$. This means that the chance of picking *no outliers at least once* after M tries is

$$P = 1 - (1 - u^k)^M.$$

The probability P, called the *confidence*, is a parameter of the RANSAC algorithm. We use $P = 0.95$. Solving for M gives

$$M = \frac{\log(1 - P)}{\log(1 - u^k)} \tag{4.10}$$

In general, the chance of picking an inlier u is not known, so we begin RANSAC assuming u is very small (making M very big). Then at each iteration, if the number of inliers n is largest we've seen so far, we set $u = n/N$ and re-evaluate (4.10), which gives a new M that is smaller than the previous. The result is an adaptive maximum iteration value.

4.4 World Plane Projection

In the two preceding sections, we have described a method for robustly estimating planar motion by observing the motion of an ensemble of points on the plane. However, the camera only captures the *images* of these points. That is, we only observe the projection of these points onto the image plane. We must then find a way of projecting images points *back* to their actual location on the plane. This task can be simplified by choosing a world coordinate system such that the plane is given by $Z = C$ where C is some constant. In that case our problem is the following: Given given the image $\mathbf{x}$ of a point with known world z coordinate, what are its world x and y coordinates?

Consider a camera placed at $\mathbf{T}_c = [X_c, Y_c, Z_c]^{\mathrm{T}}$ in the world and oriented such that its axes are given by $\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$, as shown in Figure 4.3. With this configuration, the transformation from camera coordinates to world coordinates is given by $\{\mathrm{R}_c, \mathbf{T}_c\}$, where $\mathrm{R}_c \equiv [\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3]$, that is, the columns of the rotation matrix are the world coordinates of the camera's axes. The transformation from camera coordinates to world coordinates is given by (4.3). We will find it useful to define the three rows of R_c as $\mathbf{r}_1^{\mathrm{T}}$, $\mathbf{r}_2^{\mathrm{T}}$ and $\mathbf{r}_3^{\mathrm{T}}$, respectively. Geometrically, the three rows of R_c are the world x, y and z axes, respectively, expressed in camera coordinates.

Say a point with world coordinates $\mathbf{X}_w = [X_w, Y_w, Z_w]^{\mathrm{T}}$ produces an image $\mathbf{x}_w = [x_w, y_w, z_w]^{\mathrm{T}}$, also expressed in world coordinates. Due the projective nature of the camera, there is a straight

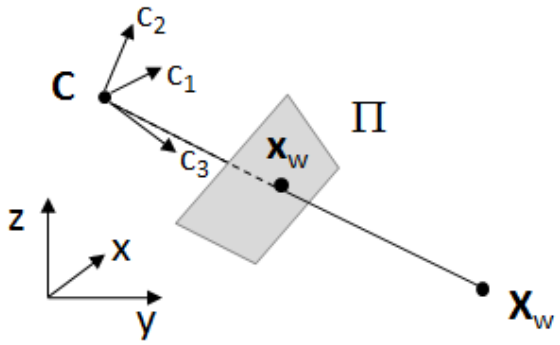


Figure 4.3: The relation between camera pose, a world point $\mathbf{X}_w$, its image $\mathbf{x}_w$ on the ideal image plane Π. The camera is rotated such that its x, y and z axes are $\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$, respectively, and it is located at $\mathbf{T}_c$ in world coordinates. Due to the projective camera model, a straight line passes through $\mathbf{T}_c$, $\mathbf{x}_w$ and $\mathbf{X}$.

line passing through the point, its image, and the camera center. We can express this line parametrically as

$$\mathbf{l} = t\left(\mathbf{x}_w - \mathbf{T}_c\right) + \mathbf{T}_c$$

with t as the free parameter. Since the line passes through $\mathbf{X}$, we have

$$\begin{aligned} \mathbf{X}_w &= t\left(\mathbf{x}_w - \mathbf{T}_c\right) + \mathbf{T}_c \\ \mathbf{X}_w - \mathbf{T}_c &= t\left(\mathbf{x}_w - \mathbf{T}_c\right) \\ \begin{bmatrix} X_w - X_c \\ Y_w - Y_c \\ Z_w - Z_c \end{bmatrix} &= t \begin{bmatrix} x_w - X_c \\ y_w - Y_c \\ z_w - Z_c \end{bmatrix} \end{aligned}$$

Say we know Z_w the world z component of the point. Then t can be solved for using the z component in the above expression,

$$t = \frac{Z_w - Z_c}{z_w - Z_c},$$

and then used to solve for the unknown x and y components:

$$\begin{bmatrix} X_w \\ Y_w \end{bmatrix} = \frac{Z_w - Z_c}{z_w - Z_c} \begin{bmatrix} x_w - X_c \\ y_w - Y_c \end{bmatrix} + \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \tag{4.11}$$

Now, in practice we do not obtain the world coordinates of the image directly. Instead, we observe the image $\mathbf{x}$ on the ideal image plane in *camera coordinates*. Transforming these to

world coordinates gives

$$\mathbf{x}_w = \mathrm{R}_c\mathbf{x} + \mathbf{T}_c = \begin{bmatrix} \mathbf{r}_1 \cdot \mathbf{x} + X_c \\ \mathbf{r}_2 \cdot \mathbf{x} + Y_c \\ \mathbf{r}_3 \cdot \mathbf{x} + Z_c \end{bmatrix},$$

which, when substituted into (4.11), gives the following relation.

Definition 4.2 (World plane projection). For a camera whose reference frame with respect to the world's is given by the rigid motion $\left\{[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3]^{\mathrm{T}}, [X_c\ Y_c\ Z_c]^{\mathrm{T}}\right\}$, the projection of the image $\mathbf{x}$ to the world $z = Z_w$ plane has world (x, y) coordinates

$$\begin{bmatrix} X_w \\ Y_w \end{bmatrix} = w(\mathbf{x}, Z_w) \equiv \frac{Z_w - Z_c}{\mathbf{r}_3 \cdot \mathbf{x}} \begin{bmatrix} \mathbf{r}_1 \cdot \mathbf{x} \\ \mathbf{r}_2 \cdot \mathbf{x} \end{bmatrix} + \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \tag{4.12}$$

4.5 Feature Tracking

So far we've discussed how to estimate planar motion between two images from corresponding points in both images. What we haven't discussed is *how* to correspond, or match, these points. That is, how do we make sure that two given points, one on each image, are both the projections of the same point in space? There are two main approaches to this problem, depending on how far apart the corresponding points are expected to be.

If there are large displacements of the camera between images, we can expect the corresponding image points to be far apart. Not only that, but if the view of the scene is significantly different, the relative positioning of corresponding points in each image is likely to be different. In this case, the approach is to find the best global matching of all image features between the two images.

However, if there are small displacements of the camera between images, we can expect the corresponding image points to be close by. Further, if the images are taken within a small time interval, the appearance of corresponding points are likely similar. This is precisely the case for our application, so we discuss it in more detail.

Say a feature has an appearance $I_t(x, y)$, where I_t is the grayscale intensity function at time t and (x, y) are image coordinates. If, at the next time, the feature has moved by (u, v), we would have

$$I_{t+1}(x + u, y + v) = I_t(x, y)$$

Note here that we assume the intensity function of the feature is shifted, but otherwise unchanged from one instant to the next. This assumption generally holds for slow-moving features captured in high frame-rate videos. A Taylor series approximation of the above equation

gives the fundamental *optical flow constraint*.

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0$$

This constraint is exploited by the Lucas-Kanade feature tracking (KLT) algorithm [22]. KLT makes the further assumption that the displacement of all points in the neighbourhood of a feature is the same. As a result, the optical flow constraint can be imposed on a neighbourhood of points for a unique unknown displacement (u, v). This gives an overdetermined system which can be solved in a least-square sense.

For larger displacements, the KLT can be made more robust by iteratively tracking features over different scales. For example, a $r \times c$ image may be subsampled to form a $r/2 \times c/2$ image. A feature can be tracked by KLT on the smaller image, and its new position is used as an initial guess for the larger one. This process can happen recursively over multiple scaled images, called *pyramids*. Typical applications use three pyramids for reliable tracking.

The remaining question is: How are features identified in the first place? Since, when we track a feature, we are looking for displacements along any direction in the image, our features ought to be distinguishable (at least locally) in all directions. If the feature has uniform intensity this is certainly not the case. Even if the feature is on an edge, its exact position along the edge is indistinguishable from others. However, if the feature is a corner then there is no ambiguity in its location in the next image, so it can be found. We therefore use corners as features.

A *corner* is defined as a point on the image for which the intensity change is high in more than one direction. We can detect this by computing the spatial gradient of the image around a point. If the spatial derivative in the direction orthogonal to the gradient is also large, then the point is a corner. A well known algorithm for detecting corners is the Harris Corner Detector [8].

When looking for features for tracking, criteria such as the spacing between features, or the number of desired features are very useful. *GoodFeaturesToTrack* (GFTT) [30] is an algorithm that sorts and filters features found by a feature detector based on three user criteria. These criteria are (1) the maximum number of features to be returned, (2) the minimum allowed distance between features and (3) the relative quality level. The quality level works as follows. Each feature has a score s, evaluated by some function particular to the feature detector being used. If q is the quality level, and the maximum score of all the features is s_{max}, than any feature with a score ratio $s/s_{max} > q$ is returned.

In our application, we use a Harris Corner Detector wrapped in GoodFeaturesToTrack to generate features for tracking.

4.6 Planar Motion from Images

Finally, we bring together the concepts discussed in this chapter to form an algorithm for robustly estimating camera motion from tracked features. For the first image, N features are identified using the GFTT algorithm. For each subsequent image, the Lucas-Kanade feature tracking algorithm updates the features' positions. Say we want to estimate the motion between image t_1 and image t_2. Our image pairs are then $(\mathbf{x}_1^i, \mathbf{x}_2^i)$, $i = 1...N$, the position of each feature at t_1 and t_2, respectively. We then use the following algorithm to estimate the camera motion.

Algorithm 4.2 (Planar Motion Estimation from Image Point Correspondences).
For a given set of image point pairs $(\mathbf{x}_1^i, \mathbf{x}_2^i)$, $i = 1...N > 2$ of points with known world z coordinates Z_i, this algorithm finds the planar motion of the camera that best fits the data, excluding any outliers.

1. **Transform image points to the world**
   For each image pair $(\mathbf{x}_1^i, \mathbf{x}_2^i)$ get the 2D world point pair $(\mathbf{g}_1^i, \mathbf{g}_2^i)$, where $\mathbf{g}_j^i = w(\mathbf{x}_j^i, Z_i)$ and $w(\cdot)$ is given by (4.12). Here, either the world z-components are known or they are assumed to be zero (on the ground).

2. **Estimate the planar motion**
   Run RANSAC (Algorithm 4.1) for the point pairs $(\mathbf{g}_1^i, \mathbf{g}_2^i)$, $i = 1...N > 2$ to get the motion parameters $\{\theta, t_x, t_y\}$ of the model (4.5).

3. **Construct the planar motion identities**
   Construct $\{\mathrm{R}_p, \mathbf{T}_p\}$ according to (4.2).

4. **Transform the motion to camera coordinates**
   Compute $\{\mathrm{R}, \mathbf{T}\}$ according to (4.4).

In this chapter, we have discussed our method for robustly estimating the planar motion of a vehicle-mounted camera from tracked image features. We use the knowledge of camera orientation to project the tracked features to world ground coordinates, where the estimation of the planar motion parameters is simplified. Further, we use the knowledge of the camera height to resolve scale ambiguity. The above algorithm is central to the motion estimation estimation module (Section 6.3) of our algorithm.

Chapter 5

3D Reconstruction

The theory of 3D reconstruction (or 3D vision) developped below assumes a calibrated camera. As mentioned above, a camera is calibrated if the mapping from image sensor to the ideal image plane (and back) is known. As a result, the theory can be neatly developed without any reference to image sensors, or lense distortion.

5.1 Epipolar Geometry

Epipolar geometry is the geometrical relation between two camera views. It tells us what we can and can't observe when imaging the same object from two locations; a set of rules we call the *epipolar constraint*. In this section we begin with a purely geometric and, hopefully, intuitive presentation of the various identities that make up epipolar geometry. Once these identities are introduced, we quantify them by adding coordinates and describing the change in position of the two cameras in terms of a rigid motion. By the end of this section, we will have both a geometric understanding and an algebraic representation of the epipolar constraint.

5.1.1 The Epipolar Identities

The Epipolar Plane

Consider two cameras and a point object positioned arbitrarily as in Figure 5.1. Consider the camera centers C_1 and C_2 and the point-object P. These three points define a plane called the *epipolar plane*. If we consider the two camera centers fixed, then every point in space defines an epipolar plane. As we shall see, the epipolar plane is very useful in establishing an intuititive relationship between two camera views.

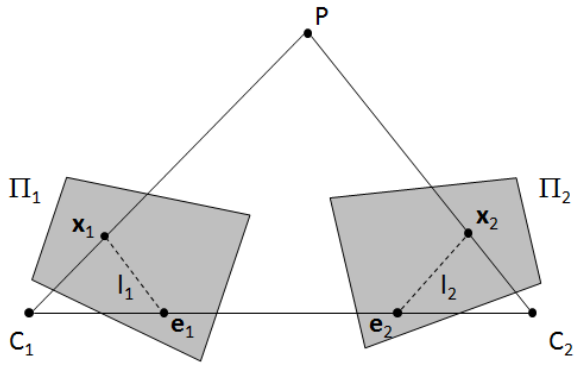


Figure 5.1: The geometry resulting from two cameras imaging the same scene is called *epipolar geometry*. The plane passing through the camera centers C_1 and C_2 and the object P is the *epipolar plane*. The intersection of the epipolar plane and the ideal image planes Π_1 and Π_2 form the *epipolar lines* $\mathbf{l}_1$ and $\mathbf{l}_2$, respectively. The projections of P onto each image plane are the *images* $\mathbf{x}_1$ and $\mathbf{x}_2$. The projections of cameras onto the other's image plane are the *epipoles* $\mathbf{e}_1$ and $\mathbf{e}_2$. Note that the epipolar lines pass through their respective epipoles and images.

The Epipolar Lines

Consider now the ideal image plane of each camera, Π_1 and Π_2. The intersection of the epipolar plane with each image plane forms a line $\mathbf{l}_1$ and $\mathbf{l}_2$ on the respective image plane. We call $\mathbf{l}_1$ (resp. $\mathbf{l}_2$) the *epipolar line* of image one (resp. two).

An alternative interpretation of the epipolar lines is the following. Consider the ray traced from C_1 to P, and imagine it is visible. How, then, would the ray look in the second image? Since the ray lies in the epipolar plane, its projection on the second image would be precisely the epipolar line $\mathbf{l}_2$. Similarly, $\mathbf{l}_1$ is the image of the ray joining C_2 and P.

What is the significance of the epipolar lines? First, we call the projections of P onto each image plane, $\mathbf{x}_1$ and $\mathbf{x}_2$, the *images* of P, and note that they lie on the epipolar lines. Now, say we were to move the object arbitrarily *within* the same epipolar plane. How would the images change? Figure 5.2 shows the images formed by moving P to P' within the same epipolar plane, resulting in the new images $\mathbf{x}'_1$ and $\mathbf{x}'_2$. We see that new images also lie on the epipolar lines. This must be true, since P' is on the epipolar plane, and the epipolar lines are, by definition, the intersection between the epipolar plane and the respective image planes. So the epipolar lines tell us where on each image we can expect to find the projection of any 3D point within the epipolar plane. A useful consequence of this is the following. Say the image $\mathbf{x}_1$ of an object is observed in the first image. If we wanted to find the same object in

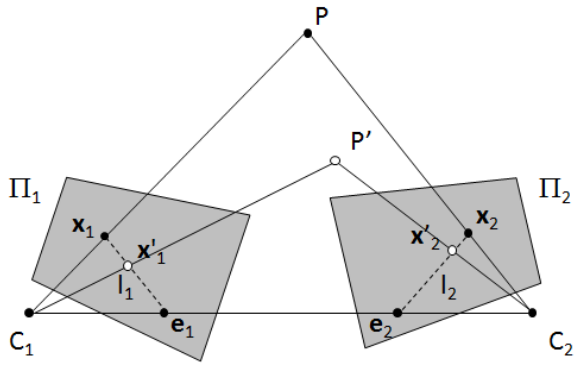


Figure 5.2: The image of any point within the same epipolar plane passes through the epipolar line. Here, the points P and P' lie in the same epipolar plane. Their respective images in the first image plane, $\mathbf{x}_1$ and $\mathbf{x}'_1$, both lie on the epipolar line $\mathbf{l}_1$. The same occurs on the second image plane.

the second image, we need only to look along the $\mathbf{l}_2$ in the second image, rather than scan the entire image.

The Epipoles

Now, the epipolar lines are not the same for every 3D point in space; For each point defining a distinct epipolar plane there is a distinct pair of epipolar lines. As we shall see, there is a simple relationship among the epipolar lines.

Refering back to Figure 5.1, consider the line joining the camera centers. The intersection of this line with each ideal image plane give the points $\mathbf{e}_1$ and $\mathbf{e}_2$. We call these points the *epipoles*. The epipole $\mathbf{e}_1$ is essentially the image of the second camera in the first image. Likewise, $\mathbf{e}_2$ is the image of the first camera in the second image. Note that the epipoles need not be on the physical image sensor. However, since the ideal plane extends to infinity, an epipole is guaranteed to exist *unless* the line joining the camera centers is *parallel* to the image plane. In that case, we express the non-existence of the epipole by saying it is located *at infinity*. If the epipoles *do* exist, they, like the images of P, lie on the epipolar line.

To find out why epipoles are worth considering, we now ask what happens when we move the object arbitrarily *off* the current epipolar plane. Figure 5.3 shows an object position P' which is not on the epipolar plane defined by P. The epipolar plane defined by the camera centers and P' gives distinct epipolar lines $\mathbf{l}'_1$ and $\mathbf{l}'_2$ and images $\mathbf{x}'_1$ and $\mathbf{x}'_2$. However, since the relative position of the cameras has not changed (*i.e.*, the camera centers and image planes

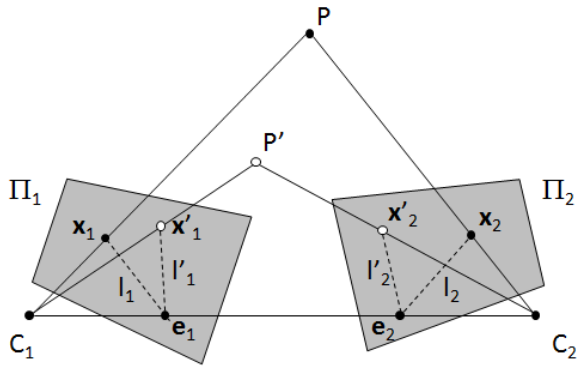


Figure 5.3: An object at P is moved to P', creating a new epipolar plane with corresponding epipolar lines $\mathbf{l}'_1$ and $\mathbf{l}'_2$. Because the relative positioning of the cameras is unchanged, the epipoles remaine the same. It follows that all epipolar lines pass through the epipoles.

have not moved), the epipoles remain the same. So, for each image, both epipolar lines pass through the epipoles. In fact, since the choice P' is arbitrary, *all epipolar lines pass through the epipoles*. So the epipoles are the convergence points for all epipolar lines on the respective images.

