MIRAGE: $O(N)$ TIME ANALYTICAL MULTI-CAMERA POSE ESTIMATION METHOD WITH APPLICATION TO TRAJECTORY TRACKING PROBLEM

by

SEMIH DINC

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
The Department of Computer Science
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2016

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this dissertation.

Semih Dinc (date)

DISSERTATION APPROVAL FORM

Submitted by Semih Dinc in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and accepted on behalf of the Faculty of the School of Graduate Studies by the dissertation committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate of the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

Dr. Ramazan S. Aygun	(Date)	Committee Chair
Dr. Farbod Fahimi	(Date)	
Dr. Letha Etzkorn	(Date)	
Dr. Timothy S. Newman	(Date)	
Dr. Huaming Zhang	(Date)	
Dr. Heggere Ranganath	(Date)	Department Chair
Dr. Sundar Christopher	(Date)	College Dean
Dr. David Berkowitz	(Date)	Graduate Dean

ABSTRACT

School of Graduate Studies
The University of Alabama in Huntsville

Degree Doctor of Philosophy College/Dept. Science/Computer Science

Name of Candidate Semih Dinc

Title Mirage: $O(n)$ time Analytical Multi-Camera Pose Estimation Method with Application to Trajectory Tracking Problem

Camera pose estimation is a critical stage for the vision based robotic control systems since a precise motion can only be achieved by successful localization step. Today, many robotic systems such as unmanned vehicles, robotic arms, or various automation systems already rely on vision systems. Besides this, by the effect of various emerging technologies as depth sensors, virtual/augmented reality headsets, the need of accurate and reliable pose estimation algorithms is significantly increasing. To serve this purpose, this disseration proposes a novel pose estimation algorithm called *Mirage*. Mirage utilizes a desired camera pose and calculates the pose parameters by minimizing the 2D projection error between desired and actual pixel coordinates. Our methodology employs target pixel errors in 2D image plane and analytically calculates the robot's pose in 3D Euclidean space. Therefore, complex computations and undesirable Euclidean trajectories are avoided.

In order to evaluate the performance of Mirage, pose estimation experiments have been performed using simulated/real data on noisy/noise-free environments. The results are compared with the state-of-the-art techniques. Mirage outperforms other methods by generating fast and accurate results in all tested environments. Mirage

has also been applied to the trajectory tracking problem on a simulated environment and real robotic system (two-wheeled nonholonomic ground vehicle). Experimental results show that Mirage generates very promising results with low error rates and it demonstrates the practical applicability to the real time systems.

This dissertation also introduces new solutions to two problems that needs to be addressed before and after the pose estimation stage. First, a new GPU based feature mapping method is presented to generate reliable feature points for Mirage. And second, a pose recovery algorithm is proposed to re-localize the camera in the case of failure of pose calculation. Both methodologies are tested under varying conditions and promising results have been achieved.

Abstract Approval: Committee Chair _____
Dr. Ramazan S. Aygun

Department Chair _____
Dr. Heggere Ranganath

Graduate Dean _____
Dr. David Berkowitz

ACKNOWLEDGMENTS

I still need to be written.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE PAGE

LIST OF SYMBOLS

SYMBOL	DEFINITION
$\mathbf{q}$	6 Degrees of Freedom Pose
$\mathbf{T}_{\mathbf{X}}^{\mathbf{Y}}$	4x4 3D Transformation Matrix from frame X to Y
$\mathbf{p}^{X}$	3D point in X frame
$\mathbf{e}^{C}$	Pixel errors between the actual and desired images
$\mathbf{u}$	Velocity vector
$\mathbf{s}$	2D image features

Dedication

CHAPTER 1

INTRODUCTION

1.1 Motivation

Unmanned vehicles are robotic systems without an on-board operator. They can be controlled either by an operator remotely or a control algorithm that utilizes various sensor inputs. In the last 2 decades, massive amount of advancements have been observed regarding unmanned vehicles for both military and civilian purposes, some of which are earth observations, fighting forest fires, transportation, or personal entertainment. There is clearly a rising trend of using these systems, and it is not difficult to estimate that this trend will continue to rise when considering the capabilities of these systems. Particularly today, most of the major car companies are investigating on automated features for their new models. These features include automatic parking, automated breaking systems, or even automatic driving (autopilot).

Minimizing the human intervention is a major issue for the autonomous vehicles. Recent systems are not mature enough for a fully reliable end-user experience since majority of these systems require human guidance even for basic missions. One of the major challenges for autonomous vehicles is *trajectory tracking* problem that is normally solved by GPS (Global Positioning System) technology [1]. However, using

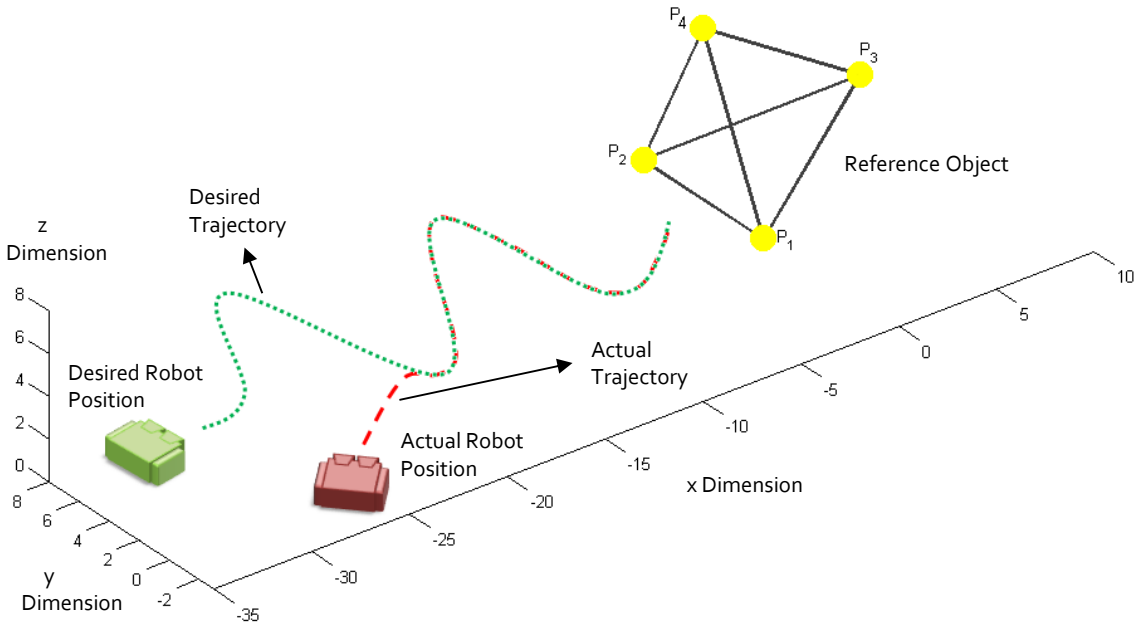


Figure 1.1: Vision based trajectory tracking system

GPS is not a feasible solution when the signal is weak, jammed, or lost. Moreover, the accuracy of pose measurement using GPS is not satisfactory for missions requiring high precision. In such cases, *vision-based* methods are good alternatives with high precision and robustness [2]. As Figure 1.1 shows, in vision-based trajectory tracking, an autonomous vehicle is expected to follow a predefined trajectory (path) by getting visual feedback from camera(s). The vehicle reconstructs its position/velocity and angle/rotation rates with respect to a reference (target) object at any instant of time during the motion.

1.2 Our Approach

This research proposes a novel pose estimation method called *Mirage* for vision based trajectory tracking problem. In the proposed methodology, the autonomous

vehicle utilizes a reference (target) object in the environment to estimate its 6DOF (degrees of freedom) pose and move accordingly on a desired path with a given speed. A 3D CAD model of the reference object is used as a visual clue on the images, so that the 3D position and orientation (pose) of the vehicle can be calculated. The name *Mirage* refers to the analogy between the literal meaning and the employment of desired (unreal) pose in the calculations. The method has linear time complexity for multi-camera systems and it analytically calculates the Euclidean pose of the vehicle in real time. The foundation of the method is presented in [3], where 2 identical cameras and a 4 point reference object is employed for calculation on a grid shape surface. Kindly note that capabilities of our algorithm is not limited to these conditions. Mirage allows one or more (nonidentical) cameras. It can be integrated as an independent module that replaces navigational sensors in unmanned aerial vehicles as well as underwater or ground vehicles. Chapters 4 and 5 provide more information about these expansion and integration steps.

Mirage pose estimation needs two types of information to estimate the pose. The first one is the set of target image(s) that are captured by the individually calibrated camera(s) on the vehicle. And the second one is the desired pose of the vehicle for that time instant. No explicit 3D pose measurement (via depth sensors) or depth-analysis (stereo-vision structure) is required to calculate vehicle pose. The major advantage of our system is that it computes the 3D pose error directly without computing actual 3D pose by reducing the error on 2D image plane using linear calculations and feeding the pose error directly to the controller.

1.3 Feature Mapping and Pose Recovery

In order to support comprehensive usability of Mirage in real world problems, in this dissertation, we introduce two new methodologies to solve "feature mapping" and "pose recovery" problems. Feature mapping is a critical stage for pose estimation to build the correspondence of 2D image features and 3D object points. It is a challenging problem since feature extraction and matching tasks are usually sensitive to noise, illumination changes, outliers, and large transformations. By exploiting the parallel power of the Graphical Processing Units (GPU), we propose a GPU based feature mapping technique that is robust and accurate.

In the case of a failure on pose calculations, a pose recovery algorithm is also necessary for tracking systems, since it is a likely condition for real world applications. Typically, a new pose calculation relies on the previous results and a miscalculation may amplify the error in future calculations. To prevent a complete system failure, we propose an alternative pose recovery algorithm that utilizes additional virtual poses to calculate the correct pose.

1.4 Summary of Contributions

The contributions of this dissertation may be summarized as follows:

1. We propose Mirage pose estimation algorithm that

    (a) performs 6 DOF (degrees of freedom) motion in 3D Euclidean space,

(b) needs only analytical computations, so it has approximately the same computational economy as that of image based systems, while lacks the problem of undesirable Euclidean trajectories,

(c) needs only single vision to operate, but supports multi-camera systems without using stereo-vision,

2. We propose a new feature mapping algorithm that

(a) constructs robust 2D-3D feature correspondence using the idea of Hough transform,

(b) supports large pose transformations and robust to the outliers,

(c) runs in parallel using GPUs and NVIDIA CUDA library

3. We propose a new pose recovery algorithm that

(a) utilizes additional virtual views of reference object to recover the vehicle pose,

(b) is robust and accurate under various challenging conditions

1.5 Organization

The remaining chapters of this dissertation is organized as follows: In Chapter 2, a detailed coverage of related work is given for pose estimation and trajectory tracking problems. Chapter 3 provides the details about feature mapping problem and our GPU based solution. In Chapter 4, Mirage is described in detail, and mathematical derivations are given. Later in the same chapter, pose estimation experiments

are shown. Chapter 5 presents an example application of Mirage to trajectory tracking problem. The details are given for different type of vehicles and experimental results are shown on both simulated and real environments. In Chapter 6, our new pose recovery methodology is presented and experimental results are shown. Finally Chapter 7 concludes the dissertation.

CHAPTER 2

RELATED WORK & BACKGROUND

Pose estimation is a well studied problem that has very large literature background. Since a broad range of interpretations of the problem are already exist, the background of the problem must be well understood. Also, before conducting a new research in the domain, a comprehensive survey of the state-of-the-art must be investigated. This chapter serves these purposes by presenting problem background and a detailed literature survey on existing pose estimation techniques. State-of-the-art techniques are compared with respect to several key features and proposed pose estimation method is categorized using those features.

In the second part of the chapter, we provide a background and literature survey of vision based trajectory tracking studies. Powerful and weak features of the methods are provided.

2.1 Pose Estimation

Camera pose estimation is the problem of calculating 6 pose (position and orientation) parameters (Figure 2.1) of calibrated camera(s) using a set of point correspondences in 3D space and their projections on 2D image plane (Figure 2.2). Pose

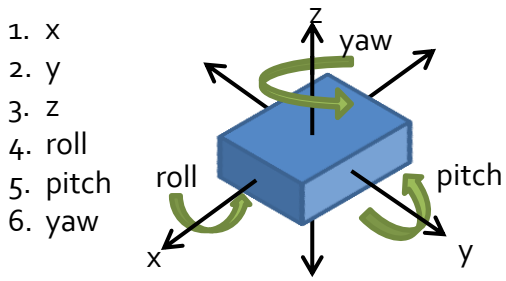


Figure 2.1: 6 parameters of an object in 3D space

estimation plays a critical role in many applications such as target tracking, robotics, and augmented reality [4]. In target tracking and robotic applications, estimated pose generates a feedback to localize the vehicle, or assists some other sensors for decision making during the motion. For augmented reality applications, it is crucial to estimate the pose of the camera precisely, so that the artificial images can be inserted into the display properly. In the literature, the problem has been studied as *Perspective-n-Point (PnP)* [5] or *object localization* [6] depending on the problem domain.