What, then, happens to the epipolar lines when an epipole is at infinity? Consider the epipole $\mathbf{e}_1$ and the pair of epipolar lines $\mathbf{l}_1$ and $\mathbf{l}'_1$ in Figure 5.3. Imagine we can grab $\mathbf{e}_1$ and slide it to right, towards C_2. Picture the $\mathbf{l}_1$ and $\mathbf{l}'_1$, always anchored to $\mathbf{x}_1$ and $\mathbf{x}'_1$, stretching to follow $\mathbf{e}_1$ as we slide it to the right. Notice that the further we slide $\mathbf{e}_1$, the closer the epipolar lines become to being parallel. It's easy to imagine, then, that in the limit that the epipole goes to infinity, the epipolar lines are *exactly* parellel.

Summary of the Epipolar Identities

Let us summarize the epipolar identities we've discovered.

Definition 5.1 (Epipolar Identities).

1. The *epipolar plane* is the plane formed by the two camera centers and a given point in space.

2. The *epipolar lines* are the intersections of the epipolar plane with the ideal image planes of each camera. All points in space within the epipolar plane produce images that lie on the epipolar lines.

3. The *epipoles* are the intersections between the line joining the two camera centers and the ideal image planes. For any point in space, the epipolar line on each image passes through the respective epipole.

The rules enfored by the epipolar identities are known as the *epipolar constraint*. So far we have presented the qualitative expression of this constraint. Now let's add an algebraic foundation to these geometric notions.

5.1.2 The Epipolar Constraint

The epipolar identities, when considered altogether, say something interesting and practical about how two images of the same object should relate. That is, they make useful predictions as to how two images are related, in terms of the motion between the two cameras. This insight is formally known as the *epipolar constraint*. In this section we present three forms of the epipolar constraint; First, we present a geometric, qualitative form of the epipolar constraint that follows directly from the preceding discussion. Second, we present a semi-qualitative form that uses camera coordinates which we will establish below. The purpose of the semi-qualitative form is to smooth the transition to the formal, algebraic form, which is the most concicse and complete of all three forms of the epipolar constraint.

Qualitative Form of the Epipolar Constraint

In the preceding section we've defined the epipolar plane, epipolar lines and the epipoles, all of which are intimately related and defined by relative position of the cameras and the observed point object. By surveying the properties of these identities, we can extract two basic rules that must be obeyed when imaging an object from two views.

Proposition 5.1 (Qualitative Epipolar Constraint). For two cameras viewing a point object, and considering the identities in Definition 5.1, we have the following

1. *The object's image in camera 1 ($\mathbf{x}_1$) defines the epipolar line for camera 2 ($\mathbf{l}_2$) on which we can expect to find the object's image in camera 2 ($\mathbf{x}_2$). (And vice versa.)*

2. *All epipolar lines pass through the epipoles, provided they exist. If they do not exist, the epipolar lines are parallel.*

This presentation of the epipolar constraint is the most pictoral and so the most intuitive. However, it is not the *whole* picture. Indeed, we have not adequately dealt with the case of non-existant epipoles. Note that qualitative epipolar constraint deals only with *projections* onto the

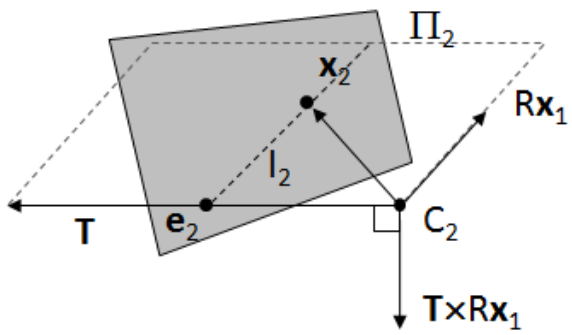


Figure 5.4: The vectors $\mathbf{x}_2$, $\mathrm{R}\mathbf{x}_1$ and $\mathbf{T}$ are coplanar. This common plane is the epipolar plane.

image planes. In essence, we have cast a 3D problem onto a 2D surface for the intuitive picture it creates. In the process of doing so, we've thrown away some of the details, and created singularities requiring special attention, such as the epipoles not existing. The solution to this, of course, is to treat the problem as it really is; A 3D one.

Semi-qualitative Form of the Epipolar Constraint

We now formulate the epipolar constraint semi-qualitatively, as a transition to the formal algebraic form. This requires system coordinates and an appropriate model for the motion between the two cameras.

Referring to Figure 4.1, let the rigid motion between two cameras be $\{\mathrm{R}, \mathbf{T}\}$. Then, a point P with coordinates $\mathbf{X}_1$ in the first camera frame has coordinates $\mathbf{X}_2 = \mathrm{R}\mathbf{X}_1 + \mathbf{T}$ in the second camera frame. From the projective relation $Z\mathbf{x} = \mathbf{X}$, we have

$$Z_2\mathbf{x}_2 = \mathrm{R}Z_1\mathbf{x}_1 + \mathbf{T} \tag{5.1}$$

Let us make two simple, but important observations. The first observation is that, from (5.1), the vectors $\mathbf{x}_2$, $\mathrm{R}\mathbf{x}_1$ and $\mathbf{T}$ are coplanar. This must be true because any two non-zero vectors $\mathbf{a}$ and $\mathbf{b}$ are coplanar, and their sum $\mathbf{c} = \mathbf{a} + \mathbf{b}$ is also within the same plane. The coplanar vectors $\mathbf{x}_2$, $\mathrm{R}\mathbf{x}_1$ and $\mathbf{T}$ are shown in Figure 5.4. Note that $\mathrm{R}\mathbf{x}_1$ is simply $\mathbf{x}_1$ (from Figure 5.1) with its tail placed at C_2, and applying R to it expresses it in coordinates of the second camera. The second observation is that, by definition, the vectors $\mathbf{T}$ and $\mathbf{x}_2$ are in the epipolar plane. We combine these two observations in the following proposition.

Proposition 5.2 (Semi-qualitative Epipolar Constraint). For two images $\mathbf{x}_1$ and $\mathbf{x}_2$ of the same point P from two cameras with relative rigid motion $\{\mathrm{R}, \mathbf{T}]\}$, the vectors $\mathbf{x}_2$, $\mathrm{R}\mathbf{x}_1$ and $\mathbf{T}$ all lie within the epipolar plane.

This is arguably a simpler statement than the qualitative form, but the mental picture is less clear. Let us now show that the two statements of Proposition 5.1 can be gotten from the single statement of Proposition 5.2.

We begin with the first statement. From Proposition 5.1, the normal to the epipolar plane is

$$\mathbf{N} = [N_x, N_y, N_z]^T = \mathbf{T} \times \mathrm{R}\mathbf{x}_1,$$

as shown in Figure 5.4. Now, the epipolar line on the second image $\mathbf{l}_2$ is the set of points $[x, y, 1]^T$ on the image plane that lie within the epipolar plane. In other words, the vectors $[x, y, 1]^T$ are perpendicular to the normal $\mathbf{N}$:

$$[x, y, 1]^T \cdot \mathbf{N} = xN_x + yN_y + N_z = 0$$

The above expression gives us the line equation of $\mathbf{l}_2$. Since $\mathbf{x}_2$ is coplanar to $\mathbf{T}$ and $\mathrm{R}\mathbf{x}_1$, it must also be that

$$\mathbf{x}_2 \cdot \mathbf{N} = x_2N_x + y_2N_y + N_z = 0$$

and so $\mathbf{x}_2$ lies on the epipolar line. We have shown that Proposition 5.2 implies that the first image ($\mathbf{x}_1$) defines a line on the second image plane ($\mathbf{l}_2$) on which we can expect to find the second image ($\mathbf{x}_2$). That is, Proposition 5.2 implies the first statement of Proposition 5.1.

For the second statement, consider the epipole $\mathbf{e}_2 = \frac{1}{T_z}\mathbf{T}$, where $\mathbf{T} = [T_x, T_y, T_z]^T$, shown in Figure 5.1. Note that for $\mathbf{e}_2$ to exist, we must have $T_z \neq 0$. Assuming $\mathbf{e}_2$ *does* exist, it is a scalar multiple of $\mathbf{T}$ so regardless of $\mathbf{x}_1$, $\mathbf{e}_2$ is perpendicular to $\mathbf{N}$:

$$\mathbf{e}_2 \cdot \mathbf{N} = \mathbf{e}_2 \cdot (\mathbf{T} \times \mathrm{R}\mathbf{x}_1) = 0$$

And since $\mathbf{x}_1$ determines the epipolar line on the second image, we have that all epipolar lines pass through the epipole $\mathbf{e}_2$, provided it exists. That's the first part of the second statement of Proposition 5.1.

Now consider the case where $\mathbf{e}_2$ *doesn't* exist, that is, $T_z = 0$. For two images $\mathbf{x}_1$ and $\mathbf{x}'_1$ resulting in two vectors $\mathbf{a} \equiv \mathrm{R}\mathbf{x}_1$ and $\mathbf{a}' \equiv \mathrm{R}\mathbf{x}'_1$, the normal to the epipolar plane is given by

$$\mathbf{N} = \mathbf{T}\times\mathbf{a} = [T_y a_z, -T_x a_z, T_x a_y - T_y a_x]^T \quad \text{and} \quad \mathbf{N}' = \mathbf{T}\times\mathbf{a}' = [T_y a'_z, -T_x a'_z, T_x a'_y - T_y a'_x]^T,$$

respectively. The epipolar lines from each are given by

$$xN_x + yN_y + N_z = 0 \quad \text{and} \quad xN'_x + yN'_y + N'_z = 0$$

First, note that we cannot have $a_z = 0$ since that would mean N_x and N_y are both zero and the line equation has no meaning. So, if $N_y = 0$ (*i.e.*, the first line is vertical), then it must

be that $T_x = 0$, which means $N'_y = 0$ as well. So if one epipolar line is vertical, then they are all vertical. Conversely, the more general case $N_y \neq 0$, implies $N'_y \neq 0$ as well, so the slope of each line is given by N_x/N_y and N'_x/N'_y. If these lines are parallel we should find that $N_x/N_y = N'_x/N'_y$. It turns out this is precisely the case:

$$\frac{N_x}{N_y} = \frac{T_y}{-T_x} = \frac{N'_x}{N'_y}.$$

Thus we have shown that if epipole doesn't exist, the epipolar lines are parrallel. And so we have shown that Proposition 5.2 implies the second statement of Proposition 5.1.

In the above analysis, we've shown that the coplanarity of the vectors $\mathbf{x}_2$, $\mathrm{R}\mathbf{x}_1$ and $\mathbf{T}$ (Proposition 5.2) leads the qualitative form of the epipolar constraint (Proposition 5.1). Having done this, relating the algebraic form of the epipolar constraint to the qualitative form will be rather straightforward.

Algebraic Form of the Epipolar Constraint

We now derive formal expression of the epipolar constraint. Our tactic is to manipulate equation (5.1) until we have an expression independent of the unkown depths Z_1 and Z_2. First, we cross $\mathbf{T}$ with both sides of equation (5.1) to obtain:

$$Z_2(\mathbf{T} \times \mathbf{x}_2) = Z_1(\mathbf{T} \times \mathrm{R}\mathbf{x}_1)$$

Next, we take the dot product of the above equation with $\mathbf{x}_2$:

$$0 = Z_1(\mathbf{x}_2 \cdot (\mathbf{T} \times \mathrm{R}\mathbf{x}_1))$$

Here $\mathbf{x}_2 \cdot (\mathbf{T} \times \mathbf{x}_2) = 0$ because $\mathbf{x}_2$ is perpendicular to $\mathbf{T} \times \mathbf{x}_2$. Finally, since Z_1 must be positive to physically realize an image we have the following result:

Theorem 5.1 (Epipolar Constraint). Two images $\mathbf{x}_1$ and $\mathbf{x}_2$ of the same point P from two cameras with relative rigid motion $\{\mathrm{R}, \mathbf{T}\}$ satisfy

$$\mathbf{x}_2 \cdot (\mathbf{T} \times \mathrm{R}\mathbf{x}_1) = 0 \tag{5.2}$$

This result states that $\mathbf{x}_2$ is perpendicular to $\mathbf{T} \times \mathrm{R}\mathbf{x}_1$, which means that $\mathbf{x}_2$ is in the plane spanned by $\mathbf{T}$ and $\mathrm{R}\mathbf{x}_1$. In other words, $\mathbf{x}_2$, $\mathbf{T}$ and $\mathrm{R}\mathbf{x}_1$ are coplanar, just as we stated in Proposition 5.2. So the geometric interpretation of (5.2) is *the same* as that of Proposition 5.2. That is, the geometric interpretation of Theorem 5.1 is indeed the qualitative constraints of Proposition 5.1.

In this section we have presented three forms of the epipolar constraint; One purely geometrical and qualitative (Proposition 5.1), one semi-qualitative (Proposition 5.2) and one algebraic (Theorem 5.1). In that order, each form is progressively more concise and complete, but becomes progressively more obscure geometrically. All three forms were presented to give a complete description of the epipolar constraint and its geometric consequences.

5.2 Triangulation

Triangulation, as applied to 3D vision, refers to depth recovery of 3D scene from two images. In this section we'll see that if two images are distinct views, and the motion between the two views is known, it is possible to infer the 3D location of points observed in both images.

5.2.1 Algebraic Formulation

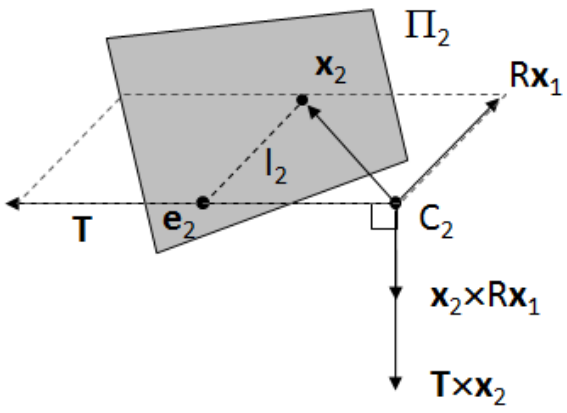


Figure 5.5: Triangulation from two image points. The vectors $\mathbf{x}_2$, $\mathrm{R}\mathbf{x}_1$ and $\mathbf{T}$ are coplanar. The depth of the point P in the first camera frame relates the magnitudes of the vectors $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{T} \times \mathbf{x}_2$, both normal to the epipolar plane.

As in the preceding section, we consider stereoscopic situation shown in Figure 5.1. The relation between the observed images in each camera is

$$Z_2\mathbf{x}_2 = \mathrm{R}Z_1\mathbf{x}_1 + \mathbf{T}$$

Since the depth of the point in one camera frame defines the depth in the other, we need only solve for one of them in the above expression. Z_2 is easily gotten rid of by crossing $\mathbf{x}_2$ with

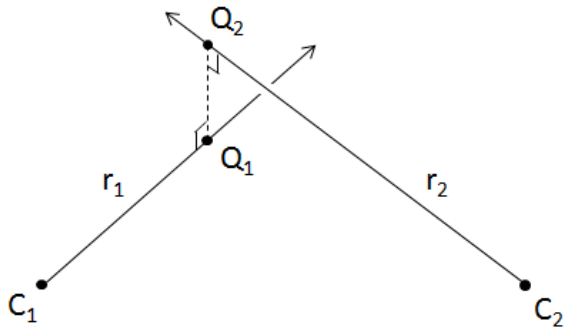


Figure 5.6: The rays r_1 and r_2 projected by each camera, in the presence of noise, may not intersect. The shortest distance between the two rays is along the line perpendicular to both rays (dashed line) connecting Q_1 and Q_2.

both sides, giving

$$Z_1(\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1) = \mathbf{T} \times \mathbf{x}_2 \tag{5.3}$$

So the vector $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ is a scalar multiple of $\mathbf{T} \times \mathbf{x}_2$, and that scalar is the depth Z_1. As shown in Figure 5.5, $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{T} \times \mathbf{x}_2$ are both normal to the epipolar plane.

(5.3) has the form $Z_1\mathbf{a} = \mathbf{b}$, where $\mathbf{a} \equiv \mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{b} \equiv \mathbf{T} \times \mathbf{x}_2$, and so Z_1 is found by taking the dot product of both sides with $\mathbf{a}$:

$$\hat{Z}_1 = \frac{(\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1) \cdot (\mathbf{T} \times \mathbf{x}_2)}{\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\|^2} \tag{5.4}$$

$\hat{Z}_1$ is the recovered object depth wrt. the first camera, and so the recovered object position w.r.t. the first camera is .

$$\hat{\mathbf{X}}_1 = \hat{Z}_1\mathbf{x}_1 \tag{5.5}$$

5.2.2 Geometric Interpretation

When the epipolar constraint is met, the vectors $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{T} \times \mathbf{x}_2$ (in (5.4) and shown in Figure 5.5) are aligned with each other and perpendicular to the epipolar plane. In that case, the recovered position (5.5) is unambiguously the true coordinates of the object w.r.t. the first camera. However, if there is uncertainty in the images or the motion between the cameras, then the epipolar constraint may not be met and $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{T} \times \mathbf{x}_2$ are *not* aligned. Nevertheless, we can *still* compute a recovered position. The question is; What is the *geometric meaning* of the recovered position (5.5) when the epipolar constraint is not met?

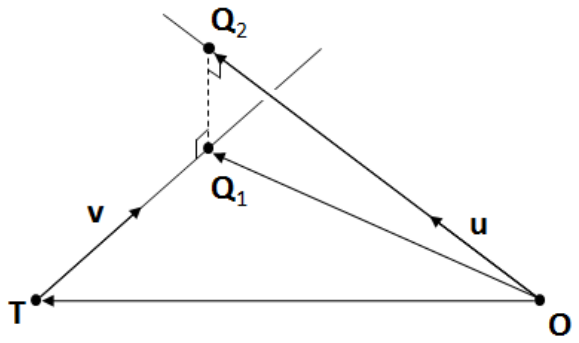


Figure 5.7: Non-intersecting rays in coordinates of the second camera, where $\mathbf{v} \equiv \mathrm{R}\mathbf{x}_1$ and $\mathbf{u} = \mathbf{x}_2$. The ray from each camera is given by $s_1\mathbf{v} + \mathbf{T}$ and $s_2\mathbf{u}$, where s_1 and s_2 are scalars.