The number of 2D-3D point correspondences is an important factor in the pose estimation. It is often preferred to solve the pose estimation problem with small number of points to reduce the burden of feature extraction and matching stages. The general case of the problem is PnP where n is an arbitrary number. If the number of points is restricted to 3 ($n = 3$) the problem becomes P3P. Similarly, it is named as P4P when 4 points ($n = 4$) are necessary.

There has been significant work on the PnP problem in the last 2 decades due to its practical significance in many applications. A good number of informative

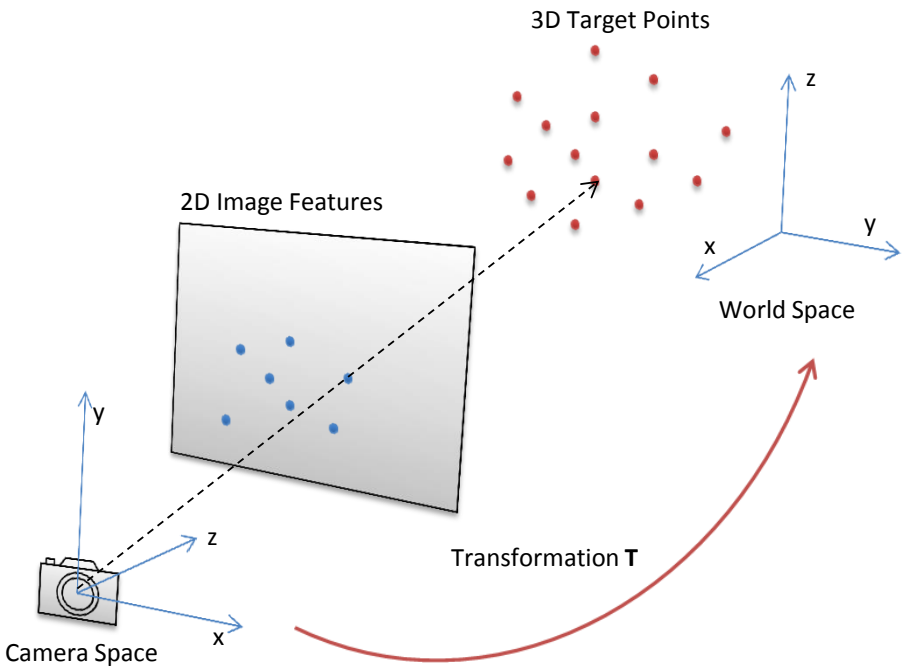


Figure 2.2: Camera Pose Estimation

studies ([7], [8], [9]) exist in the literature. We can broadly classify the PnP studies as numerical and analytical solutions. Numerical methods are iterative methods that minimize a specific error (geometric or algebraic [10]) by optimizing an objective function. These methods are robust to noise and they can benefit from high number of matching points since they can iteratively search the solution space. However, they are susceptible to be trapped in local minima, which prevents them converging to the optimal solution. An exhaustive search can be initiated to overcome the local minima entrapment problem, but it would have high computational cost. On the other hand, analytical methods can directly solve the parameters as unknowns of an equation system to reach the optimal solution. However, the time complexity of these methods may not be tractable for real-time systems. In addition, these

methods are usually sensitive to noise, which causes an incorrect pose estimation. The noise in this context results from inaccurate computation of pixel positions of feature points. Another categorization can be made based on time complexity, multi-camera support, and determining the 3D positions of feature points. Next, we briefly look into the related work from these perspectives and later give a comparison of methods in Table 2.1

2.1.1 Time Complexity

The minimal case of PnP is P3P, where only 3 matching points are employed to solve unknowns in the equation system. Mathematically, this requires solving an 8-degree polynomial that may yield up generally two but up to 4 solutions, which causes an ambiguity [4]. Thus, in later studies, additional feature points (4 points in [11], 5 points in [12]) have been included into the solutions to clarify this ambiguity. Theoretically, using a small number of points is sufficient to calculate the pose, however, recent studies show that redundancy of the points helps to avoid planarity and reduce the negative effect of the noise [13]. Many existing studies target to get benefit from redundancy and utilize all available points [14], [5], [15]. The time complexity of these methods include $O(n^8)$ (method by Ansar et al. [16]), $O(n^5)$ (method by Quan et al. [17]), and recent $O(n^2)$ (methods in [18], [19]). Finding the optimal solution in *real-time* is as important as reaching the optimal solution. The methods with time complexity of $O(n^b)$ where $b > 1$ are not suitable for real-time applications. First well-known linear time solution is Direct Linear Transform (DLT) [20], where the transformation parameters are solved without involving any constraints. Even

though it is relatively a fast algorithm, it is very sensitive to noise, therefore not applicable for real world applications. To increase resiliency to noise, Lepetit et al. (2009) [21] proposed the EPnP method with $O(n)$ time complexity. EPnP generates 4 reference points from n points and solves the pose parameters using these reference points. Based on EPnP, subsequent linear time solutions have been proposed such as Hesh et al. [22], Li et al. [23], Zheng et al. [13], and Ferraz et al. [10]. Although these methods increased the robustness of the system, our experiments show that there is still room for further improvements for high noise rates and low number of feature points.

2.1.2 Multi-camera Support

Another important factor in the domain has been emerged by the recent advancements of vision systems. As hardware prices were reduced, multi-camera systems have become available at reasonable prices to users. There is a clear advantage of using a multi-camera system over a single camera system since multi-camera systems provide an extended field of view or allow depth calculations via stereo vision [24]. However, multi-camera PnP problem has rarely been studied in the literature. Majority of existing studies is designed for a single-camera structure. Clearly, this does not indicate that the existing techniques cannot be extended, but rather the extension is unclear and not tested. For instance, some methods [25], [26] may require the same feature points to be visible in all cameras, while some others [19], [24], [27] do not have this requirement. Therefore, the new techniques should explicitly address how to deal with feature points from multiple cameras. To the best of our knowledge,

Kneip et al. [24] propose a well-defined method called gPnP that has $O(n)$ time complexity with multi-camera support. Although their method is more resilient to noise than the EPnP method, the translational error of the gPnP is roughly 3 times of the EPnP method with 100 feature points available under their experimental setting. However, it is still considered as competitive with the EPnP method with respect to the number of feature points.