Consider Figure 5.6. The image of the object in each camera defines a ray (r_1 and r_2) that is projected out into the world. As far as each camera is concerned, the object may exist anywhere on its respective ray. For ideal, noise-free cameras, the recovered object position is unambiguously the intersection point between the two rays. However, in the presence of noise these two rays may not intersect. Nevertheless, there exists a point on each ray (Q_1 and Q_2) that is closest to the other ray. We will show that the recovered position $\hat{\mathbf{X}}_1$ is precisely the point Q_1.

Consider now Figure 5.7, which is a representation of Figure 5.6 in coordinates of the second camera. For convenience, we denote $\mathbf{v} \equiv \mathrm{R}\mathbf{x}_1$ and $\mathbf{u} = \mathbf{x}_2$. $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are on the ray traced by the first and second camera, respectively. So

$$\mathbf{Q}_1 = s_1\mathbf{v} + \mathbf{T} \quad \text{and} \quad \mathbf{Q}_2 = s_2\mathbf{u}$$

where s_1 and s_2 are scalars.

Now, $\mathbf{Q}_1$ are the coordinates of Q_1 (from Figure 5.6) in the second camera frame. Since $\mathbf{Q}_1 = \mathrm{R}(s_1\mathbf{x}_1) + \mathbf{T}$, we see by inspection that its coordinates in the first camera frame are $s_1\mathbf{x}_1$. So in order to show that $\hat{\mathbf{X}}_1$ is the point Q_1 we must show that $s_1 = \hat{Z}_1$.

Let's begin with finding an expression for s_1. We can express the relationship between the two scalars s_1 and s_2 by recognizing that $\mathbf{Q}_2$ is the projection of $\mathbf{Q}_1$ onto $\mathbf{u}$:

$$\begin{aligned} \mathbf{Q}_2 &= \tfrac{\mathbf{Q}_1 \cdot \mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} \\ s_2\mathbf{u} &= \tfrac{(s_1\mathbf{v}+\mathbf{T}) \cdot \mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} \end{aligned} \qquad \Rightarrow \qquad s_2 = \frac{(s_1\mathbf{v} + \mathbf{T}) \cdot \mathbf{u}}{\|\mathbf{u}\|^2}$$

Now, let $\mathbf{d}$ be the difference between $\mathbf{Q}_2$ and $\mathbf{Q}_1$. For now, we'll not worry about any

specific properties of $\mathbf{d}$. We have

$$\begin{aligned}
\mathbf{Q}_2 - \mathbf{Q}_1 &= \mathbf{d} \\
s_2\mathbf{u} - (s_1\mathbf{v} + \mathbf{T}) &= \mathbf{d} \\
\left(\tfrac{(s_1\mathbf{v}+\mathbf{T})\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\right)\mathbf{u} - (s_1\mathbf{v} + \mathbf{T}) &= \mathbf{d} \\
s_1\left(\tfrac{\mathbf{v}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} - \mathbf{v}\right) &= \mathbf{d} + \mathbf{T} - \tfrac{\mathbf{T}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u}
\end{aligned}$$

Here we have, similarly to (5.3), an equation of the form $s_1\mathbf{a} = \mathbf{b}$, where $\mathbf{a} = \frac{\mathbf{v}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} - \mathbf{v}$ and $\mathbf{b} = \mathbf{d} + \mathbf{T} - \frac{\mathbf{T}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u}$. Again, the solution to s_1 is found by taking the dot product of both sides with $\mathbf{a}$:

$$s_1 = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|^2}$$

Let's evaluate each of these dot products. Evaluating $\mathbf{a}\cdot\mathbf{b}$ is greatly simplified by recognizing the following fact: Since $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are the points on each ray that is closest to the other ray, the line connecting them is perpendicular to both rays. So $\mathbf{d}$ is perpendicular to both $\mathbf{u}$ and $\mathbf{v}$, and so $\mathbf{d}\cdot\mathbf{u} = 0$ and $\mathbf{d}\cdot\mathbf{v} = 0$. With this insight, we end up with

$$\mathbf{a}\cdot\mathbf{b} = \mathbf{T}\cdot\left(\frac{\mathbf{v}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u} - \mathbf{v}\right) \quad \text{and} \quad \|\mathbf{a}\|^2 = \|\mathbf{v}\|^2 - \frac{(\mathbf{v}\cdot\mathbf{u})^2}{\|\mathbf{u}\|^2},$$

and our expression for s_1 is

$$s_1 = \frac{\mathbf{T}\cdot\left((\mathbf{v}\cdot\mathbf{u})\,\mathbf{u} - \|\mathbf{u}\|^2\,\mathbf{v}\right)}{\|\mathbf{v}\|^2\|\mathbf{u}\|^2 - (\mathbf{v}\cdot\mathbf{u})^2}. \tag{5.6}$$

Recall our goal is to show $\hat{Z}_1 = s_1$, so we must show that the right hand sides of (5.4) and (5.6) are equal. Our tactic is to manipulate (5.4) until we arrive at the same expression as in (5.6). To do this, we'll make use of two fundamental properties of the cross product: For any vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, we have

1. $\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c}) = \mathbf{b}\cdot(\mathbf{c}\times\mathbf{a})$ (Scalar Triple Product)
2. $\mathbf{a}\times(\mathbf{b}\times\mathbf{c}) = (\mathbf{a}\cdot\mathbf{c})\,\mathbf{b} - (\mathbf{a}\cdot\mathbf{b})\,\mathbf{c}$ (Vector Triple Product)

In the following derivation, employment of the Scalar Triple Product and Vector Triple Product is indicate by "STP" and 'VTP", respectively. Again using the simplifying notation $\mathbf{v} \equiv \mathrm{R}\mathbf{x}_1$ and $\mathbf{u} = \mathbf{x}_2$, the numerator of (5.4) is

$$\begin{aligned}
(\mathbf{u}\times\mathbf{v})\cdot(\mathbf{T}\times\mathbf{u}) &= \mathbf{T}\cdot(\mathbf{u}\times(\mathbf{u}\times\mathbf{v})) && \text{(STP)} \\
&= \mathbf{T}\cdot\left((\mathbf{v}\cdot\mathbf{u})\,\mathbf{u} - \|\mathbf{u}\|^2\,\mathbf{v}\right) && \text{(VTP)}
\end{aligned}$$

and the denominator of (5.4) is

$$\begin{aligned}(\mathbf{u}\times\mathbf{v})\cdot(\mathbf{u}\times\mathbf{v}) &= \mathbf{u}\cdot(\mathbf{v}\times(\mathbf{u}\times\mathbf{v})) && \text{(STP)}\\ &= \mathbf{u}\cdot\left(\|\mathbf{v}\|^2\,\mathbf{u}-(\mathbf{v}\cdot\mathbf{u})\,\mathbf{v}\right) && \text{(VTP)}\\ &= \|\mathbf{v}\|^2\|\mathbf{u}\|^2-(\mathbf{v}\cdot\mathbf{u})^2\end{aligned}$$

Finally, we have

$$\hat{Z}_1=\frac{\mathbf{T}\cdot\left((\mathbf{v}\cdot\mathbf{u})\,\mathbf{u}-\|\mathbf{u}\|^2\,\mathbf{v}\right)}{\|\mathbf{v}\|^2\|\mathbf{u}\|^2-(\mathbf{v}\cdot\mathbf{u})^2}=s_1$$

So indeed $\hat{Z}_1 = s_1$ and so we have shown that the recovered depth (5.4) gives the point Q_1 on the first ray r_1 which is closest to second ray r_2.

5.2.3 Theoretical Constraints

For a perfectly measured, noise-free system, the epipolar constraint is completely obeyed. Still, this does not mean we are guaranteed a unique, let alone a physical solution for $\hat{Z}_1$. Even in an ideal, error-free system we must check the existence, uniqueness and physicality of the depth inferred by equation (5.3).

Uniqueness of Depth

Using the notation $\mathbf{a} \equiv \mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{b} \equiv \mathbf{T} \times \mathbf{x}_2$ and referring to equation (5.4), let us examine the conditions for a unique solution for $\hat{Z}_1$. We have the following four cases:

1. If $\mathbf{a} = \mathbf{0}$ and $\mathbf{b} = \mathbf{0}$, then $\hat{Z}_1$ is not unique.
2. If $\mathbf{a} = \mathbf{0}$ and $\mathbf{b} \neq \mathbf{0}$, no solution exists.
3. If $\mathbf{a} \neq \mathbf{0}$ and $\mathbf{b} = \mathbf{0}$, then $\hat{Z}_1 = 0$.
4. If $\mathbf{a} \neq \mathbf{0}$ and $\mathbf{b} \neq \mathbf{0}$ then a solution exists provided $\mathbf{a}$ and $\mathbf{b}$ are colinear.

If we express the colinearity of $\mathbf{a}$ and $\mathbf{b}$ by their cross product, we have the following theorem.

Theorem 5.2 (Uniqueness of depth recovery). Given the images $\mathbf{x}_1$ and $\mathbf{x}_2$ of a point P taken by two cameras with relative rigid motion $\{\mathrm{R}, \mathbf{T}\}$, a unique depth $\hat{Z}_1$ of P w.r.t. the first camera exists if

$$\mathbf{a} \neq \mathbf{0} \quad \text{and} \quad \mathbf{a} \times \mathbf{b} = 0$$

where $\mathbf{a} \equiv \mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ and $\mathbf{b} \equiv \mathbf{T} \times \mathbf{x}_2$.

Physicality of Depth

We have given the necessary conditions for a unique solution, but what about the conditions for a *physical* one? An image is only physically realizable if the object is in front of the camera. In other words, the depth of the object must be positive. For our two-view system the depth w.r.t. *both* cameras must be positive.

Enforcing $\hat{Z}_1 > 0$ eliminates the possibility $\mathbf{a} \neq \mathbf{0}$ and $\mathbf{b} = \mathbf{0}$ (case 3 above). It also refines our statement about the colinearity of $\mathbf{a}$ and $\mathbf{b}$; For $\hat{Z}_1$ to be positive $\mathbf{a}$ and $\mathbf{b}$ must be *aligned*, rather than just colinear. This can be stated formally in terms of the dot product; Two vectors $\mathbf{a}$ and $\mathbf{b}$ are aligned if $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\|$.

Now, the recovered position w.r.t. the second camera is $\hat{\mathbf{X}}_2 = \mathrm{R}\left(\hat{Z}_1 \mathbf{x}_1\right) + \mathbf{T}$. Its z-component, given by $\hat{Z}_1 \left(\mathbf{r}_3 \cdot \mathbf{x}_1\right) + t_z$, is the depth of the object w.r.t. the second camera. So enforcing $\hat{Z}_2 > 0$ implies $\hat{Z}_1 \left(\mathbf{r}_3 \cdot \mathbf{x}_1\right) + t_z > 0$.

The criteria developed in the above discussion is summarized in the following theorem.

Theorem 5.3 (Physical depth recovery). Given the images $\mathbf{x}_1$ and $\mathbf{x}_2$ of a point P taken by two cameras with relative rigid motion $\{\mathrm{R}, \mathbf{T}\}$, then the positive recovered depths $\hat{Z}_1$ and $\hat{Z}_2$ of P w.r.t. the each camera exist if

1. $\mathbf{a} \neq \mathbf{0}$
2. $\mathbf{b} \neq \mathbf{0}$
3. $\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = 1$
4. $\hat{Z}_1 \left(\mathbf{r}_3 \cdot \mathbf{x}_1\right) + t_z > 0$

where $\mathbf{a} \equiv \mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$, $\mathbf{b} \equiv \mathbf{T} \times \mathbf{x}_2$, $\mathbf{r}_3$ is the third row of R, t_z is the z-component of $\mathbf{T}$ and $\hat{Z}_1$ is given by (5.4).

Interestingly, these theoretical constraints say something about the physical requirements for triangulation. Consider the first constraint $\mathbf{T} \times \mathbf{x}_2 \neq \mathbf{0}$.

First, it says neither $\mathbf{T}$ nor $\mathbf{x}_2$ can be zero. We already know this fact about $\mathbf{x}_2$, since it is on the second image plane (z-component is one). So *we must have at least some displacement ($\mathbf{T} \neq \mathbf{0}$) for triangulation*.

Second, it says $\mathbf{T}$ and $\mathbf{x}_2$ can't be colinear, which is equivalent to saying the image $\mathbf{x}_2$ is not equal to the epipole $\mathbf{e}_2$. This can only happen if the point P responsible for the image is on the same line that joins the camera centers. So the first constraint says that (1) the two camera views must be distinct and (2) the point P must not lie on the line joining the camera centers.

Now consider the second constraint $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1 \neq \mathbf{0}$. Since neither $\mathbf{x}_1$ nor $\mathbf{x}_2$ can be zero, this statements simply says $\mathbf{x}_2$ and $\mathrm{R}\mathbf{x}_1$ are not colinear. Well, when would such a situation arise? If the point P was very far from the first camera, then the separation between the two cameras would be negligible and so (5.1) becomes $Z_2\mathbf{x}_2 = Z_1\mathrm{R}\mathbf{x}_1$ (in other words, they are colinear). So the second constraint says the point P must be at a finite distance from the cameras.

5.2.4 Practical Constraints

In the preceding section we have established the constraints for a unique and positive depth recovery from two images of a point in an ideal, error-free system. However, real cameras and computer vision systems are not error-free. As such, we ought to relax the conditions (1)-(3) in Theorem 5.3 by employing thresholds, such as

1. $\|\mathbf{a}\| > \epsilon_1$
2. $\|\mathbf{b}\| > \epsilon_2$
3. $\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{a}\|} > \epsilon_3$

where $\mathbf{a}$ and $\mathbf{b}$ are defined as in Theorem 5.3.

The thresholds we set must be based on the uncertainty in the measurements of our system. Our measurements take place on the sensor of each camera, and a measurement, to be exact, is a *location* on the sensor. An uncertainty in the measurement, therefore, is a distance on the sensor. So our thresholds must be related to distances on the camera sensor.

We assume the cameras are calibrated, and that the image sensor is related to the ideal image plane by (*not written yet*), so we can relate distances on the image sensor to distances on the ideal image plane. The question is: Are the values $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ distances on the image plane? It turns out they are not, but are closely related to meaningful, observable distances. In this section we will discuss these "observable distances" and modify the conditions (1) and (2) in terms of them.

Relation to Disparity

We begin by developing physical interpretation of the vector $\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$ in equation (5.3). Consider two cameras whose axes are aligned and whose relative displacement is parallel to the image planes. This setup, shown in Figure 5.8, is known as a *calibrated stereo rig*. In a calibrated configuration, the depth Z is the same for both cameras. Due to similar triangles, we

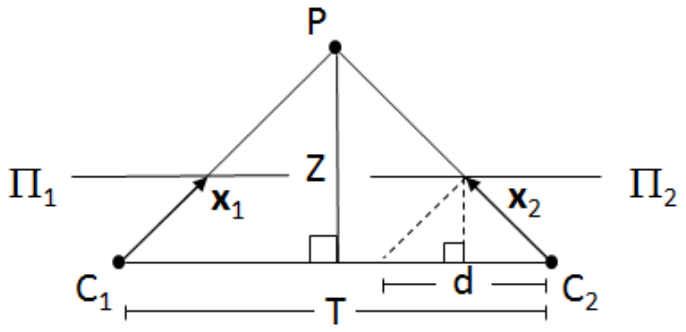


Figure 5.8: Triangulation from two image points in a calibrated stereo rig. The image planes are aligned and parallel to the translation between the two cameras.

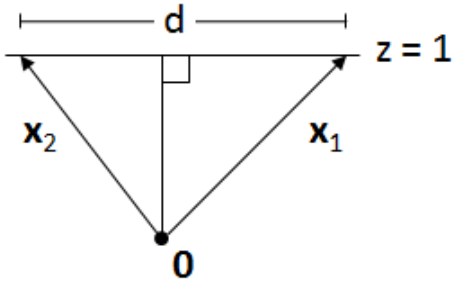


Figure 5.9: Two image vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ have a disparity $d = \|\mathbf{x}_1 \times \mathbf{x}_1\|$.

have

$$Z = \frac{T}{d}$$

where $d = \|\mathbf{x}_2 - \mathbf{x}_1\|$, known as the *disparity*, is the distance between the first and second image. The disparity is a very relevant physical quantity because it is a distance on the image plane, and so relates directly to units of the image sensor. In other words, the disparity, after transformed to pixel coordinates, is the number of pixels between the first and second images.

Now consider the two image vectors, as shown in Figure 5.9. The image vectors and the $z = 1$ plane form a triangle. Say the area of this triangle is A. Then, since this triangle has a base of length d, and a height of 1, we have $d = 2A$. Now, $\|\mathbf{x}_2 \times \mathbf{x}_1\|$ is the area of the parallelogram with sides $\|\mathbf{x}_1\|$ and $\|\mathbf{x}_2\|$, which is twice the area of the triangle in Figure 5.9. So $\|\mathbf{x}_2 \times \mathbf{x}_1\| = 2A$ as well. This gives us the following interesting result.

Theorem 5.4 (Disparity Relation). If images $\mathbf{x}_1$ and $\mathbf{x}_2$ of a point P are taken by two

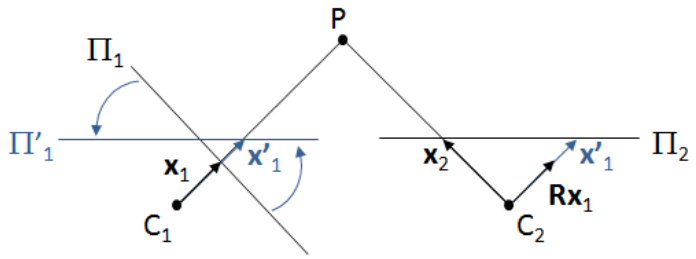


Figure 5.10: Disparity in the presence of rotation. Due to the rotation between the cameras, the vector $\mathrm{R}\mathbf{x}_1$ is not on the image plane Π_2. If we were to rotate the first camera such that its axes align with those of the second camera ($\Pi_1 \rightarrow \Pi_1'$), the new image $\mathbf{x}_1'$ would be on the image plane.

cameras with aligned axes, then their disparity d is the magnitude of their cross product:

$$d = \|\mathbf{x}_2 - \mathbf{x}_1\| = \|\mathbf{x}_2 \times \mathbf{x}_1\|$$

This means that when there is no rotation between the cameras the value $\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\| \rightarrow \|\mathbf{x}_2 \times \mathbf{x}_1\|$ is, in fact, the disparity.