2.1.3 Determination of 3D Positions

The PnP methods assume the availability of the 3D positions of the feature points. The studies can be divided into two groups in that sense. The first group of studies uses a specific equipment (e.g., depth sensors) in their system to determine 3D positions of feature points on the reference object [28]. For instance, Dryanovski et al. [29] propose a camera trajectory estimation using RGB-D camera. After extracting feature points on the image and estimating 3D positions using their uncertainty model, their method tries to register 3D positions to a model built using previous frames. In another study, Choi et al. [18] propose an iterative method for object pose estimation using multiple point clouds of objects. In their evaluations, they pay more attention to accuracy of detection and recognition rather than pose estimation errors. In the second group, methods employ a complete or sparse 3D model of the object to determine the 3D coordinates of feature points [30]. In this study, we also prefer 3D model based solution, since there is no requirement for an additional depth sensor, and constructing the 3D model of any object is done only once offline.

Table 2.1: Comparison of well-known and recent pose estimation methods

Paper Name	Complexity	Depth Sensor	#Cameras
Dementhon (1995) - [POSIT]	$O(mn)$ + Iterative	No	1
Lu (2000) - [LHM]	Iterative	No	1
Ansar (2003)	O(n)	No	1
Chang (2004)	Iterative	No	Multiple
Stewenius (2006)	Iterative *	No	2**
Lepetit (2009) - [EPnP]	$O(n)$	No	1
Chen (2009)	Iterative	No	Multiple**
Noell (2010)	Iterative *	No	1
Hesch (2011) - [DLS]	$O(n)$	No	1
Choi (2012)	$O(n^2)$ + Iterative	Yes	1
Li (2012) - [RPnP]	$O(n)$	No	1
Jaramillo (2013)	Iterative	No	1
Lee (2013)	$O(n^2)$ + Iterative	No	Multiple
Dryanovski(2013)	Iterative	Yes	1
Kneip (2013) - [gPnP]	$O(n)$	No	Multiple
Zheng (2013)	$O(n)$	No	1
Ferraz (2014)	$O(n)$	No	1
Fabian (2014)	$O(n)$	Yes	1
Vandenhouten (2015)	Iterative	No	1
Mirage	$O(n)$	No	Multiple

* Partially iterative, iterative approach is used in small part of the solution
** All cameras are expected to capture the same set of points.

2.2 Trajectory Tracking

In vision-based trajectory tracking systems, the motion instructions (linear and angular velocities) of the vehicle are calculated by getting visual feedbacks from 2D image features of a reference (target) object. The studies regarding vision-based trajectory tracking problem can be classified with respect to *camera placement* and *pose error computation* [31]. There are three types of camera placement approaches (Figure 2.3): 1) *eye-to-hand*: the camera is placed on a fixed position [32], 2) *eye-in-hand*:

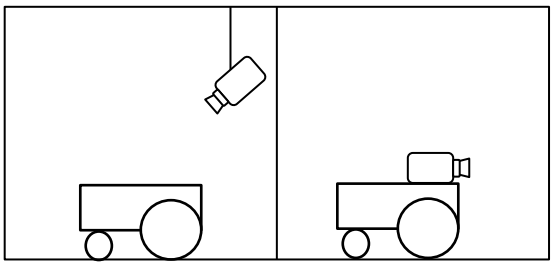


Figure 2.3: Eye-to-hand (left) and Eye-in-hand (right) robotic systems

the camera is mounted on the moving vehicle [33], and 3) *hybrid systems*: these support both mechanisms [34].

Another categorization is made regarding the method of pose error measurement [35]. In the first group of studies, the vehicle is guided such that pixel errors vanish in 2D image space *(image based visual servoing)*. In that approach, visual features (edges [36] or corners [37]) on the image are employed with corresponding interaction matrix to determine new linear and angular velocity of the vehicle [38] (Figure 2.4). These methods are free from camera calibration and usually considered to be robust. However, 3D motion of the vehicle cannot be controlled directly, because the Euclidean pose of the vehicle is not calculated. Marchand et al. [39] proposed *virtual visual servoing* framework that adopted an alternative image based (2D visual servoing) approach for real time augmented reality applications. In 2007, Mariottini et al. [40] proposed an image-based control method for non-holonomic mobile robots. Their algorithm was established based on epipolar geometry calculations using current and desired camera views. In 2008 Choi et al. [41] presented another image-based tracking system based on Kande-Lucas-Tomasi (KLT) tracker. Their

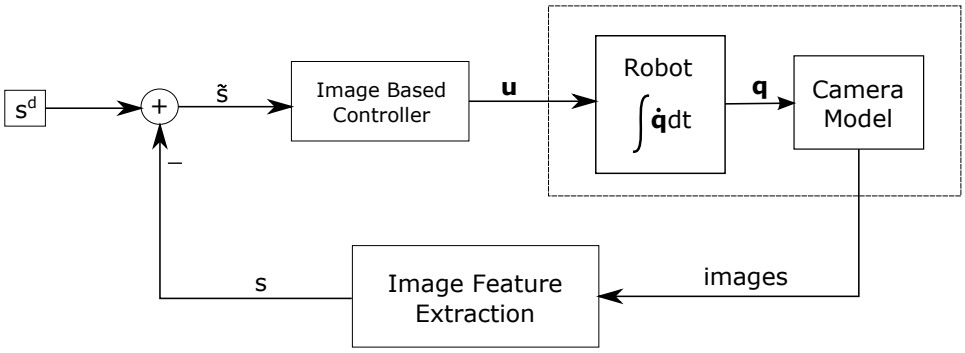


Figure 2.4: Overview of typical image based visual servoing

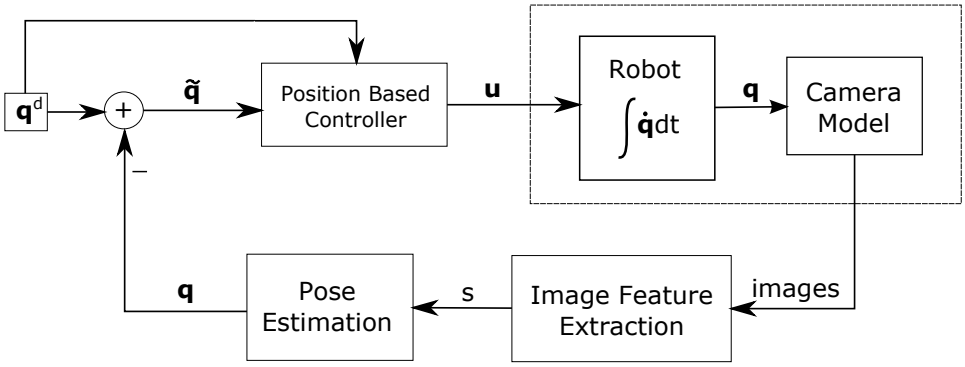


Figure 2.5: Overview of typical position based visual servoing

method runs under the assumption of known 3D geometric model with SIFT description of an object. A recent study in 2014 [42] employs homography decomposition to formulate the kinematic model of the mobile robot after determining the fixed depth parameter. The method was tested on simulated experiments and promising results have been received.