Now, what happens to the disparity when there is a rotation between the two cameras? Let's consider the uncalibrated situation shown in Figure 5.10. Because of the rotation, $\mathrm{R}\mathbf{x}_1$ is not necessarily on the second image plane. To express the disparity, then, we would have to go realign the camera axes. That is, we would have to undo the rotation. Doing so would produce a new image plane Π_1' and, correspondingly, a new image $\mathbf{x}_1'$. Since we're back in the calibrated case, $\mathbf{x}_1'$ is on the second image plane and we can once again define disparity. So the distance $\|\mathbf{x}_2 - \mathbf{x}_1'\|$, which we'll call the *calibrated disparity*, is the disparity we would have observed if the cameras axes were aligned. Now, notice from Figure 5.10 that undoing the rotation in the first camera is equivalent to projecting $\mathrm{R}\mathbf{x}_1$ to the second image plane. So we have $\mathbf{x}_1' = \pi(\mathrm{R}\mathbf{x}_1)$. Let's put these notions together in the following definition.

Definition 5.2 (Calibrated Disparity). For two images $\mathbf{x}_1$ and $\mathbf{x}_2$ of a point P taken from two cameras with relative rotation R, the *calibrated disparity* d_R is the distance between the image $\mathbf{x}_2$ and the projection of $\mathrm{R}\mathbf{x}_1$ onto the second image plane:

$$d_\mathrm{R} \equiv \|\mathbf{x}_2 - \pi\left(\mathrm{R}\mathbf{x}_1\right)\|$$

Geometrically, the calibrated disparity is the disparity that would have been observed if the axes of the first camera were aligned with those of the second camera.

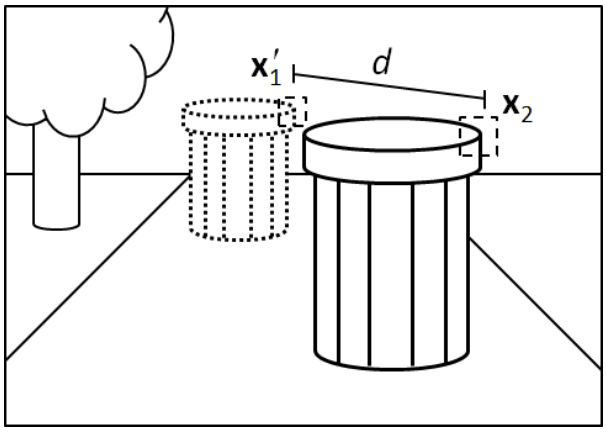


Figure 5.11: The calibrated disparity for two image points $\mathbf{x}_1$ and $\mathbf{x}_2$. An object (a garbage can) as seen from the second camera is drawn with solid lines. The same object as seen from the first camera is drawn with dotted lines[1]. The calibrated disparity d is the distance between the two image points $\mathbf{x}_1'$ and $\mathbf{x}_2$, where $\mathbf{x}_1' = \pi\left(\mathrm{R}\mathbf{x}_1\right)$.

If we let z' be the z-component of $\mathrm{R}\mathbf{x}_1$, by Theorem 5.4 the calibrated disparity can be expressed as

$$\begin{aligned} \|\mathbf{x}_2 - \pi\left(\mathrm{R}\mathbf{x}_1\right)\| &= \|\mathbf{x}_2 \times \pi\left(\mathrm{R}\mathbf{x}_1\right)\| \\ &= \left\|\mathbf{x}_2 \times \left(\tfrac{1}{z'}\mathrm{R}\mathbf{x}_1\right)\right\| \\ &= \tfrac{\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\|}{|z'|}, \end{aligned}$$

which proves the following.

Corollary 5.1 (Calibrated Disparity). For two images $\mathbf{x}_1$ and $\mathbf{x}_2$ of a point P taken from two cameras with relative rotation R, the calibrated disparity is given by

$$d_{\mathrm{R}} = \frac{\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\|}{|z'|}$$

where z' is the z-component of $\mathrm{R}\mathbf{x}_1$.

The value $\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\|$, as it turns out, is closely related to the calibrated disparity. And since the calibrated disparity is a distance on the image plane, we modify condition (1) as

$$\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\| > \epsilon_1 \qquad \rightarrow \qquad \frac{\|\mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1\|}{|z'|} > \epsilon_1 \qquad \Leftrightarrow \qquad \|\mathbf{x}_2 - \pi\left(\mathrm{R}\mathbf{x}_1\right)\| > \epsilon_1.$$

In other words, we require that the distance between the image points $\pi\left(\mathrm{R}\mathbf{x}_1\right)$ and $\mathbf{x}_2$ (*i.e.*, the calibrated disparity) be greater than the threshold ϵ_1. This is illustrated in Figure 5.11. Here we see the image of an object (a garbage can) as viewed from the second camera. The image of the

same object, as seen from the first camera, is superimposed on the same image and is shown with dotted lines [1]. Now let's consider the top-right corner of the object and its image $\mathbf{x}_1'$ in the first camera and $\mathbf{x}_2$ in the second camera. The distance between these two image points is d. The modified condition (1) says that the pair $(\mathbf{x}_1, \mathbf{x}_2)$ are triangulated only if $d > \epsilon_1$.

Relation to the Epipole

We use a similar line of reasoning for interpreting $\|\mathbf{T} \times \mathbf{x}_2\|$. Recall that the projection of $\mathbf{T}$ onto the second image plane is the second epipole $\mathbf{e}_2$. This can be seen in Figure 5.5. Notice that, by Theorem 5.4, the distance between $\mathbf{x}_2$ and $\mathbf{e}_2$ is

$$\begin{aligned} \|\mathbf{e}_2 - \mathbf{x}_2\| &= \|\mathbf{e}_2 \times \mathbf{x}_2\| \\ &= \left\| (\tfrac{1}{t_z}\mathbf{T}) \times \mathbf{x}_2 \right\| \\ &= \tfrac{\|\mathbf{T} \times \mathbf{x}_2\|}{|t_z|} \end{aligned}$$

So we have the following.

Corollary 5.2 (Distance to the epipole). For two cameras with relative translation $[t_x\ t_y\ t_z]^{\mathrm{T}}$ ($t_z \neq 0$) and an image $\mathbf{x}_2$ on the second image plane, the distance between $\mathbf{x}_2$ and the epipole $\mathbf{e}_2$ is given by

$$\|\mathbf{e}_2 - \mathbf{x}_2\| = \frac{\|\mathbf{T} \times \mathbf{x}_2\|}{|t_z|}$$

This is, of course, assuming $\mathbf{e}_2$ exists (or equivalently, $t_z \neq 0$). For our case of a rear view camera on a backing vehicle, the camera's optical (z) axis is in the general direction of the vehicle's velocity, so this assumption is valid. So we have $\|\mathbf{T} \times \mathbf{x}_2\| = |t_z|\,\|\mathbf{e}_2 - \mathbf{x}_2\|$, where $|t_z|$ And since $\frac{\|\mathbf{T} \times \mathbf{x}_2\|}{|t_z|}$ is a distance on the image plane, we modify condition (1) as

$$\|\mathbf{T} \times \mathbf{x}_2\| > \epsilon_2 \qquad \rightarrow \qquad \frac{\|\mathbf{T} \times \mathbf{x}_2\|}{|t_z|} > \epsilon_2 \qquad \Leftrightarrow \qquad \|\mathbf{e}_2 - \mathbf{x}_2\| > \epsilon_2.$$

In other words, we require that the distance between the image point $\mathbf{x}_2$ and the epipole $\mathbf{e}_2$ be greater than the threshold ϵ_2. This is illustrated in Figure 5.12. The epipole $\mathbf{e}_2$ is in the direction of the vehicle's translation, and ϵ_2 is the radius of the the shaded circle surrounding the epipole. Effectively, an image point pair $(\mathbf{x}_1, \mathbf{x}_2)$ are triangulated only if $\mathbf{x}_2$ is located outside the shaded circle.

[1] To be precise, the dotted image is the object seen from the first camera *if the axes of the first camera were aligned with those of the second camera*, as in Figure 5.10. This would result in a perspective-skewed version of the actual image observed by the first camera. However, for the sake of clarity, here we draw it as if there is no rotation between the two cameras, and so the dotted image is exactly the image seen by the first camera.

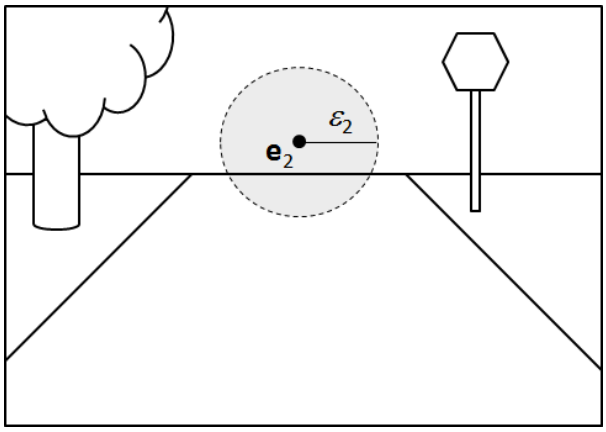


Figure 5.12: A feature must be at a certain distance from the epipole in order to obtain reliable triangulation. Only features outisde the shaded circle are triangulated.

Setting the Thresholds

The sensor is made up of discrete units (pixels), so the threshold for disparity ϵ_1 and the distance to epipole ϵ_2 should be greater than the width of a single pixel. In fact, it is convenient to express the thresholds in terms of an integral number of pixels. Say we choose the minimum number of pixels between any two image points to be n, and the lens has a focal length f (in units of pixels), then the minimum disparity on the ideal image plane is $d_{\text{min}} = n/f$. In our application, we set $d_{\text{min}} = 20/f$ for both conditions (1) and (2), so we have $\epsilon_1 = \epsilon_2 = d_{\text{min}}$.

For condition (3), the left side of the inequality is the cosine of the angle between $\mathbf{a}$ and $\mathbf{b}$. So for a maximum angle of θ_{max} one should set $\epsilon_3 = \cos\theta_{\text{max}}$. We use 10° as the maximum angle between these two vectors, so we have $\epsilon_3 = 0.985$.

Finally, we summarize the results of this section in the following.

Proposition 5.3 (Practical constraints for triangulation). Given the images $\mathbf{x}_1$ and $\mathbf{x}_2$ of a point P taken by two cameras with relative rigid motion $\{\mathrm{R}, \mathbf{T}\}$, a minimum disparity d_{min} and a maximum angle θ_{max}, then the positive recovered depths $\hat{Z}_1$ and $\hat{Z}_2$ of P w.r.t. the each camera can be reliably computed if

1. $\|\mathbf{a}\| > |z'|\, d_{\text{min}}$ (Calibrated Disparity)
2. $\|\mathbf{b}\| > |t_z|\, d_{\text{min}}$ (Distance to Epipole)
3. $\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} > \cos\theta_{\text{max}}$ (Vector Alignment)
4. $\hat{Z}_1\,(\mathbf{r}_3 \cdot \mathbf{x}_1) + t_z > 0$ (Positive depth $\hat{Z}_2$)

where $\mathbf{a} = \mathbf{x}_2 \times \mathrm{R}\mathbf{x}_1$;
$\mathbf{b} = \mathbf{T} \times \mathbf{x}_2$;
$\mathbf{r}_3 =$ third row of R;
$t_z = z$-component of $\mathbf{T}$;
$z' = z$-component of $\mathrm{R}\mathbf{x}_1$;

and $\hat{Z}_1$ is given by (5.4).

5.2.5 Optimal Triangulation

We acknowledge that the recovered depth given by (5.4) is not optimal in the sense that it does not minimize errors on the image planes. Optimal triangulation methods recover depth by minimizing the distance between the object's image (as seen from each camera) and the respective epipolar line). Such a minimization is non-linear and so iterative and computationally expensive. During the development of our algorithm we have found the gain in optimal triangulation negligible, and so use (5.4) instead.

5.2.6 Multiview Triangulation

The least squares solution for the depth of a point wrt. the first camera, given by (5.4), is easily extended to a least squares solution for multiple views. Say we have k images. From these images we can define $k-1$ independent inter-image motions. Let $\{\mathrm{R}_i, \mathbf{T}_i\}$ be the rigid motion between the image $i = 2 \ldots k$ and image 1. That is, all rigid motions are wrt. the first image. For a point object with corresponding points $\mathbf{x}_1,\ \mathbf{x}_2\ \ldots\ \mathbf{x}_{k+1}$ in each image, expression (5.3) applies to each pair of images:

$$Z_1(\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_1) = \mathbf{T}_i \times \mathbf{x}_i \qquad i = 2 \ldots k \tag{5.7}$$

If we denote $\mathbf{a}_i \equiv \mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_1$ and $\mathbf{b}_i \equiv \mathbf{T}_i \times \mathbf{x}_i$, then we have the system of $k-1$ equations $Z_1\mathbf{a}_i = \mathbf{b}_i$. If we stack all the $\mathbf{a}$ and $\mathbf{b}$ vectors to form the vectors $\mathbf{a}_s$ and $\mathbf{b}_s$, respectively, we get an equivalent single vector equation; $Z_1\mathbf{a}_s = \mathbf{b}_s$. The least squares solution to this expression is

$$Z_1 = \frac{\mathbf{a}_s \cdot \mathbf{b}_s}{\|\mathbf{a}_s\|}$$

which is equivalent to summing the numerator and denominator portions of the 2-view least squares solution (5.4)

$$Z_1 = \frac{\sum_{i=2}^{k} (\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_1) \cdot (\mathbf{T}_i \times \mathbf{x}_i)}{\sum_{i=2}^{k} \|\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_1\|} \tag{5.8}$$

Now, due to noise and tracking error, point pairs are not guaranteed to obey the epipolar constraint. So we must check that a given point pair fits the criteria in Proposition 5.3 before including it in the summation in (5.8). If none of the point pairs obey the epipolar constraint, then the depth remains undefined. This leads to a more robust multi-view triangulation.

We present our multi-view triangulation in the following algorithm.

Algorithm 5.1 (Multiview Triangulation).
For a set of rigid motions $\{\mathrm{R}_2, \mathbf{T}_2\} \ldots \{\mathrm{R}_m, \mathbf{T}_m\}$, where $\{\mathrm{R}_i, \mathbf{T}_i\}$ is the motion between image i and image 1, and a set of point correspondences $\mathbf{x}_1 \ldots \mathbf{x}_m$, this algorithm computes the depth of the point Z_1 wrt. the first image.

numerator := 0
denominator := 0
for $i = 2$ to m **do**
  if point pair $(\mathbf{x}_1, \mathbf{x}_i)$ fits the criteria in Proposition 5.3, wrt. $\{\mathrm{R}_i, \mathbf{T}_i\}$ **then**
    numerator += $(\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_1) \cdot (\mathbf{T}_i \times \mathbf{x}_i)$
    denominator += $\|\mathbf{x}_i \times \mathrm{R}_i\mathbf{x}_1\|$
  end if
end for
if denominator > 0 **then**
  Z_1 := numerator/denominator
else
  $Z_1 := \infty$
end if

Chapter 6

Obstacle Detection

In this chapter we describe the Obstacle Detection (OD) algorithm in detail. In the first section we present the main components and data flow of the algorithm. The following sections describe each component in detail.

6.1 Algorithm Overview

Our algorithm achieves obstacle detection through 3D reconstruction of the scene using a single camera. To do this we detect and track image features across multiple frames. From the tracked features we estimate the motion of the camera and triangulate the features to create a 3D model of the scene. Features located within a collision corridor behind the vehicle are labelled as obstacles and the system reports the nearest obstacle distance to the driver.

An overview of the algorithm is shown in Figure 6.1. The algorithm is composed of six modules; (1) Feature Tracking, (2) Snapshot Management, (3) Motion Estimation, (4) Feature Triangulation, (5) Feature Labelling and (6) Feature Location.

The *Feature Tracking* module has three main responsibilities; (1) track existing features, (2) delete invalid features and (3) create new features. It takes as input the current image (*Image (t)*), the previous image (*Image ($t-1$)*), and the previous list of features (*Features (t-1)*), and it outputs the current list of features (*Features (t)*).

The *Snapshot Management* module selects a "good" set of previous images to pair with the current image for 3D reconstruction. Here, "good" means that there is sufficient camera displacement between each image in the set. For an image taken at time t_s and belonging to this "good" set, we call t_s a *snapshot*. The input to the Snapshot Management module is the current list of tracked features and the output is a list of snapshots.

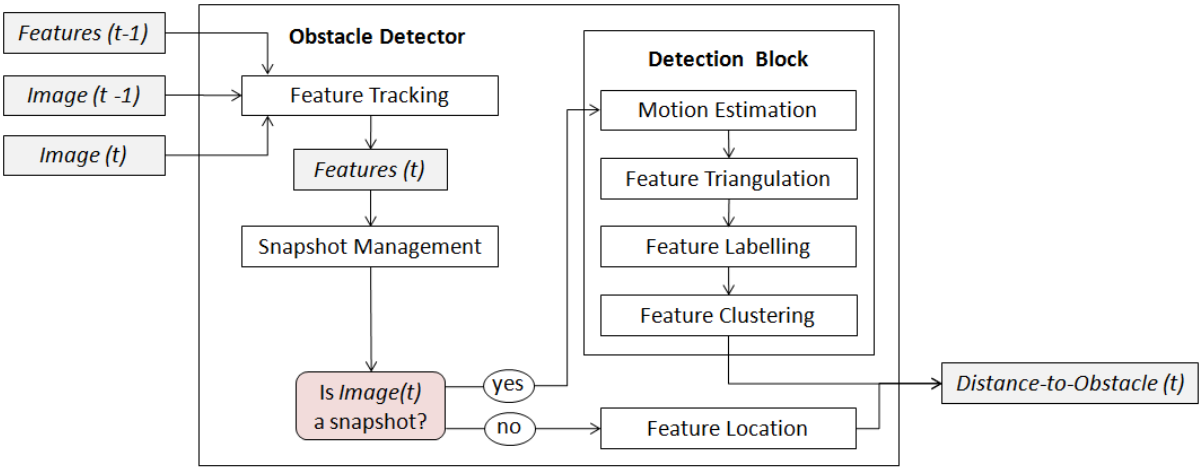


Figure 6.1: Overview of the obstacle detection algorithm. The current and previous images, and the previous list of features are fed to the *Feature Tracking* module to obtain the current list of features. The *Snapshot Management* module analyzes the motion of the features to determine what images (or *snapshots*) to use for 3D reconstruction. The *Motion Estimation* module estimates the planar motion of the camera between the current snapshot and all snapshots in the past. The *Feature Triangulation* module finds the 3D location of features, provided they obey the epipolar constraint. The *Feature Labelling* module assigns interpretive labels (such as "ground" or "obstacle") to each feature based their location w.r.t. the vehicle. The *Feature Clustering* module spatially clusters detected features to reduce false detections. The *Feature Location* module updates the location of triangulated points assuming a constant height. Finally, the distance to the nearest obstacle is reported to the driver.

If the current image is selected as a snapshot, the *detection block* is called. The detection block is where image features are triangulated and labelled based on their location in 3D space. If the current image is not a snapshot, then the 3D location of previously triangulated features are updated in the *Feature Location* module.

The *Motion Estimation* module estimates the planar motion between all snapshots and the current time. That is, the Motion Estimation module produces a set of motion parameters $\{\mathrm{R}_i, \mathbf{T}_i\}$ for each snapshot, where $\{\mathrm{R}_i, \mathbf{T}_i\}$ is the motion between the current time and the i^{th} snapshot.

The *Feature Triangulation* module performs triangulation on the features w.r.t. the list of motion parameters, provided they obey the epipolar constraint. For each feature, it is indicated whether or not the epipolar constraint is obeyed, and if it is, the 3D location of the feature is computed.

The *Feature Labelling* module assigns interpretive labels (such as "ground" or "obstacle") to each feature based their location w.r.t. the vehicle.

The *Feature Clustering* module spatially clusters detected features as a means of rejecting isolated false detections.

With all the features labelled and clustered, the OD algorithm outputs the distance to the nearest obstacle.

The *Feature Location* module updates the 3D location of all features that have been previously triangulated. It works on the assumption that the height (world z-component) of any feature is constant, and so each feature is projected to its known world z-plane.

The remainder of this chapter is organized as follows. Section 6.2 describes the Feature Tracking module. Section 6.3 describes feature selection for motion estimation and the core motion estimation algorithm. Section 6.4 describes the Snapshot Management module. Section 6.5 describes the Motion Estimation module. Section 6.6 describes the Feature Triangulation. Section 6.7 describes the Feature Labelling module. Section 6.8 describes the Feature Clustering module. Finally, Section 6.9 describes the Feature Labelling module.

6.2 The Tracking Module

The *Feature Tracking* module has three main responsibilities; (1) track existing features, (2) delete invalid features and (3) detect new features. It takes as input the current image (*Image (t)*), the previous image (*Image ($t-1$)*), and the previous list of features (*Features (t-1)*), and it outputs the current list of features (*Features (t)*).

To track existing features from one image to the next, we calculate the optical flow at

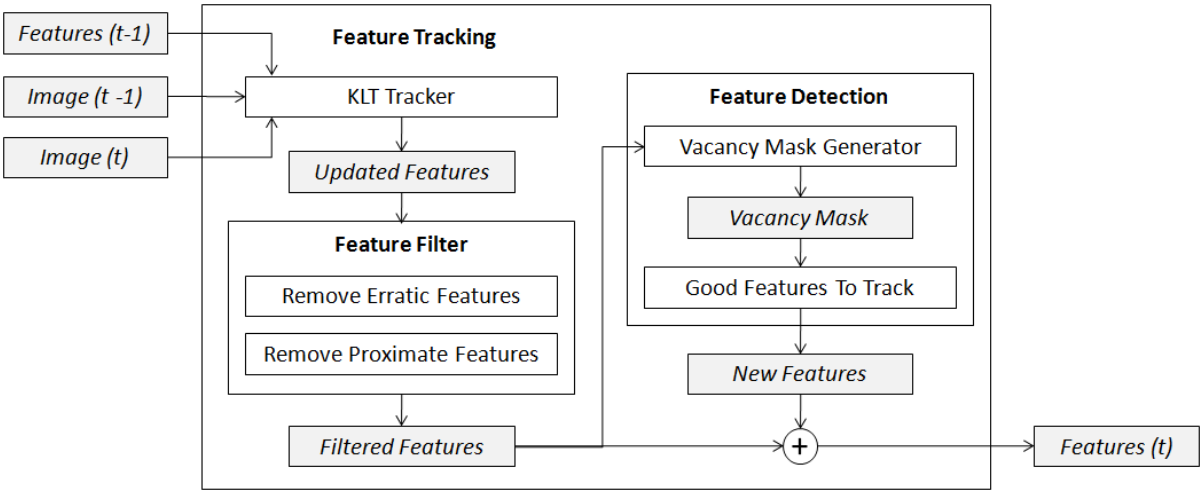


Figure 6.2: The Feature Tracking module consists of three processes; (1) the Lucas-Kanade (KLT) tracking algorithm, (2) feature filtering and (3) feature detection. Feature filtering consists of removing erratically moving features and features the become too close to one another. Corner features are detected using the *Good Features To Track* algorithm wherever features don't already exist.

the center of each feature with the Lucas-Kanade (KLT) algorithm [22]. We use a square feature template size of $s_T = 15$ pixels and 3 pyramid levels to cope with large motions. For each feature, the KLT algorithm sets a flag indicating whether or not the flow was able to be computed. If the flow was found, we update the feature's position according to the flow vector. If the flow was not found, we delete the feature from the list. The result is a list of features with updated positions, which is passed to the Feature Filter module.

The *Feature Filter* module deletes features based on two criteria; (1) features with erratic motion and (2) features in close proximity to other features. Erratically moving features are deleted because, since frame rate of the camera is high and the vehicle moves smoothly, we expect the image of the scene to move smoothly. So, any erratic motion in the image is due to either tracking error or some object moving about in the scene. In either case, the feature is unwanted for both ground motion estimation and 3D reconstruction.

We measure the non-smoothness of a feature's trajectory as a deviation from a linear trajectory over a recent window of time. Specifically, we least-square-fit the last w_δ pixel positions of a feature to a line $\mathbf{y}(t) = \mathbf{a}t + \mathbf{b}$. Then we compute the mean reprojection error of the trajectory

$$\bar{\delta} = \frac{1}{m} \sum_i \|\mathbf{y}(i) - \mathbf{x}_i\|.$$

We define a *smoothness of trajectory threshold* ϵ_δ and remove any feature with $\bar{\delta} > \epsilon_\delta$. We use the last $w_\delta = 5$ frames and set $\epsilon_\delta = 10$ pixels. The smoothness window of 5 frames corresponds to a very short-term trajectory (0.16s at 30fps). We expect the vehicle motion to be smooth wrt. this time interval, so vehicle rotations do not have a significant impact on trajectory smoothness. Instead, we filter out the high frequency erratic motion typically manifested by noisy tracking.

The remaining features are then filtered based on their proximity to other features. Feature templates can overlap when features are tracked along edges. In particular, features tend to accumulate on the edge of the vehicle bumper at the bottom of the image. To avoid tracking the same image region multiple times, it is important to remove proximate tracked features.

For each pair of features, if their distance is less than the threshold r_{min}, we eliminate the feature with the greater mean reprojection error $\bar{\delta}$ (described above). This way, features producing smooth trajectories are favoured. We set $r_{min} = s_T/2$, the half-width of the tracking template size (7 pixels).

The list of remaining features is passed to the *feature detection* module, which is responsible for generating new features to track. The first step in our feature detection, is the generation of a *vacancy mask*. The vacancy mask is a binary image indicating the regions of the image that are currently unnoccupied by any features. It is only these vacant regions that we search for features to track, since we do not want to create trackers where some already exist.

To generate the vacancy mask we start with a white binary image of the same size as the image on which tracking is performed. Then, for each tracked feature, the pixels within a square region of size s_v and centered at the feature's current location are set to black. To ensure that features are not initialized too close to each other, we set $s_v = 1.5 s_T$, one and a half times the tracking template size.

The vacancy mask is used as the mask for applying a *modified* Good Features to Track (GFTT) algorithm [30]. The modification is the following. In GFTT, a relative quality threshold (between 0 and 1) is used to decide whether or not a corner feature is "good enough". So, if the image is of uniform texture and the strongest feature is weak (in absolute terms), then the corners returned by GFTT are due to noise and so are meaningless. To avoid this situation, we use an *absolute* corner quality threshold of $q_c = 0.001$, rather than a relative one.

The new features detected by our modified GFTT are added to the list of currently tracked features to become the complete current feature list (features (t) in Figure 6.2).

6.3 Motion Estimation from Good Ground Features

Before describing the Snapshot Selection and Motion Estimation components (Figure 6.1), it will be convenient to first describe in detail our motion estimation procedure. In Section 4.6 we describe an algorithm for estimating planar motion from image features *assumed* to be moving on the plane. In this section we describe a feature selection method for increasing the likelihood that a selected feature belongs to the ground. Then we describe our motion estimation algorithm that makes use of these selected features, and that is used in both the Snapshot Management (Section 6.4) and Motion Estimation (Section 6.5) components.

6.3.1 Selecting Good Ground Features

We now address the issue of selecting good ground features for estimating the motion between two images. Let the image indices be t_1 and t_2.

First and foremost, a feature considered for ground motion estimation must exist in both images. That is, the feature has an ideal image position $\mathbf{x}_1$ and $\mathbf{x}_2$ for the images t_1 and t_2, respectively.

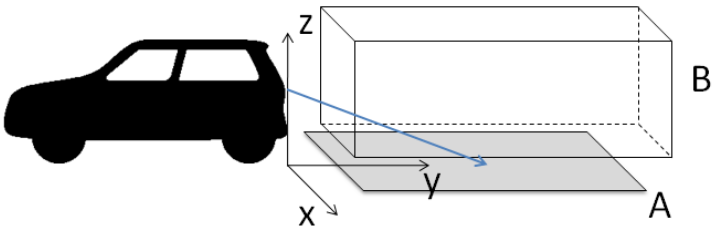


Figure 6.3: The regions of interest in the vehicle coordinate system. The ground region of interest (A) is the region in which ground features are used for motion estimation. Any feature located in the collision volume (B) is determined to be an obstacle. The blue arrow indicates the camera's optical axis.

Second, for a feature to possibly belong to the ground, it must be within the image region onto which the ground is projected. To assert this, we define a rectangular *ground region of interest* (ground ROI) in vehicle coordinates, shown in Figure 6.3. Only features that project to the ground ROI (via (4.12)) are used for motion estimation.

Third, we would like to avoid using stationary features since they may correspond to shadows or objects moving with the vehicle, or points at infinity. So the feature should have sufficient disparity between the two images. We therefore assert that $\|\mathbf{x}_2 - \mathbf{x}_1\| > d_{\text{min}}$, where d_{min}

is the minimum disparity threshold defined in Section 5.2.4.

The fourth condition requires some prior justification. The third condition filters out features near the top of the ground ROI that, due to the perspective projection, move less than features near the bottom of the ground ROI. By the same token, the movement of features near the bottom of the ground ROI may produce sufficient disparity on the image, but correspond to insignificant ground motion. For this reason, we place an additional constraint on the feature's *ground projection*. Let $\mathbf{g}_1$ and $\mathbf{g}_2$ be the respective ground projections of $\mathbf{x}_1$ and $\mathbf{x}_2$, given by (4.12). Then we assert $\|\mathbf{g}_2 - \mathbf{g}_1\| > d_g$, where d_g is the minimum ground displacement threshold (in meters). We have found a practical value for d_g to be $0.1h$, where h is the camera height.

We summarize the criteria for selecting a good ground feature for motion estimation in the following proposition.

Proposition 6.1 (Selecting Good Ground Features). A *good* set of features for ground motion estimation between frames t_1 and t_2 must meet the following criteria:

1. The feature must be present at both frames t_1 and t_2.
2. The feature must currently be within the ground ROI.
3. The feature must have sufficient disparity between the frames. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the feature's ideal image coordinates at frames t_1 and t_2, respectively. Then we assert $\|\mathbf{x}_2 - \mathbf{x}_1\| > d_{\text{min}}$.
4. There must be sufficient displacement of the feature's ground projection. Let $\mathbf{g}_1$ and $\mathbf{g}_2$ be the respective ground projections of $\mathbf{x}_1$ and $\mathbf{x}_2$, given by (4.12). We assert $\|\mathbf{g}_2 - \mathbf{g}_1\| > d_g$.

6.3.2 The Motion Estimation Algorithm

Now that we have defined what it means for a feature to be a "good ground feature" (Proposition 6.1), we may define a refined ground motion estimation algorithm. This refined algorithm uses only good ground features as input to the planar motion estimation algorithm (Algorithm 4.2). Additionally, to increase robustness of the estimate we assert that the number of good features exceeds a minimum point-correspondence threshold N_{min}. We use $N_{min} = 10$.

Algorithm 6.1 (Planar Motion Estimation from Good Ground Features).
For a set of tracked features F, this algorithm estimates the planar motion $\{\mathrm{R}, \mathbf{T}\}$ between frames t_1 and t_2.

1. **Select good ground features.**
   Let F_g be the subset of "good ground features". That is,

   $$F_g = \{f \in F | f \text{ adheres to the criteria in Proposition 6.1}\}.$$

2. **Check that there are enough good ground features.**
   If $|F_g| < N_{min}$, exit the procedure.

3. **Estimate the motion.**
   Compute the planar motion $\{\mathrm{R}, \mathbf{T}\}$ between frames t_1 and t_2 using the point correspondences of F_g in Algorithm 4.2.

6.4 Snapshot Management

In Section 5.2.3 we showed that triangulation of correspondence points requires a non-zero translation of the camera. In fact, there must be sufficient translation to overcome the discretization of the image, image noise, and tracking error. For this reason we must ensure there is sufficient camera displacement between the frames for which we are attempting to perform triangulation.

To this end, we select keyframes for which the estimated camera displacement between them is greater than some appropriate threshold. We call these keyframes *snapshots*. The snapshots are triggered by vehicle displacement, rather than elapsed time. The faster the vehicle moves, the more frequently snapshots are taken, and vice versa.

A snapshot is simply indicated by its frame number. At any given time, we have a *snapshot list* $S = \{s_1,\ s_2,\ \ldots\ s_m\}$ consisting of the numbers of all the frames that were chosen as snapshots. It is precisely these frame intervals that are considered for our multiview triangulation procedure (Algorithm 5.1).

The purpose of the Snapshot Management module, then, is to modify the snapshot list S based on feature tracking. A modification to the list can be one of two things; (1) the clearing of the list, and/or (2) the addition of the current frame to list. The combination of clear the list and adding the current frame can be thought of as an initialization, or resetting of the list.

First, we define the condition for clearing the list. When tracking features, if the features in the ground ROI are too few, the motion estimation will be unreliable. Theoretically, only two features are required for a planar motion estimate. However, we would like to have more to increase the robustness of the estimate, so we use the minimum point-correspondence threshold N_{min} (Section 6.3.2)). So, if the number of features within the ground ROI is less than N_{min}, we clear the snapshot list. Otherwise, we consider the following situations.

Next, we define the conditions for resetting the list. Say the number of features within the ground ROI exceeds N_{min}, If the snapshot list is empty (as is the case when the system starts up) make the current frame the first snapshot.

Alternatively, if there exists a previous snapshot, but it was taken long ago, there is the possibility that features have drifted, and that many new features of been created or deleted in that interval. To avoid long-term tracking drifts and make use of recently detected features, we set a maximum snapshot interval threshold $\Delta_s = 300$. At 30 fps, this amounts to a maximum snapshot interval of 10 seconds. If the interval between the current frame and the last snapshot exceeds Δ_s, we reset the snapshot list.

Yet another alternative is that, although the current number of features may be great, the number of features that *existed at the previous snapshot* may be small. If that is the case, the motion estimation will be unreliable, since we would be comparing the motion between the current frame and the previous snapshot. Again we use the minimum point-correspondence threshold N_{min}, and reset the snapshot list if the number of features existing at the previous snapshot fall below N_{min}.

Last, we define the condition for adding the current frame to the snapshot list. If there are sufficient features in the ground ROI and that existed at the previous snapshot, we estimate the planar motion $\{\mathrm{R}, \mathbf{T}\}$ using Algorithm 6.1. If the motion was successfully computed (*i.e.*, there are enough "good" ground features), then we check if there is sufficient displacement between the frames. Let us define the snapshot displacement threshold d_s. Then the current frame is added to the snapshot list if $\|\mathbf{T}\| > d_s$. We have found a practical value for d_s to be $0.2h$, where h is the camera height.

The above discussion is summarized in the following algorithm.

Algorithm 6.2 (Snapshot Management).
For a list of features $F = \{f_1 \dots f_n\}$ of the current frame t_c, and a list of snapshot times $S = \{s_1 \dots s_m\}$, this algorithm decides whether or not to clear the snapshot list, and whether or not the current frame is a snapshot.

1. **Decide if the snapshot list should be cleared.**

Clear the snapshot list if the number of features (n) is less than the minimum feature threshold (N_{min}).

2. **Decide if the snapshot list should be reset.**
   Resetting the snapshot list means making the current frame the only snapshot, that is $S \leftarrow \{t_c\}$. This is done if any of the following conditions are met.

   (a) There are no previous snapshots ($S = \varnothing$).

   (b) The time since the last snapshot has exceeded the maximum snapshot interval threshold ($t_c - s_m > \Delta_s$).

   (c) The number of features present at the previous snapshot is less than N_{min}.

3. **Decide if this frame should be appended to the snapshot list.**
   Compute the planar motion $\{\mathrm{R},\ \mathbf{T}\}$ between the previous snapshot and the current frame using Algorithm 6.1. If the displacement ($\|\mathbf{T}\|$) of the camera is greater than the second displacement threshold (d_s), add this frame to the snapshot list.

If the current image is selected as a snapshot, and there are at least two elements in the snapshot list, then the Detection Block (Figure 6.1) is called. Otherwise, triangulated feature locations are updated in the Feature Location Module.

6.5 Inter-Snapshot Motion Estimation

In this section we describe how we estimate motion over multiple snapshot intervals. Say we have a list of snapshots $S = \{s_1 \ldots s_m\}$ where s_i is the frame number of the i^{th} snapshot. If the current frame is selected as a snapshot in the Snapshot Management module (Section 6.4), then s_m is the current frame. What we want is a motion estimate between each snapshot in the list and the last one (which is the current frame). This is depicted in Figure 6.4. Having all motions wrt. the current snapshot simplifies the multi-view triangulation in the Feature Triangulation module (Section 6.6).

These motion parameters could be obtained from previous motion estimates with little computational effort. However, to avoid the aggregation of tracking error and motion estimation error, we estimate these parameters directly each time a snapshot is taken.