The second group of methods minimize the pose error in 3D Euclidean space by using actual 3D pose of the vehicle *(position based visual servoing)* [43]. This procedure often requires a pose estimation method that calculates the vehicle pose in 3D space using visual features (Figure 2.5). Beavidez et al. [44] proposed a tracking

system by utilizing MS Kinect RGB-D camera, which simplifies the pose measurement problem since a direct depth measurement is possible. A fuzzy logic controller is used to control the robot. Elqursh et al. [45] investigated the robot pose estimation by using 2 parallel lines and 1 more line orthogonal to the others in the image. Estimated pose is employed to control the mobile robot. However, pose estimation part usually involves complex calculations or numerical (iterative) solutions, which reduces the preferability of the method [46]. Even though there exist $O(n)$ time solutions such as [21], [10], our previous experiments show that the robustness is still a challenging problem to solve.

2.3 Summary

This chapter investigates the background of pose estimation and trajectory tracking problems. A detailed literature survey of the problems has been established. Key features of state-of-the-art techniques are presented and the methods are categorized based on several characteristics. Weaknesses and strengths of each category is presented to provide a clear understanding of the methods to decide which approach performs better under specific conditions.

CHAPTER 3

FEATURE MAPPING

Feature extraction and mapping are one of the essential steps of vision based trajectory tracking and pose estimation problems. At this stage, the images are processed and salient 2D features of the scene are extracted. Later, these 2D features are mapped to the corresponding 3D coordinates of the scene. 2D-3D feature mapping process is achieved by several approaches. In the first approach a depth sensor or a stereo-vision system is utilized to find the 3D coordinates of 2D features. This technique requires additional equipment or computational complexity that leads researchers to another approach. In the second technique, a 3D CAD model of a reference (target) object is used to construct the 2D-3D relations (Figure 3.1). This study follows the idea of this approach, in which a desired image of the object is generated from a known camera pose and then it is registered to the actual image. In other words, 2D features the actual and desired images are extracted and matched. And finally, using the desired image and the pose, 3D coordinates of each matching features can be found.

In this chapter, we present our alternative GPU based image registration technique that can be used to calculate 2D-3D feature correspondences. Image registra-

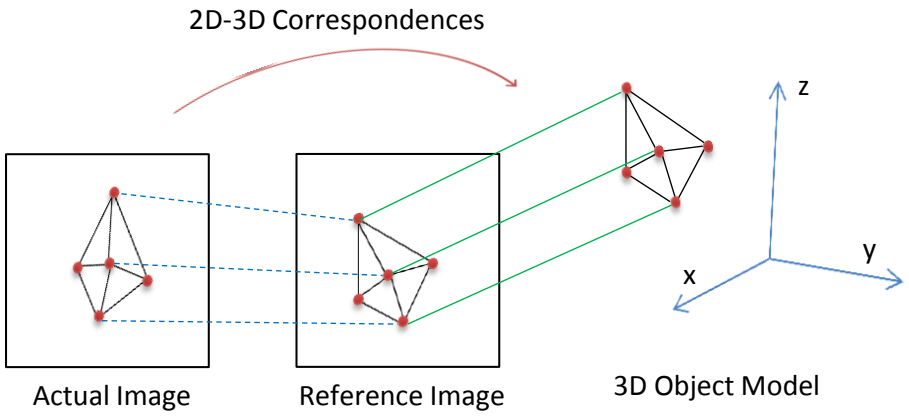


Figure 3.1: 2D-3D feature mapping using 3D object model

tion is aligning process of multiple images using their distinctive visual features [20]. Many applications have an image registration stage such as medical imaging, geospatial visualization, or mosaic image generation. In the literature, a good number of registration studies exist [47] [48]. CPU based methods mainly suffer from the trade-off between processing speed and accuracy due to the burden of feature extraction and matching. Moreover, majority of these algorithms propose iterative solutions that prevents them to be parallelized easily. As an alternative, GPU based solutions may provide better accuracy without sacrificing the speed. Few GPU based image registration studies exist, however, most of them are limited to a problem domain or they support limited transformations. For instance, Kubias et al. [49] studied the 2D/3D image registration for only translation and rotation on X-Ray like images. Kohn et al. [50] proposed GPU based image registration system on medical images but the registration accuracy is not provided in the paper. Sinha et al. [51] proposed a

GPU based feature tracking technique using SIFT and KLT, where matching features are tracked but image frames are not registered.

Our GPU based image registration technique is implemented using NVIDIA CUDA libraries [52]. Our system employs raw matching features generated by scale invariant feature transform (SIFT) descriptor [53], and determines the transformation between images by checking all possible combinations of feature points via Hough transform [54]. Since feature points are processed exhaustively, our method supports transformations with significant translation, scaling, or rotation. Experimental results show significant speed-ups with respect to CPU time while generating robust image registration results.

3.1 2D Image Transformation

Digital images can be considered as 2D matrices where each pixel has x and y coordinates and an intensity value of that position. When we apply a 2D transformation to an image, we basically transform the coordinates of pixels to new coordinates according to a given transformation matrix.

In order to have such transformation, a transformation matrix is required that comprises 3 major transformations: Translation, Rotation, and Scale. Usually the goal is to find a composite transformation that includes all major transformations. Additional transformations can also be included to list such as shearing, skewness, and reflection but they are not considered as major transformations. If additional transformations were included, the complexity of the problem would be higher. In this study, we consider only major transformations that have 4 parameters to calculate:

2 translation, 1 rotation and 1 scale parameters. Equation (3.1) shows the formula that transforms an image **A** to image **B**.

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{A} \tag{3.1}$$

where t_x, t_y, θ, and s are translation, rotation and scale parameters, respectively. Equation (3.2) shows final form of the transformation after the multiplication of individual transformation matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.2}$$

where (x', y') in **B** corresponds to the (x, y) coordinates in **A**. Once we solve 4 transformation parameters, we can use this transformation matrix to register all remaining pixels of **A**.