The planar motion $\{\mathrm{R}_1, \mathbf{T}_1\}$ between the previous snapshot s_{m-1} and the current one s_m is estimated in the Snapshot Management module (step 3 of Algorithm 6.2), so it need not

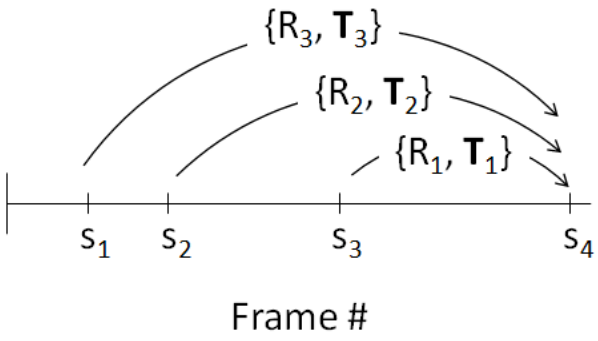


Figure 6.4: Inter-Snapshot motion estimation. Here four snapshots are plotted on a timeline. The latest snapshot s_4 is the current frame. All motion estimates are between previous snapshots and the latest one.

be computed again. We therefore begin by estimating the planar motion $\{\mathrm{R}_2, \mathbf{T}_2\}$ between snapshots s_{m-2} and s_m using Algorithm 6.1. If $\{\mathrm{R}_2, \mathbf{T}_2\}$ is successfully computed (*i.e.*, enough good ground features are found for the interval $(s_{m-2},\ s_m)$), then we proceed to estimate the motion between s_{m-3} and s_m. We continue until all the snapshot interval motions are estimated, or until one of the interval motions cannot be computed. Once we are finished, we have a list of planar motion parameters $M = \{\{\mathrm{R}_1, \mathbf{T}_1\}, \{\mathrm{R}_2, \mathbf{T}_2\}, \ \ldots\ \{\mathrm{R}_k, \mathbf{T}_k\}\}$. Here, $\{\mathrm{R}_i, \mathbf{T}_i\}$ is the motion between the snapshot pair $(s_{m-i},\ s_m)$), and $1 \geq k \leq m$ is the number of successful motion estimates.

The above discussion is summarized in the following algorithm.

Algorithm 6.3 (Inter-Snapshot Motion Estimation).
For a list of snapshot times $S = \{s_1 \ldots s_m\}$, this algorithm produces a list of planar motion parameters $M = \{\{\mathrm{R}_1, \mathbf{T}_1\}, \{\mathrm{R}_2, \mathbf{T}_2\}, \ \ldots\ \{\mathrm{R}_k, \mathbf{T}_k\}\}$, where $\{\mathrm{R}_i, \mathbf{T}_i\}$ is the motion between the snapshot pair $(s_{m-i},\ s_m)$), and $1 \geq k \leq m$ is the number of successful motion estimates.

1. **Initialize the list of motion parameters**
   The first inter-snapshot list is computed in the Snapshot Management module, so it need not be re-computed. Set $M = \{\{\mathrm{R}_1, \mathbf{T}_1\}\}$.

2. **Compute the remaining inter-snapshot motions.**

   for $i = 2$ to m **do**
   if $\{\mathrm{R}_i, \mathbf{T}_i\}$ is successfully computed using Algorithm 6.1 **then**
   append $\{\mathrm{R}_i, \mathbf{T}_i\}$ to M

else
break
end if
end for

6.6 Feature Triangulation

In this section we describe our procedure for triangulating features from multiple views.

From the Motion Estimation module we have a list of planar motion parameters $M = \{\{\mathrm{R}_1, \mathbf{T}_1\} \ldots \{\mathrm{R}_k, \mathbf{T}_k\}\}$. Each parameter set describes the camera motion from some snapshot in the past, to the current frame. Let s_i denote the past snapshot corresponding $\{\mathrm{R}_i, \mathbf{T}_i\}$.

For a given tracked feature, we must determine how many views (*i.e.*, snapshots) can be used in the triangulation. In order to use a particular snapshot, the feature must have existed at that frame, so as to have a corresponding image point. So, for each feature, we build a list of motions parameters $M' \subseteq M$ for which the feature has existed, and the corresponding set of image points $X = \{\mathbf{x}_1,\ \mathbf{x}_2,\ \ldots\}$. These sets are then passed to Algorithm 5.1, where the feature's depth wrt. the current frame is computed. If M' is empty, triangulation is not attempted and the depth remains undefined.

The following algorithm summarizes our multiview triangulation for tracked features.

Algorithm 6.4 (Feature Multiview Triangulation).
For a list of planar motion parameters $M = \{\{\mathrm{R}_1, \mathbf{T}_1\} \ldots \{\mathrm{R}_k, \mathbf{T}_k\}\}$, where $\{\mathrm{R}_i, \mathbf{T}_i\}$ relates a previous snapshot s_i to the current frame t_c, this algorithm computes the depth Z_1 of a given tracked feature f wrt. the current frame using the most views possible.

1. **Build the set of motions for which this feature has a correspondence point.**
   $M' = \{\{\mathrm{R}_i, \mathbf{T}_i\} \,|f$ exists at snapshot $s_i\}$
   $X = \{\mathbf{x}(t_c)\} \cup \{\mathbf{x}(s_i)|f$ exists at snapshot $s_i\}$
   where $\mathbf{x}(t)$ is the feature's position at frame t.

2. **Perform least squares multiview triangulation.**
   If M' is not empty, compute Z_1 using M' and X as inputs to Algorithm 5.1. Otherwise, set Z_1 as undefined.

6.7 Feature Labelling

With all the features robustly triangulated based on multiple views, the next step is to label the features as obstacles or not, based on the 3D location of the feature. We define an obstacle as any feature occupying the collision volume Figure 6.3 behind the vehicle. The height and width of the collision volume are determined by the size of the vehicle on which the OD system is installed. The depth of the volume (along the y-axis) can be made small if the user wants to remove long-range detections. The height of the volume must be set to reflect the sensitivity of the scene reconstruction; The volume height must be high enough such that there are not too many false detections of noisy ground features. We have found a practical collision volume height to be $0.2h$, where h is the camera height.

Our labelling scheme is the following. All features are initially labelled *undefined*, since no we have no knowledge of their motion. After a snapshot is taken and the Detection Block is called(Figure 6.1), triangulation is attempted for each feature.

If triangulation of a given feature is successful, then we can label that feature based on its 3D location. Any feature within the collision volume is labelled an *obstacle*. For features outside the collision volume, we have two possible labels. If they are below the collision volume height (in our case, $0.2h$), we label them *ground*. Otherwise, we label them *above-ground*. This ground/above-ground distinction is not critical to the detection aspect of the algorithm, but is useful in verifying that the scene is correctly interpreted.

Features that were not successfully triangulated may remain labelled as *undefined* for the purpose of static obstacle detection. However, we provide an extra label for detecting moving obstacles. If a feature has enough disparity on the image, but does not move along its epipolar line, this could indicate a moving object in the scene. When this occurs, we cannot triangulate the feature, but we can label it *moving*.

Specifically, this is accomplished by monitoring which conditions of Proposition 5.3 fail when attempting to triangulate a feature. If conditions 1 and 2 pass, but 3 fails, this indicates a feature is moving but not along its epipolar line.

Now, because we do multiview triangulation we must check these conditions for each pair of views. In Algorithm 5.1, Proposition 5.3 is checked for each pair of correspondence points. If all point pairs fail, and the majority reason for failure is that the point does not move along its epipolar line, then we label this point *moving*.

6.8 Feature Clustering

Unfortunately, no matter how good the video quality there are always features that are mistracked. Most mistracks are in directions other than the epipolar lines, and so are rejected by Proposition 5.3. However, due to smooth surfaces, edges and glare,

there remain a significant number of features that are mistracked such that the epipolar constraint is still obeyed. To be precise, 5 of our 14 test videos (see Chapter 7) show at least one instance of a false detection due such mistracks.

These mistracks cannot be disambiguated from good ones by means of epipolar geometry alone.

We have observed that the false detections through mistracks tend to be spatially sparse, both on the image and in the reconstructed space. Then, to filter out these false detections, we spatially cluster the detected features and set a minimum cluster size N_c for detection. This way, spatially sparse detections are filtered out. The disadvantage to this, of course, is that true detections from small, or sparsely featured obstacles are also filtered.

We cluster detected features in the following way. First, we randomly select a feature and create a cluster for that feature. Then, all other features that have a similar distance to the vehicle are added to the cluster. For all the features that were not accepted into the cluster, the process is repeated to form a second cluster. This continues until every feature is assigned to a cluster. Then we remove all clusters with fewer than the minimum cluster size.

The intention here is to cluster large surfaces perpendicular to the ground, so features are clustered based on their distance to the vehicle only. Specifically, we use the relative difference in the feature's world y-coordinate. If feature 1 and feature 2 have respective world y-coordinates Y_{w1} and Y_{w2}, then feature 1 is clustered with feature 2 if

$$\frac{|Y_{w1} - Y_{w2}|}{|Y_{w1}|} < w_c$$

where w_c is the *cluster width threshold*.

The clustering algorithm is as follows.

Algorithm 6.5 (Feature Clustering).
For a list detected obstacle features $F = \{f_1, f_2, \ldots, f_n\}$ with respective world y-coordinates $\{y_1, y_2, \ldots, y_n\}$ and a cluster halfwidth threshold w_c, this algorithm spatially clusters the features.

1. **Begin with the entire list of features**
   Begin with the list $F' = F$.

2. **Randomly select a seed feature**
   Randomly select a *seed* feature $f_s \in F'$ and create a cluster for that feature. Let the y-coordinate of the seed feature be y_s.

3. **Compare all other features to the seed feature**
   $\forall f_i \in F'$, if $\frac{|y_s - y_i|}{|y_s|} < w_c$ then add f_i to the cluster and remove it from F'.

4. **Repeat until all the features belong to a cluster**
   If there are unclustered features, go to step 2.

5. **Remove all clusters with fewer than N_c features.**

Of course, the final cluster configuration depends on the randomly chosen seed features. We would like a configuration that clusters the most features in the least amount of clusters. To obtain this we wrap Algorithm 6.5 in a RANSAC process and search for the configuration that maximizes n_f/n_c, where n_f is the number of features assigned to a cluster and n_c is the number of clusters.

6.9 Feature Location

If the current frame is not selected as a snapshot, the 3D location of previously triangulated features are updated in the Feature Location module. In this module, the world (x, y) coordinates of each triangulated feature is updated, given the features current location on the image $\mathbf{x}$ and its known (triangulated) world z-coordinate Z. The updated (x, y) coordinates are given by the expression $w(\mathbf{x}, Z)$ of (4.12). However, this expression diverges as the triangulated world height Z approaches the camera height. So we only update features whose height is not too close the camera's. We use a height similarity threshold of $h_s = 0.1h$.

Algorithm 6.6 (Feature Location).
For a feature whose world z-coordinate Z is known from a previous triangulation and a height similarity threshold h_s, this algorithm computes the feature's world (x, y) coordinates from its current image location $\mathbf{x}$.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} w(\mathbf{x}, Z) \\ Z \end{bmatrix} \qquad \text{if } |Z - h| > h_s$$

where h is the camera height, h_s is the height similarity threshold, and $w(\cdot)$ is given by (4.12)

In this chapter we have presented an overview of our OD algorithm and have described each component in detail.

Chapter 7

Experimental Results

7.1 Test Clips and Ground Truth

For testing the algorithm, we have installed a Boyo Vision CMOS rear view colour camera (VTB170) at the rear of a minivan and calibrated it using OpenCV's calibration routine [3], which estimates the camera matrix and distortion parameters from a series of checkerboard images, such as Figure 7.1 (a).

The camera's orientation, wrt. the ground was manually estimated as follows. Given a calibrated camera, if one knows the camera's orientation wrt. the ground, one can produce a *bird's eye view* [15] of the ground. That is, one can define a homographic transformation of the ground such that the image appears to have been taken from directly above. To manually estimate the orientation of the camera, we reverse this process: we adjust the angles of the camera wrt. the ground until we produce a good bird's eye. Namely, a "good" bird's eye view is one that preserves the geometry of known patterns on the ground, such as right angles and parallel lines. An example of this is shown in Figure 7.1 (b, c). On the left, we have a dewarped image taken from the installed camera. On the right is a bird's eye view of the ground region marked by the red trapezoid in the dewarped image. We have chosen the camera orientation such that the handicapped parking symbol is rectangular and that the parking lines are parallel in the bird's eye. The red horizontal line indicates the world horizon as seen from the camera, given our chosen orientation. This line serves as an extra check to make sure our chosen orientation is realistic.

We captured 14 videos of the vehicle backing into various stationary obstacles. The raw video resolution was 640×480 and the resolution after dewarping and cropping was 576×370. The obstacles include large objects (parked vehicles, shrubs, dumpster), medium sized objects

(a) (b) (c)

Figure 7.1: Lens distortion is estimated from a series of checkerboard images (a). Camera orientation is estimated manually by generating a bird's eye view. The bird's eye view (c) taken from the dewarped image (b) is produced by manually selecting the camera orientation w.r.t. the ground.

(garbage can, fire hydrant, leaf bag), and narrow objects (bike parking rack, wooden 2x4 used as construction site barricade). The videos are MPEG-2 compressed at 30 fps. Stills of these sequences are shown in Appendix A.

The algorithm, unoptimized, runs at 6 fps on a 2.3GHz Pentium computer. This corresponds to a processing time of 160ms per frame. The bulk of the processing time, 155ms, is due to KLT feature tracking. Feature detection and the detection of obstacles requires 5ms per frame. The optimized algorithm runs at 25 fps on CogniVue's hardware architecture.

Figure 7.2: The true object distance is established by manually indicating its base. Using the knowledge of the camera's orientation, height and intrinsics, the indicated pixel (white crosshair) is projected to the world ground plane.

For performance evaluation, we have established a ground truth of obstacle position by

manually marking the base of the objects on the image, for every frame. The knowledge of the camera height and orientation allow us to convert these pixel coordinates into world ground coordinates. Figure 7.2 shows the establishment of the ground truth for a garbage can. The base of the can is manually indicated (white cross hair), and this pixel is projected to the world ground plane. The world y-component of the projected point is the orthogonal distance to the vehicle, and is considered the *true* distance d_{true} (indicated on the top left).

Similarly, we take the world y-component of a detected (*i.e.*, triangulated) obstacle to be its *detected* distance, d_{OD}, to the vehicle. We say the true and reported distances are *similar* if their relative difference is less than one half, that is, if

$$\frac{|d_{\text{true}} - d_{\text{OD}}|}{d_{\text{true}}} < 0.5.$$

Then we define the following types of detection:

True Positive (TP):	An obstacle is present, detected, and the reported and true distances are similar
False Negative (FN):	An obstacle is present and undetected *OR* an obstacle is present, detected, and the reported and true distances *are not* similar
False Positive (FP):	No obstacle is present, but one is detected
True Negative (TN):	No obstacle is present, and none is detected

With these definitions we may tally the number of TPs, TNs, FPs and FNs for each snapshot and for each video. The confidence that a detection actually corresponds to an obstacle is the *precision*, and is given by

$$\frac{\text{\# of True Positives}}{\text{\# of True Positives } + \text{ \# of False Positives}}.$$

The confidence not to miss an obstacle is the *recall*, and is given by

$$\frac{\text{\# of True Positives}}{\text{\# of True Positives } + \text{ \# of False Negatives}}.$$

A perfect detector is one with a precision and recall both equal to one.

For a collision avoidance system such as this, a high recall is most important. A low recall means there is a greater chance of missing an obstacle, and so there is a greater risk of collision. On the other hand, a high precision is very important for a satisfactory user experience. If the precision is low, then many false detections arise, and the user will eventually stop paying attention to the detector. So ultimately, both precision and recall must be maximized to obtain an optimal collision avoidance system.

7.2 Parameter Tuning

The OD algorithm has 14 free parameters. Four of these are *tracking* parameters, while the rest are *obstacle detection* parameters. The tracking parameters have been set over the course of development to give the most visually intuitive tracking performance. We list the tracking parameters and their chosen values in the Table 7.1.

Symbol	Value	Description	Page Ref.
s_T	15×15	The tracking template size.	61
q_c	0.001	The corner quality (eigenvalue) threshold.	62
w_δ	5	The window size for evaluating trajectory smoothness	62
ϵ_δ	10	The smoothness of trajectory threshold	62

Table 7.1: List of tracking parameters.

The complete set of obstacle detection parameters are listed in Table 7.2. The parameters n, θ_{max} and d_s are crucial in feature triangulation. The first two decide what features are suitable for triangulation based on their motion (discussed in Section 5.2.4), and the third sets the minimum camera displacement between images. Another important parameter is the minimum obstacle cluster size N_c. Increasing minimum cluster size filters out spatially sparse false detections, but removes true detections from small objects. As such, it is imporant to understand the appropriate range of these parameters.

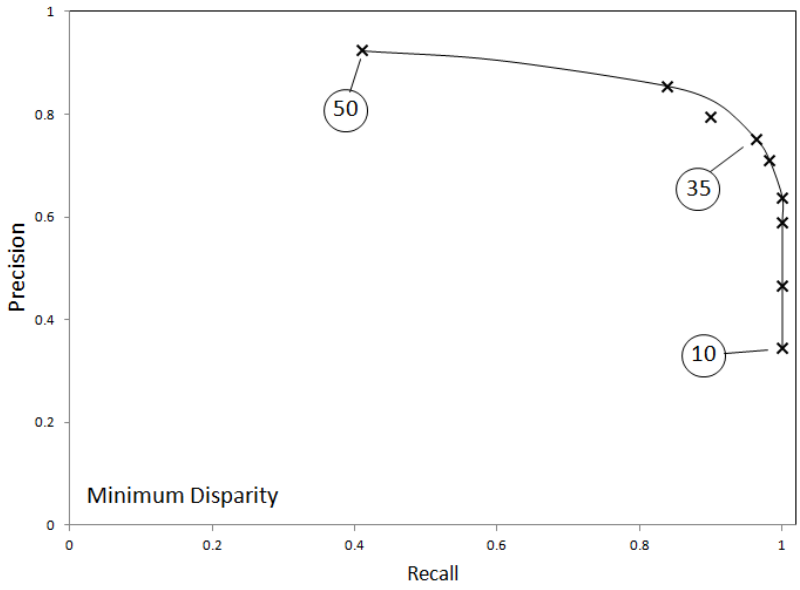
To evaluate the optimality of these parameters, we have varied each for a reasonable range and plotted precision VS recall graphs. Thus, we may optimize a given OD parameter by choosing the value that maximizes both precision and recall.

The precision-recall plots for the four chosen parameters are shown in Figure 7.3. The precision-recall plot for the minimum disparity threshold n is Figure 7.3 (a). We varied n from 10 (high recall, low precision) to 50 (low recall, high precision). We we observe the best algorithm performance for n ranging from 20 to 40 and an optimal value of 35, corresponding to a precision of 0.75 and a recall of 0.96.

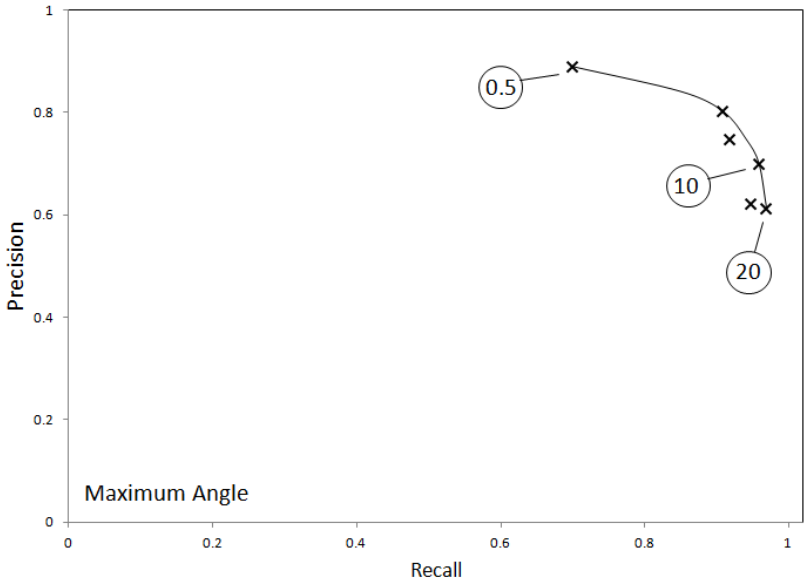
The precision-recall plot for the maximum angle between a feature's trajectory and its epipolar line, θ_{max}, is Figure 7.3 (b). We varied θ_{max} from 0.5° (high recall, low precision) to 20° (low recall, high precision). We observe the best performance for θ_{max} ranging from 1° to 10° and an optimal value of 1°, corresponding to a precision of 0.8 and a recall of 0.9.

The precision-recall plot for the snapshot displacement threshold d_s is Figure 7.3 (c). We varied d_s from $0.1h$ to $0.4h$ and observed a significant effect on the precision alone. The recall

(a)


Precision
Recall
Minimum Disparity
50
35
10
0
0.2
0.4
0.6
0.8
1

(b)


Precision
Recall
Maximum Angle
0.5
10
20
0
0.2
0.4
0.6
0.8
1

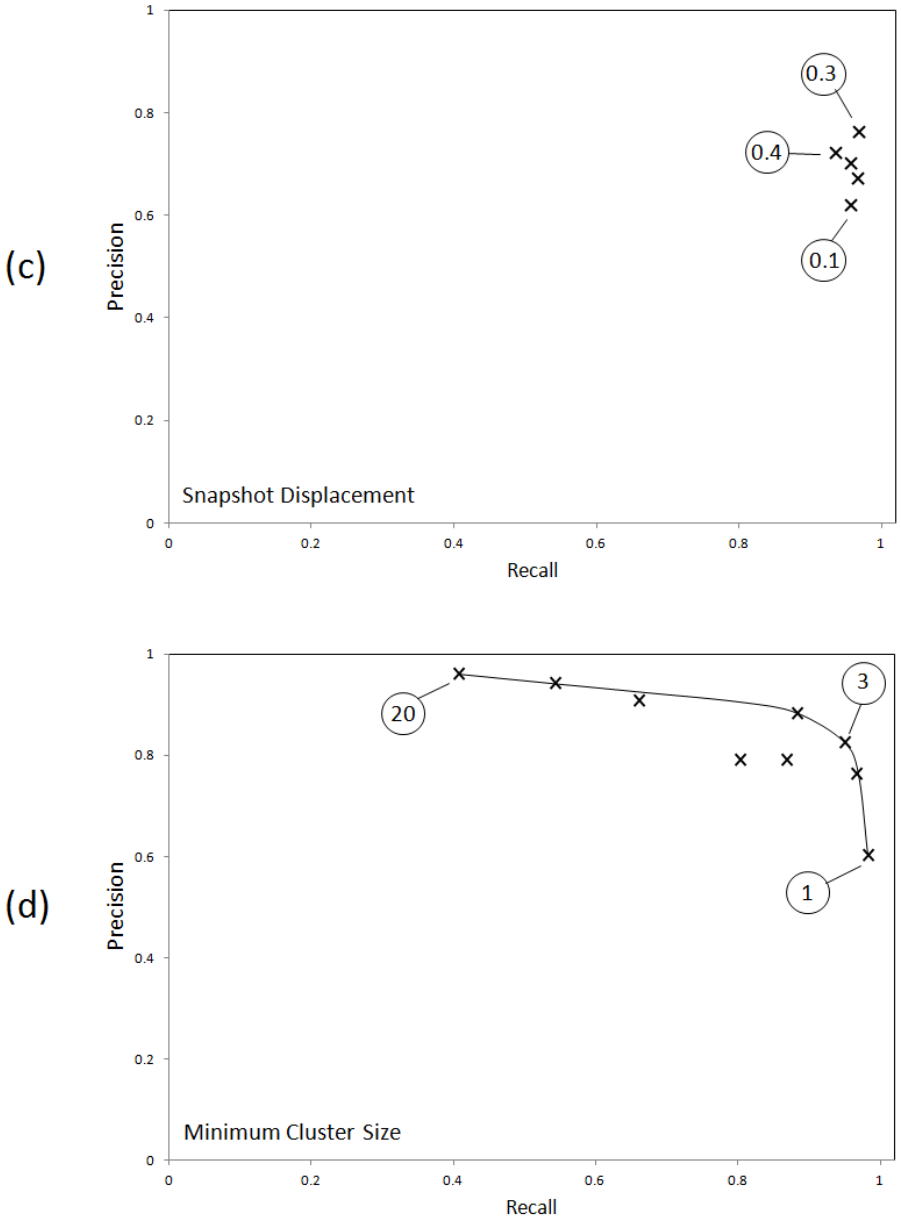


Figure 7.3: Precision VS Recall plots for various OD parameters. The parameter being varied is indicated on the bottom left of each plot.

Symbol	Value	Description	Page ref.
N_{min}	10	Minimum number of points required for various computations.	66
n	20	Minimum disparity in pixels.	55
$\theta_{\max}$	10°	Maximum angle between a point trajectory and its epipolar line.	55
Δ_s	300	Maximum number of frames between snapshots.	66
d_g	$0.1h$	Minimum ground displacement to consider a feature as moving.	64
d_s	$0.2h$	Minimum ground displacement to trigger a snapshot.	66
ϵ_r	$0.1h$	RANSAC reprojection error for rigid motion estimation.	30
h_s	$0.1h$	Minimum height difference between a feature and the camera in order to project that feature to the ground.	72
N_c	3	Minimum obstacle cluster size.	71
w_c	0.2	Minimum relative cluster depth.	71

Table 7.2: List of obstacle detection parameters.

remained at aproximately 0.95 for all values. We observe the best performance for d_s ranging from $0.2h$ to $0.4h$ and an optimal value of $0.4h$, corresponding to a precision of 0.76 and a recall of 0.97.

Finally, the precision-recall plot for the minimum cluster size N_c is Figure 7.3 (d). We varied N_c from 1 (low recall, high precision) to 20 (high recall, low precision). We observe the best performance for N_c ranging from 2 to 4 and an optimal value of 3, corresponding to a precision of 0.83 and a recall of 0.95.

These results are summarized in Table 7.3.

The highest performance observed is for the parameter values shown in Table 7.2, giving a precision of 0.83 and a recall of 0.95.

Parameter	Test Range	Acceptable Range	Optimal Value	Precision	Recall
n	10 - 50	20 - 40	35	0.75	0.96
$\theta_{\max}$	$0.1°$ - $20°$	$1°$ - $10°$	$1°$	0.8	0.9
d_s	$0.1h$ - $0.4h$	$0.2h$ - $0.4h$	$0.2h$	0.76	0.97
N_c	1 - 20	2 - 4	3	0.83	0.95

Table 7.3: Results of Precision-Recall analysis. For each parameter we show the range of values that were tested (Test Range), the range of values for which high precision and recall was observed (Acceptable Range), the optimal value and the associated precision and recall.

7.3 Algorithm Performance

7.3.1 Performance on Selected Clips

In this section we present the detection and distance estimation performance of the OD algorithm for selected set of sequences.

Figure 7.4 shows the output of the OD algorithm for three different obstacles; (a) a large dumpster, (b) a garbage can and (c) a bicycle parking rack. For each obstacle, we show a snapshot of the output when the obstacle is at an approximate distance of $2\mathrm{m}$ (top), 1m (center) and $0.5\mathrm{m}$ (bottom). The points drawn on the images are the features tracked by the Feature Tracking Module. These points are coloured to indicate the labelling provided by the OD module: green points are on the ground, blue points are below the ground, yellow points are above the ground (but not in the collision volume), red points are obstacles and white points have not yet been labelled. The distance to the nearest obstacle feature is displayed at the top-left of the image, and the projection of that distance on the ground is the horizontal red line. That is, the horizontal red line marks the base of the detected obstacle (provided successful detection). Note that the images shown are not the original input images, but the dewarped images obtained by camera calibration.

We have chosen these three obstacles to illustrate the behaviour of the algorithm over a wide range of detectable obstacle features. The dumpster (a) is large, flat and textured and so provides many detectable features for tracking. The garbage can (b) is of moderate size, rounded and smooth and so provides fewer detectable features. The bike rack (c) is short and tubular and so provides little in the way of detectable features.

In Figure 7.5 we show the true and estimated obstacle distances (top graphs) and the number of detected obstacle features (bottom graphs) for the sequences shown in Figure 7.4. The

Figure 7.4: Output of the OD algorithm for three objects: (a) a dumpster, (b) a garbage can and (c) a bicycle parking rack. The points drawn on the image are tracked features and are colour-coded as follows: Green = ground, blue = below ground, yellow = above ground (but not in the collision volume), red = obstacles and white = unknown.

true distance was obtained by manually indicating the base of the object (where the object meets the ground) at each frame. That pixel coordinate, combined with the intrinsic and extrinsic camera parameters give a world ground coordinate $[X\ Y\ 0]^{\mathrm{T}}$. The distance from the obstacle to the camera (equivalently, the rear bumper) along the car's axis (*i.e.*, the world y-axis) is $|Y|$.

The dumpster is accurately located at a distance of $2.3\mathrm{m}$ and remains so until the vehicle stops. The maximum number of detected features on the dumpster is 36. The garbage can is initially detected at $2.0\mathrm{m}$ and a stable, accurate localization is maintained below $1.5\mathrm{m}$. The maximum number of (stable) detected features on the garbage can is 18. Note that some of the features detected in the dumpster sequence actually belong to the wall behind the dumpster. This is because the detection results we are showing here are before any spatial clustering. So any feature detected is displayed, regardless of location.

The bike rack is initially detected at $1.9\mathrm{m}$, but a stable, accurate localization is only

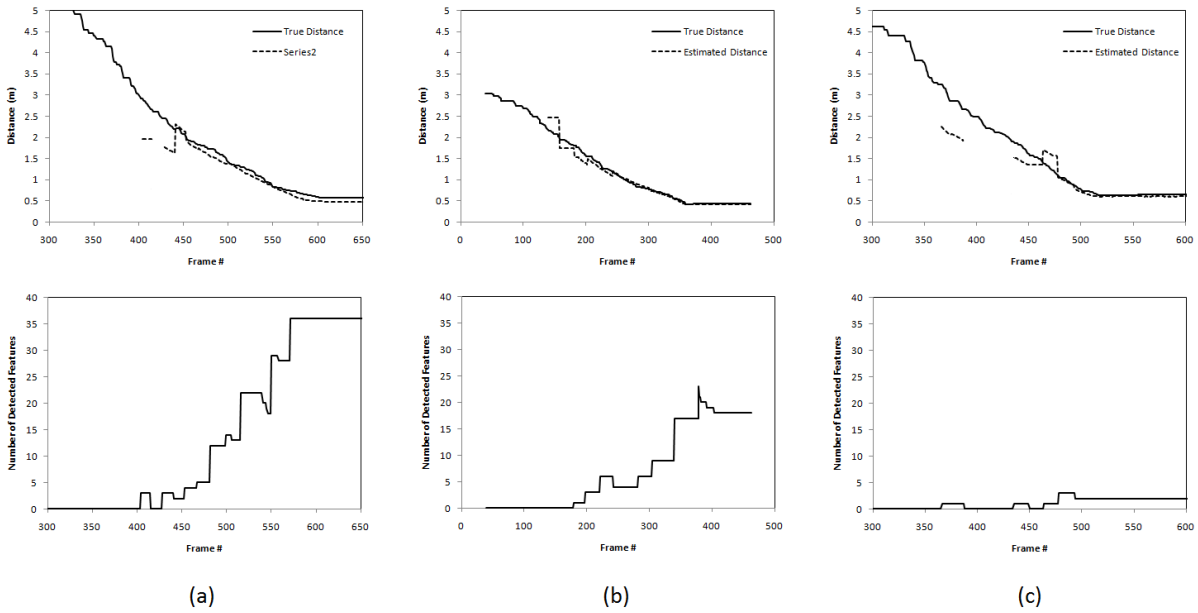


Figure 7.5: The true and estimated obstacle distances (top) and the number of detected obstacle features (bottom) for the sequences shown in Figure 7.4.

achieved below 1m. This detection is achieved with a maximum of only 3 stable detected features.

From these results we make the following observations. First, as one might expect, larger obstacles can be detected at a greater distance than small ones. The reason for this shown in Figure 7.6 is the following. Recall that a feature can be reliably triangulated only if it is sufficiently far from the epipole (stated formally as condition 2 in Proposition 5.3). For a vehicle backing towards an obstacle, the epipole typically (and unfortunately) lies somewhere on the object. As the obstacle gets closer, it's features diverge from the epipole. Once the obstacle is close enough, the features on its periphery are far enough from the epipole for triangulation (and thus detection) to occur. So naturally, the distance for which this occurs will be greater for larger objects (Figure 7.6 (a)) than smaller objects (Figure 7.6 (b)).

Second, the more textured the object, the more robust the detection. The dumpster is large and textured and so offers many features for detection. When it is closest to the vehicle (Figure 7.4 (a), bottom panel) a total of 36 of its features are detected. The garbage can is smaller and smooth, so there are fewer features available for detection. When it is closest (Figure 7.4 (b), bottom panel), a total of 10 of its features are detected (the additional detected features actually belong to the wall behind it). Lastly, the bike rack offers very few features for detection. When it is closest (Figure 7.4 (c), bottom panel), only two of its features are detected. Such a wide

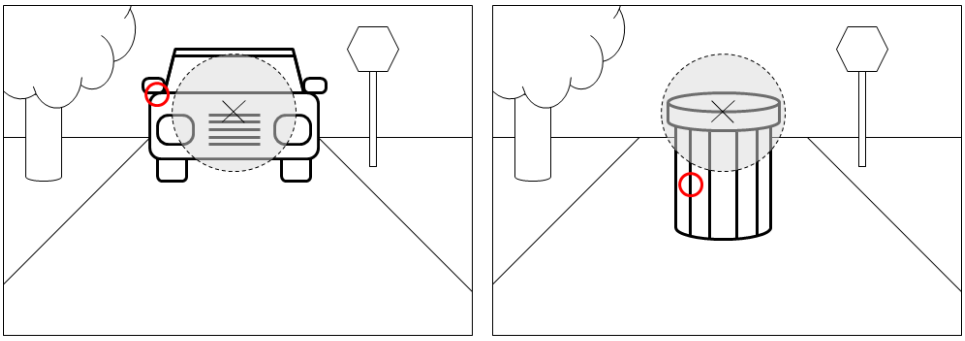


Figure 7.6: The peripheral features (marked by a red circle) of large objects are sufficiently far from the epipole (marked by an "X") at greater distances from the camera than smaller objects. For this reason, large objects may be detected at a greater distance than small ones.

range of object size and texture make it difficult to apply filtering schemes meant to reduce noise and false detections, because the detection of small, low-textured objects (such as the bike rack) would likely fail.

Third, the accuracy of the distance estimation is not dependent on the number of detected features. Although detection occurs at different distances (depending on object size), we observe similar accuracies in distance estimation for all three objects, once detection has occurred. This is due to our choice in what we report as the obstacle distance. We report the distance of the nearest detected feature and ignore all others, so the accuracy in distance estimation is the accuracy in triangulating any one feature. For any given feature, the accuracy in its triangulation depends on two things; the accuracy of ground motion estimation and the accuracy in the tracking of the feature itself. So, given similar quality in tracking and motion estimation we should expect similar accuracy in distance estimation, regardless of the number of features detected.

7.3.2 Overall Performance

In this section we evaluate the performance of the OD algorithm for all fourteen sequences. In addition to the precision-recall analysis provided in Section 7.2, we provide the following three performance metrics. First, we look at the density of detected features, an imporant issue for detection robustness. Next, we evaluate the range of the detector by looking at the detection rate as a function of obstacle distance. Finally, we examine the precision of the distance-to-obstacle estimate.

Detected Feature Density

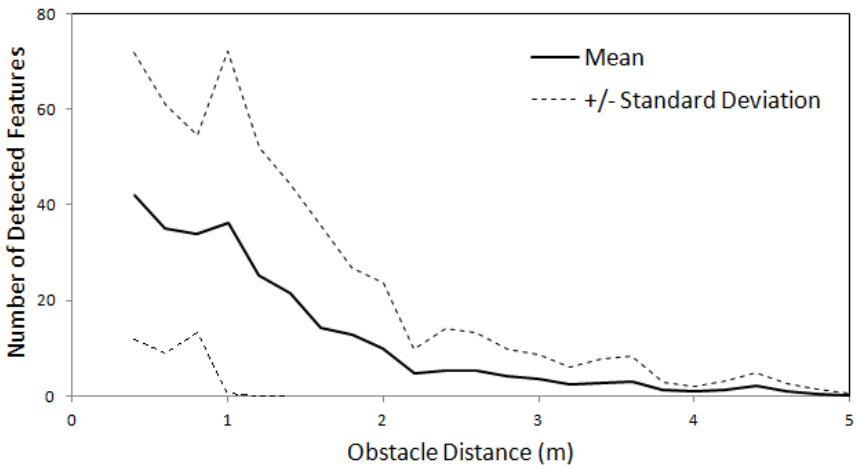


Figure 7.7: The average number of detected features as a function of obstacle distance. The upper and lower bounds (dotted lines) are the average $\pm$ one standard deviation.

Figure 7.7 shows the average number of detected features as a function of obstacle distance for all fourteen sequences. The dotted curves are the average $\pm$ the standard deviation for a given distance. The large variation in detected features at close range is a result of the widely varying size and texture of the objects used in our experiment. Some are small and smooth, yielding but a few trackable features, while others are large and grainy, giving more than 60 detected features. This large variation makes it particularly challenging to enhance detection performance via spatial filtering or clustering of detected features. As a case in point, the detection recall, shown in Figure 7.3 (d), drops dramatically as the minimum cluster size in increased beyond 4. In other words, small objects go undetected if we require large feature clusters.

Range of Detection

To evaluate the obstacle detection range we calculate the detection rate for a given true obstacle distance. Specifically, we divide a 5m detection range into 20cm bins. Let n_i be the number of snapshots for which an obstacle is located in bin i. Let m_i be the number of snapshots for which an obstacle is located in bin i *and* it was detected. Both n_i and m_i are counted over all fourteen videos. Then, the overall detection rate for the i^{th} is given by m_i/n_i. This can be thought of as the likelihood of detecting an obstacle, given the obstacles distance from the vehicle.