3.2 Calculation of Transformation Parameters

Equation (3.2) can be solved by reorganizing its terms such that all unknowns stay as a single vector **t** with the coefficient matrix **Q** on one side of the equation, and constant vector **W** stays on the other side. Equation (3.3) shows the new form of the system that is solvable simply by inverting **Q**.

$$\overbrace{\begin{bmatrix} x' \\ y' \end{bmatrix}}^{\mathbf{W}} = \overbrace{\begin{bmatrix} x & -y & 1 & 0 \\ y & x & 0 & 1 \end{bmatrix}}^{\mathbf{Q}} \overbrace{\begin{bmatrix} s\cos\theta \\ s\sin\theta \\ t_x \\ t_y \end{bmatrix}}^{\mathbf{t}} \tag{3.3}$$

Note that the rank of the **Q** matrix is not enough to solve unknowns by employing a single match. That means, at least 2 pixel matches are necessary to obtain a full rank (non-singular) **Q**. Solving all unknowns individually requires nonlinear calculations. In order to keep all computations linear, we assumed the entries of **t** are our 4 unknowns. The main reason behind this assumption is that solving **t** is sufficient to find the transformation. Based on these considerations, if we rewrite Eq. (3.3) for the following unknowns as $a = s\cos\theta$, $b = s\sin\theta$, $c = t_x$, $d = t_y$, and add the second match to the equation, final form of the equation becomes,

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \end{bmatrix} = \begin{bmatrix} x_1 & -y_1 & 1 & 0 \\ y_1 & x_1 & 0 & 1 \\ x_2 & -y_2 & 1 & 0 \\ y_2 & x_2 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{3.4}$$

At this point, the equation can be solved using any technique as long as **Q** has full rank. Equation (3.5) shows the calculation of the **t** vector using Least Squares Optimization (LSO). We prefer to use LSO because it does not restrict number of matches in the equation. We have the flexibility to include more matching pixels. So,

if there were a third matching pixel, we could add two more rows to the Eq. (3.4) and solve it with new values. A good advantage is that this update does not require further modification in Eq. (3.5).

$$\mathbf{t} = (\mathbf{Q}'\mathbf{Q})^{-1}(\mathbf{Q}'\mathbf{W}) \tag{3.5}$$

When $\mathbf{t}$ is calculated, the original transformation parameters can be obtained. c and d are assigned to t_x and t_y, respectively. s and θ are obtained using the following calculations: $s = \sqrt{a^2 + b^2}$ and $\theta = atan2(b/a)$.

3.3 GPU Based Robust Image Matching

LSO is a very flexible method with some important weaknesses. If there are high number of incorrect matches (outliers) in Eq. (3.4), the results may be affected significantly, since LSO is sensitive to the invalid data in the equation. Unfortunately, almost all feature extraction techniques (such as SIFT) extract invalid pixel matches, which shall not be underestimated to have accurate results. To solve this problem, in this paper, we propose a robust approach that calculates all possible solutions and searches for a common solution in the solution space. This method is an exhaustive approach based on Hough transform. As mentioned earlier, this idea is not practical for CPU implementation but can be a good solution for GPU implementation.

Assume that N matching features are extracted using SIFT descriptor. Since only 2 matches are needed to solve unknowns, we can choose all possible 2 matches out of N. So there will be

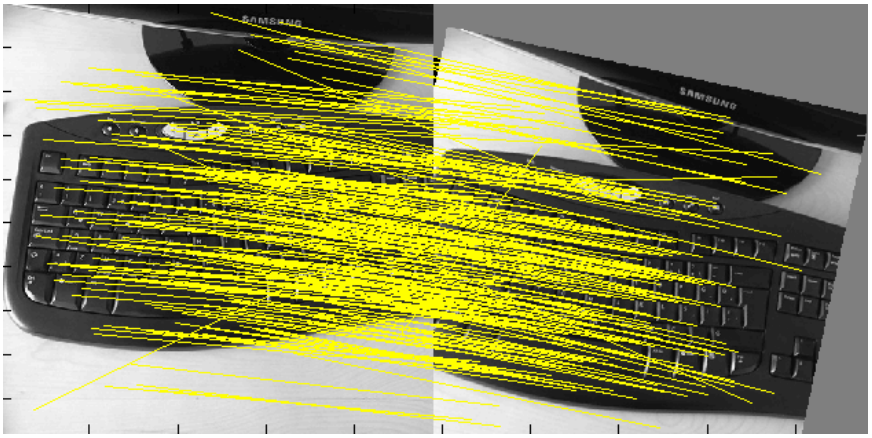


Figure 3.2: Pixel matches generated by SIFT

$$M = \binom{N}{2} = \frac{N!}{2!(N-2)!} \tag{3.6}$$

number of system of equations. For each equation system, we solve 4 unknowns and store the results. Our assumption is that the majority of matches are correct and majority of combinations, which use these correct matches, will return correct solution. Theoretically, if two combinations are both correct, they must return exactly the same unknown values from their equations. In this manner, we can eliminate the incorrect matches that cannot agree on a single solution. After all combinations are solved, correct solutions will be closer to each other in the solution space and incorrect solutions will be outliers. Finally, if we calculate the histogram in 4 dimensional space and group the solutions based on their distances, the most populated histogram bin center will be the final solution to the problem. The complexity of our solution is $O(n^2)$, which can easily be derived from Eq. (3.6). Herein, n is the number of matches and power 2 indicates the number of matches necessary.

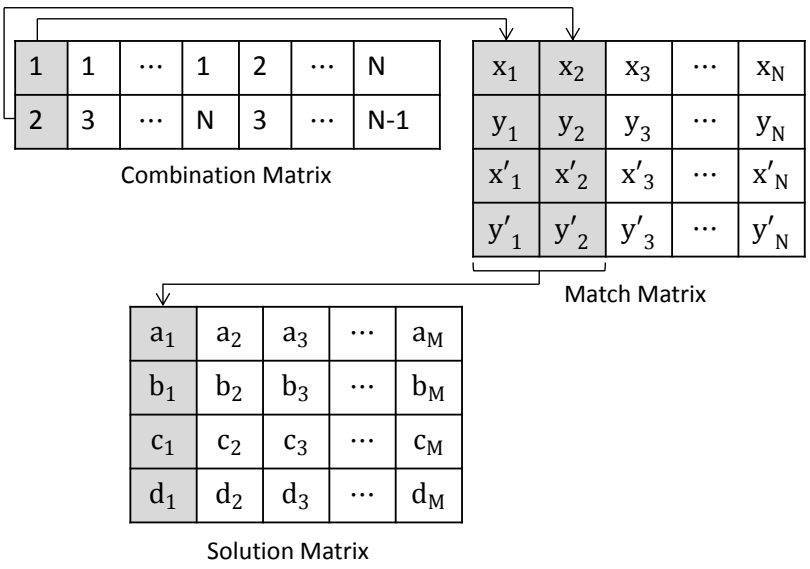


Figure 3.3: Input and output matrices in CUDA Kernel for each matching pixel coordinates

In CUDA kernel, we employ two input matrices and one output matrix as given in Figure 3.3. One CUDA thread is assigned to a combination in "combination matrix" that stores the match indices. The thread loads the corresponding matching pixels from "match matrix" and calculates the unknowns for that equation. Finally it stores the solution into the corresponding index of the "solution matrix".

There are $M = \binom{N}{2}$ number of solutions in the "solution matrix" as derived in Eq. (3.6). Since the calculations of a solution are independent, the threads are able to run in parallel and solve their equations simultaneously. Once all equations are solved, the best solution is determined based on a histogram structure, in which the final solution is selected as the most populated bin center.

Figure 3.4: Image registration for significant rotation shown in the bottom

3.4 Application to Real Image Registration

In the first experiment, we evaluated our method on real images. Since our registration algorithm calculates 4 transformation parameters (a rigid transform of images), real images can be problematic due to perspective or affine effect. For this reason, we captured images where there are only rotation, translation, and scale. In the first example, two images are registered where there is a significant rotation difference (about 45 degrees) and very minor translation and scale difference. Figure 3.4 shows the registration result.

In the second example, we captured two images that have significant scale and translation between each other. The results are given in Figure 3.5. Since the second image does not cover any new regions, we showed the final image with transparency effect so that the reader can see accuracy of the registration.

3.5 Application to Mosaic Image Generation

We also evaluated our method for mosaic image generation problem. A mosaic image is large composite view of an object or a scene, which is formed by a collection

Figure 3.5: Image registration for significant scale and translation difference with overlapping areas highlighted in the bottom image

of small images that partially covers the object [55]. Similar to the idea of "jigsaw" puzzles, every small image is placed to its correct position and orientation in the large mosaic image. This problem is very suitable application area to image registration studies since every small image must be registered to a single image in order to obtain the complete mosaic image. We apply our method on synthetic videos that are generated to check the correctness of mosaic generation using a set of camera motion patterns such as zigzag, spiral, zoom, rotate, or combined patterns presented in [56] and [55].

In this study, 3 synthetic video datasets (having 50 frames) are generated. Sizes of the original image and the frame (subimage) are 450×450 and 150×150, respectively. All 3 transformations (translation, rotation, and scale) are applied to subimages and then they are saved in the synthetic video. In Figure 3.6, the first column shows the original images, while the rest of images are 9 sample frames out of 50 in the video.

In the first dataset *(painting)*, we performed a composite transformation with significant amount of translation (0 to 400 pixels in x and y directions), rotation (0 to π), and scale (1 to 0.5). In the second dataset (*shuttle*), only translational transfor-

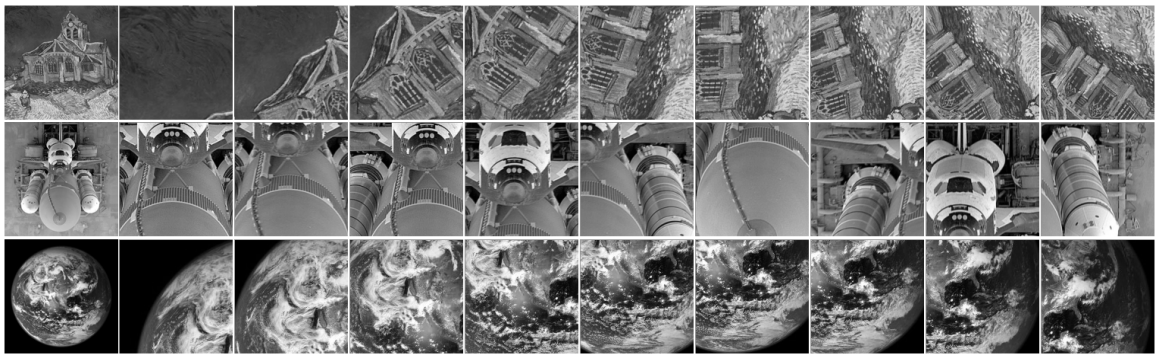


Figure 3.6: A subset of all frames in our 3 datasets. First frames of the rows represent the original image and rest of them are transformed frames for that dataset. In *painting* (first row), there is significant amount of rotation, translation and scale. In *shuttle* (second row), there is only translation. Finally in *earth* (third row), there is significant translation and scale applied to the frames.

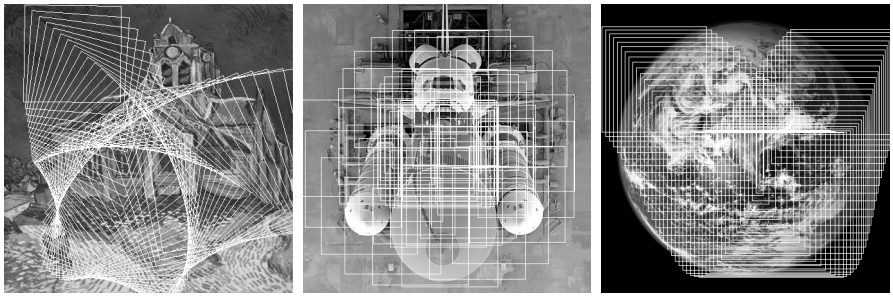
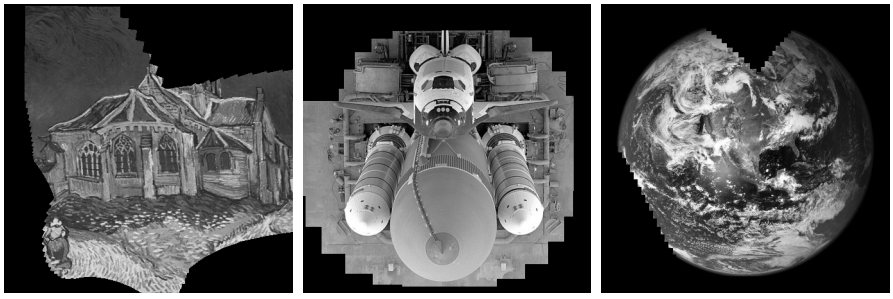


Figure 3.7: Generating video frames from 3 datasets. White rectangular windows represent transformed frames on the synthetic videos

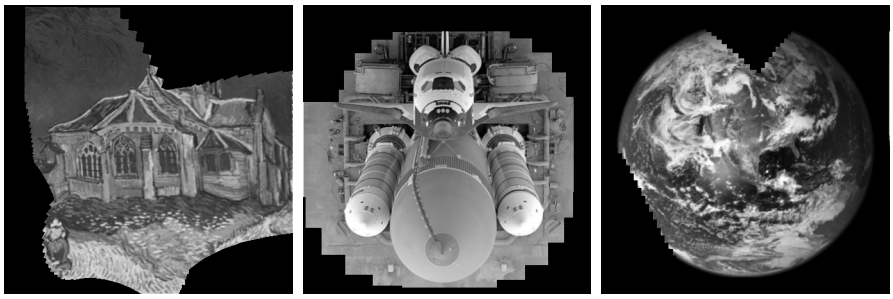
mations (-150 to 150 pixels in x and y directions) are applied with a circular motion on the original image. Finally, in the third dataset (*earth*), significant translational (0 to 300 pixels in x, -150 to 150 pixels in y direction) and scale (1 to 1.7) factors are applied. Every new frame is transformed with respect to its original position using these transformation parameters. Figure 3.7 shows the captured frames from the original image in white rectangular regions.