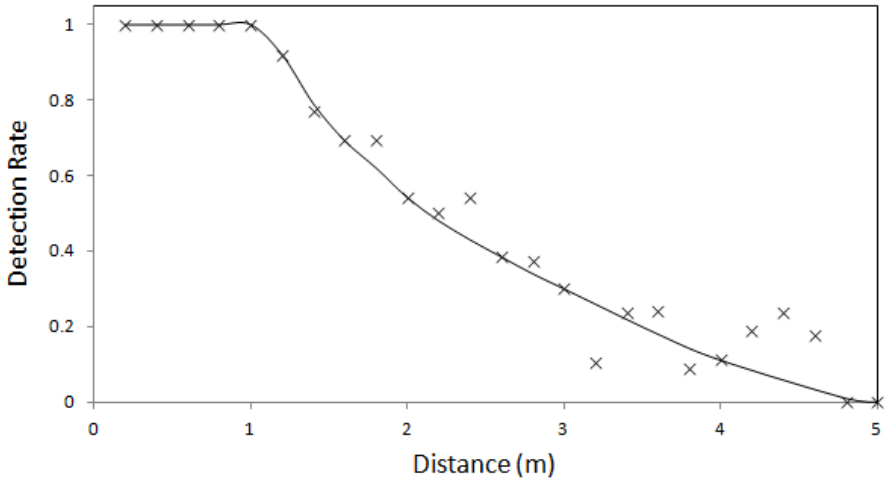


Figure 7.8: Detection rate as a function of obstacle distance.

As we see in Figure 7.8, the detection rate is roughly inversely proportional to the obstacle's distance. This can be explained from the following two observations: (1) The likelihood of detecting a feature on an object is proportional to the object's size on the image (*i.e.*, the apparent size). (2) An object's apparent size scales inversely proportional to its distance from the camera. So, we should expect the detection rate to be inversely proportional to the obstacle's distance.

The detection rate for obstacles 1m is 1. That is, in all fourteen videos, obstacles within 1m are detected 100% of the time. Obstacles at 2m are detected half of the time, and obstacles at 3m are detected one third of the time. The rapid drop in the detection rate is due to the unfortunate fact that obstacles are always in the vehicle's (and so, the camera's) direction of travel. This results in obstacle features having the least amount of disparity for a given camera displacement. If the camera were to strafe to the side of the obstacle, rather than towards it, the disparity would be much greater (maximal, in fact) and the detection rate would be proportionally so. It is this "minimal disparity" situation that is the greatest technical challenge in scene reconstruction in the context of a parking camera.

Precision of Obstacle Distance

To measure the precision of the distance-to-obstacle estimate provided by the detector, we compared the computed distances with the true ones. Again, we divide the detection range into 0.2m bins. We then compute the standard deviation of $d_{\text{true}} - d_{\text{OD}}$ (the difference between the true distance and the reported distance) for all occurences within a given bin. From Figure

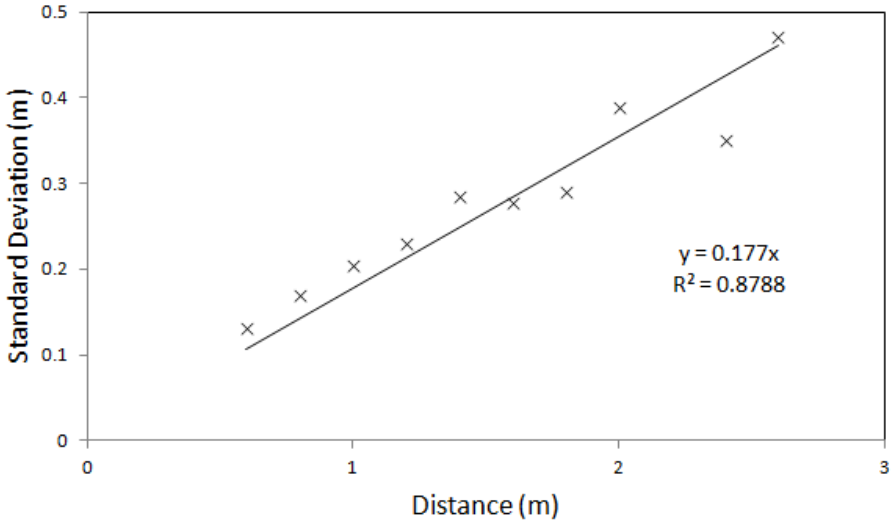


Figure 7.9: Standard deviation of distance-to-obstacle estimate as a function of distance.

7.9, we see that the standard deviation of $d_{\text{true}} - d_{\text{OD}}$ is proportional to d_{true} and has a slope of 0.177. If we define the uncertainty of the detector as being one standard deviation, then the reported distance has a relative uncertainty of 18%.

Limitations

This algorithm is designed to detect static obstacles through feature tracking. Features on moving obstacles generally do not obey the epipolar constraint. As a result, triangulation and subsequent detection is not attempted on these features. However, if the obstacle moves in the same direction as the vehicle, a detection will result. This is because the features of such obstacles still move along their epipolar lines. The disparity of these features, however, are shortened for outgoing obstacles, and amplified for incoming obstacles. The result is that incoming obstacles are detected as closer than their true distance and outgoing obstacles are detected as farther than their true distance.

The front end of this algorithm is feature tracking which assumes the constant illumination of features. So abrupt lighting changes, reflective surfaces and illumination by the vehicle's backing lights negatively impact algorithm performance. Low ambient light scenarios, such as parking garages, are particularly challenging since the vehicle's backing lights appreciably illuminate the scene. This makes for inconsistent lighting of the features and, as a result, erroneous tracking.

The detection system also has an inherent speed limit primarily dictated by the motion

blur of ground features. Motion blur of ground features increase with vehicle speed. And of course, the blurrier the ground features, the poorer the tracking of these features. Fortunately, for typical rear parking speeds (<10km/h) there is very little ground motion blur in a typical off-the-shelf CMOS camera.

7.3.3 Obstacle Detection on a Top View

This algorithm was designed to operate in real-time on Cognivue's multi-core, system-on image processing chip. Aside from hardware-specific optimizations that are beyond the scope of this thesis, an important conceptual variant was used to decrease execution time of the algorithm. This variant is the following: instead of estimating the motion and reconstructing the scene from the original (dewarped) images, we do so from a top (bird's eye) view transformed image. Such an image is shown in Figure 7.1.

Running the algorithm on a top-view has three main advantages:

1. The top-view can be as much as 4 times smaller than the input image, so there is a much smaller area on which to perform feature tracking

2. Ground features on the top-view are not perspective skewed, but undergo only translation and (slight) rotation. As such, we can track features using a simple block-matching scheme, which is much faster than the KLT tracker.

3. Using a top-view effectively narrows the detector's focus to the region directly behind the vehicle, in the sense that the periphery of the input image are not even processed.

As a result, and largely thanks to the switch from KLT tracking to block-matching, the algorithm runs much faster on the top-view transformed image. Note that conceptually the algorithm remains the same, except the triangulation formulae in Section 5.2 are modified to operate in top-view image coordinates rather than ideal coordinates.

There is, however, one noteworthy disadvantage of using the top-view for detection. Because the top-view is essentially a trapezoidal region of the dewarped image mapped to a rectangular image (see Figure 7.1), the taller an object is, the closer it must be to the vehicle to enter the top-view. What's worse, objects above the height of the camera never enter the top-view at all. So, using this scheme, the system is blind to any obstacle above the height of the camera.

Chapter 8

Conclusion

In this thesis we have presented 3D reconstruction-based obstacle detection method for a single rear view parking camera. We track ground features and use the knowledge of camera pose to efficiently and robustly estimate the planar motion of the vehicle. Snapshots are selected to ensure a sufficient baseline between images. Multiview triangulation is performed among these snapshots. Once the scene is reconstructed, features located within a collision volume behind the vehicle are considered obstacles. To increase robustness, detected features are spatially clustered. Finally the distance to the nearest obstacle is reported to the driver.

The proposed system successfully meets the objectives of reliability and efficiency as demonstrated by experimentation. Namely, we have shown a very high detection rate for obstacles within 1m of the vehicle. The algorithm runs unoptimized at 6fps on a standard PC. Using an optimized top-view variant, the algorithm runs at 25fps on CogniVue's hardware architecture.

Our contributions to the field of collision avoidance are the following. We have developed a novel method for static obstacle detection using a single parking camera. This method uses a novel keyframe system to reconstruct the scene in 3D based on images taken at equal distance intervals as the vehicle moves. The method is general enough to work on a top-view-transformed image of the scene (with modification of the triangulation expressions in Section 5.2). This allows for tracking to be done on a smaller image which focuses on the ground directly behind the vehicle. Tracking on the top-view image a critical element in the optimization that allows the algorithm to run in real time Cognivue's system-on chip. Our most significant contribution, then, is an effective and productized static obstacle detector that runs in real time on Cognivue's system-on chip.

8.1 Future Work

Currently, the vehicle motion is estimated from ground features only. Robustness in the estiamation may be increased if all image features were considered. This would be especially beneficial when the vehicle is backing through a clutrered scene. Incorporating all image features into motion estimation may also allow us to estimate camera pose directly from the images. This would reduce the calibration procedure to simply specifying the camera height. It would also allow the system to recover from sudden camera pose changes.

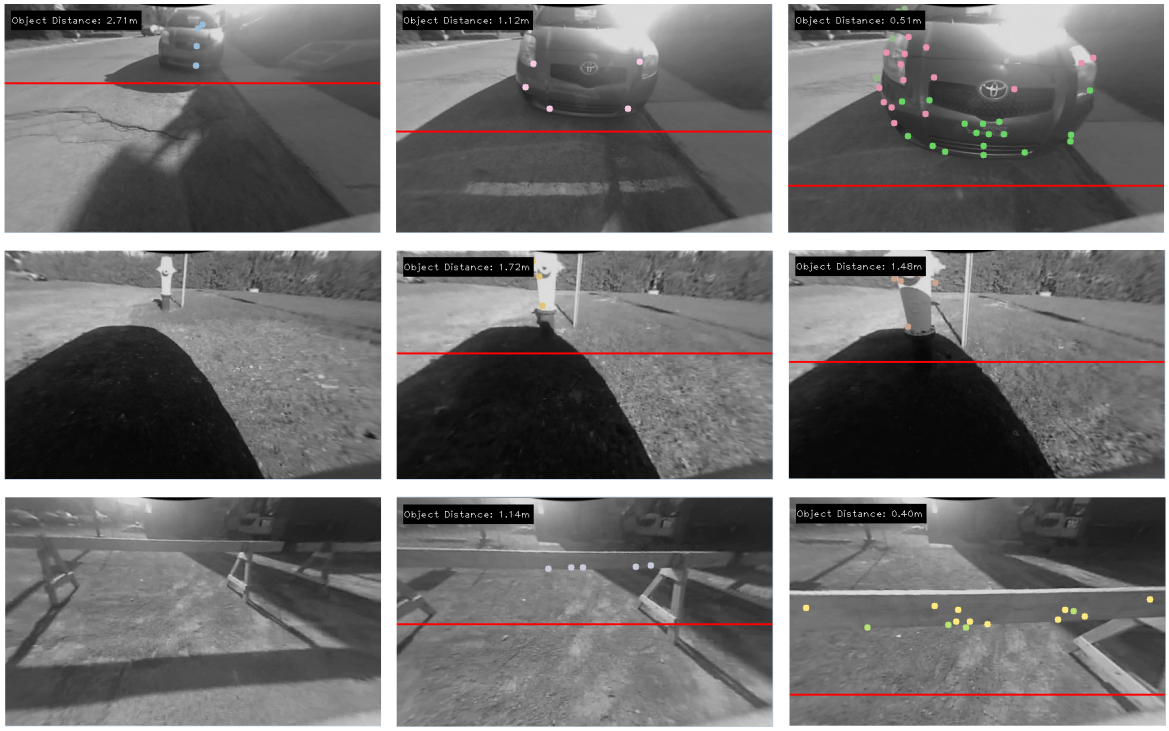
The most difficult challenge in developing this system was the reduction of false detections due to tracking error. Our solution of spatial clustering improved the false positive rate but made the detection of small or narrow objects less reliable. Tracking error may be decreased through the use of more sophisticated features, such as SIFT or SURF. Another solution may be a bundle adjustment across the multiple views.

The algorithm presented here deals only with static obstacles. Obstacles moving wrt. the scene, when tracked on the image, do not obey the epipolar constraint and so cannot be triangulated. However, this fact can be used to detect the presence of moving obstacles within the scene. Actually locating moving obstacles, on the other hand, is quite difficult. For rigid objects moving within the scene, tracked features may be fit to multiple motion models and each feature is triangulated using its respective motion. However, this is computationally intensive and requires dense and reliably tracked features. Even so, this would not work for non-rigid objects, like pedestrians, whose individual parts have many local motions. Therefore, a very interesting challenge would be the design of collision avoidance system for moving rigid and moving non-rigid bodies.

Appendix A

Video Clips

Here we show still shots from all 14 videos used in our evaluation of the obstacle detection algorithm. Sequences 2, 5 and 7 are omitted as they are shown in Figure 7.4.




6
Object Distance: 2.24m
Object Distance: 1.46m
Object Distance: 0.81m
8
Object Distance: 2.45m
9
Object Distance: 2.94m
Object Distance: 1.88m
Object Distance: 0.71m
10
Object Distance: 2.70m
Object Distance: 1.10m
Object Distance: 0.82m
11
Object Distance: 2.70m
Object Distance: 1.68m
Object Distance: 0.71m
12
Object Distance: 1.45m
Object Distance: 0.79m
Object Distance: 0.33m

13


Object Distance: 1.84m


Object Distance: 1.35m


Object Distance: 0.94m

14


Object Distance: 1.82m


Object Distance: 0.63m


Object Distance: 0.32m

Bibliography

[1] Parag Batavia and Sanjiv Singh. Obstacle detection in smooth high curvature terrain. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, May 2002.

[2] James R. Bergen, P. Anandan, Th J. Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. pages 237–252. Springer-Verlag, 1992.

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. O'Reilly Media, Inc., 2008.

[4] Tommy Chang, Steve Legowik, and Marilyn N. Abrams. Concealment and obstacle detection for autonomous driving. In *Proceedings of the Robotics & Applications 1999 Conference*, pages 28–30, 1999.

[5] Wilfried Enkelmann. Obstacle detection by evaluation of optical flow fields from image sequences. *Image Vision Comput.*, 9(3):160–168, 1991.

[6] A. Ess, B. Leibe, K. Schindler, and L. Van Gool. A mobile vision system for robust multi-person tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[7] Robert M. Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13(3):331–356, 1994.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.

[9] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19:580–593, June 1997.

[10] Berthold K. P. Horn and E J. Weldon. Direct methods for recovering motion. *International Journal of Computer Vision*, 2(1):51–76, 1988.

[11] Ian Horswill. Visual collision avoidance by segmentation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 902–909. IEEE Press, 1994.

[12] Michael Jenkin and Allan Jepson. Detecting floor anomalies. In *Proceedings of the British Machine Vision Conference, BMVC-94*, pages 731–740.

[13] BerndManfred Kitt, Joern Rehder, AndrewD Chambers, Miriam Schonbein, Henning Lategahn, and Sanjiv Singh. Monocular visual odometry using a planar road model to solve scale ambiguity. In *Proc. European Conference on Mobile Robots*, September 2011.

[14] W. Kruger. Robust real-time ground plane motion compensation from a moving vehicle. *Mach. Vision Appl.*, 11:203–212, December 1999.

[15] Robert Laganière. Compositing a bird's eye view mosaic. In *In Proc. Conf. Vision Interface*, pages 382–387, 2000.

[16] Jeffrey Lalonde, Luc Martel, and Robert Laganière. Single-camera distance estimation. U.S. Patent App. No. 13/208919, August 2011.

[17] Gildas Lefaix, ric Marchand, and Patrick Bouthemy. Motion-based obstacle detection and tracking for car driving assistance. In *ICPR (4)'02*, pages 74–77, 2002.

[18] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155–166, 2009.

[19] Liana M. Lorigo, Rodney A. Brooks, and W. E. L. Grimson. Visually-guided obstacle avoidance in unstructured environments. In *IEEE Conference on Intelligent Robots and Systems*, pages 373–379, 1997.

[20] Manolis I. A. Lourakis and Stelios C. Orphanoudakis. Visual detection of obstacles assuming a locally planar ground. In *Proceedings of the Third Asian Conference on Computer Vision-Volume II*, ACCV '98, pages 527–534, London, UK, 1997. Springer-Verlag.

[21] Steven Lovegrove, Andrew J. Davison, and Javier Ibanez-Guzman. Accurate visual odometry from a rear parking camera. In *Intelligent Vehicles Symposium*, pages 788 – 793, 2011.

[22] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. pages 674–679, 1981.

[23] Yi Ma, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision - From Images to Geometric Models*. Springer-Verlag New York, Inc., 2004.

[24] Ezio Malis. Improving vision-based control using efficient second-order minimization techniques. In *ICRA*, pages 1843–1848. IEEE, 2004.

[25] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652 – I–659 Vol.1, june-2 july 2004.

[26] David Nister. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:756–777, June 2004.

[27] J. M. Odobez and P. Bouthemy. Robust multiresolution estimation of parametric motion models. *Jal of Vis. Comm. and Image Representation*, 1995.

[28] J. Oliensis and Yakup Genc. New algorithms for two-frame structure from motion. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 737–, Washington, DC, USA, 1999. IEEE Computer Society.

[29] D Scaramuzza, F Fraundorfer, and R Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *Proc. of The IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[30] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.

[31] Gideon P. Stein, Ofer Mano, and Amnon Shashua. A robust method for computing vehicle ego-motion. In *In IEEE Intelligent Vehicles Symposium (IV2000)*, 2000.

[32] Toshihiko Suzuki and Takeo Kanade. Measurement of vehicle motion and orientation using optical flow. In *1999 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems.*, pages 25 – 30, 1999.

[33] A. Talukder, R. Manduchi, A. Rankin, and L. Matthies. Fast and reliable obstacle detection and segmentation for cross-country navigation. In *In IEEE Intelligent Vehicles Symposium*, pages 610–618, 2002.

[34] Iwan Ulrich and Illah Nourbakhsh. Appearance-based obstacle detection with monocular color vision. In *Proceedings of AAAI 2000*, 2000.

[35] D. Willersinn and W. Enkelmann. Robust obstacle detection and tracking by motion analysis. pages 717 – 722, 1997.

[36] K Yamaguchi, T Kato, and Y Ninomiya. Moving obstacle detection using monocular vision. *2006 IEEE Intelligent Vehicles Symposium*, pages 288–293, 2006.

[37] Zhongfei Zhang, Richard Weiss, and Allen R. Hanson. Obstacle detection based on qualitative and quantitative 3d reconstruction. *IEEE Trans. on PAMI*, 19:15–26, 1997.