In order to generate mosaic image, consecutive video frames are processed in order. For the overlapping regions, a weighted pixel averaging strategy is preferred. To measure the registration accuracy, the mean of squared error (MSE) is calculated per parameter. Also in the end, the peak signal-to-noise ratio ($PSNR$) is measured between the original and the mosaic image. Registration results have been compared to another GPU based image registration method called *imregdemons* provided in MATLAB 2014b (M-GPU).

According to the results in Table 3.1, proposed method shows a significant improvement over M-GPU in the second dataset for all parameters. In the first dataset, a good amount of improvement is achieved for s and θ, and slightly better results than M-GPU for t_x and t_y. Finally in the third dataset, significant accuracy increase is achieved for t_x, t_y, and s. Please note that M-GPU was not able generate proper mosaic for this dataset, since it made a transformation error that propagated to the rest of the frames. Interestingly, both methods generate acceptable $PSNR$ values; however, there is a clear superiority of the proposed method over M-GPU on the second dataset. According to the visual results in Figure 3.8, no significant defect can be seen on the mosaic images. In such applications, it is often unavoidable to lose information (or quality) due to scaling up frames. Particularly, this problem can be seen in the central regions of *earth* dataset, where the frames need to be enlarged up to 1.7 times.



(a)



(b)

Figure 3.8: First row shows the ground truth images and second row shows the results of our algorithm.

Table 3.1: MSE and PSNR values and Improvement ratio of the proposed method

		e_{t_x}	e_{t_y}	e_s	e_θ	PSNR
Proposed Method	Painting	0.192	1.895	0.001	0.000	28.341
	Shuttle	0.079	0.034	0.000	0.008	27.630
	Earth	0.012	0.005	0.000	0.000	27.383
M-GPU	Painting	0.227	2.009	0.001	0.006	28.965
	Shuttle	0.777	0.168	0.015	0.035	22.984
	Earth	0.057	0.073	0.005	0.005	27.891
Imp(%)	Painting	15.1%	5.6%	92.2%	93.5%	N/A
	Shuttle	89.7%	79.31%	99.7%	97.9%	N/A
	Earth	79.2%	93.1%	97.6%	-6.9%	N/A

Table 3.2: Total processing times of the proposed method and M-GPU for varying image size (in seconds)

	10%	20%	40%	60%	80%	100%
Proposed*	0.043	0.044	0.048	0.053	0.061	0.072
M-GPU	1.727	2.208	3.998	6.842	10.402	15.213

* without feature extraction time.

3.6 Performance Evaluation

Processing Times: Total processing time of our algorithm depends on 2 stages: feature extraction and image transformation, while M-GPU uses a hierarchical registration method without a feature extraction stage. In other words, the complexity of our method depends on the number of features extracted, while the complexity of M-GPU depends on directly image size. To be able to compare two methods we generated an experiment where the processing time of two methods are measured by examining 10 different sizes of the same image pair from 326×245 (10%, 65 feature points) to 3264×2448 (100%, 1543 feature points). The results are shown in Table 3.2. Please note that our method requires feature extraction, which is not included in the table due the comparability reasons. Feature extraction can be done quite fast as presented in [57], in which it takes around 0.11 seconds to extract 2500 features from a 1080p video frame. Based on the results of this study, our method performs significantly faster than M-GPU, even if feature extraction is included in the timing.

CPU vs. GPU Implementation:We did our experiments in CPU and GPU to see how much speed-up we can reach with GPU programming. In both environ-

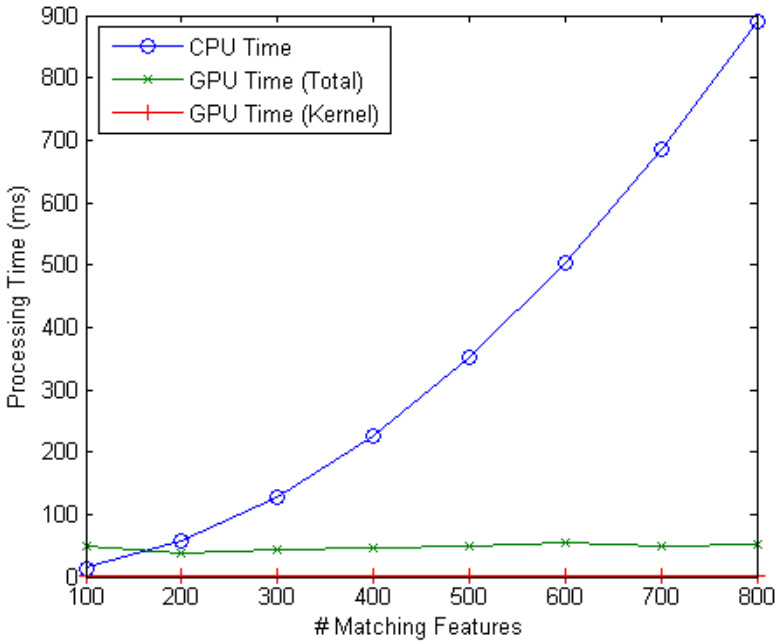


Figure 3.9: CPU vs GPU processing times

ments, we use the same source code. Only difference in GPU is that calculation of every combination is assigned to a single CUDA thread instead of using loop structure. First, data is transferred from RAM to GPU memory and then kernel function is executed. After the execution is done, results are sent back to the memory. We assigned 1 dimensional block of threads and 1 grid in the CUDA kernel. The block size is assigned to 256, which is a proper value for our hardware. In CUDA, we used "shared memory" and "tiling" techniques to reduce the total data transfer time. For the comparison, we measured 3 execution times: *CPU time (Total)*, *GPU time 1 (Total)*, and *GPU time 2 (Only Kernel Execution)*.

We ran the code on NVIDIA GeForce 320M graphics card and Intel Core 2 Duo 2.4 GHz CPU. Depending on the number of features, the time difference between

Table 3.3: CPU and GPU processing times depending on the number of features (in milliseconds)

#Features	CPU (ms)	GPU Total (ms)	GPU Kernel (ms)
100	13.926	48.02	0.045
200	56.261	38.355	0.049
300	127.992	44.607	0.066
400	226.608	45.184	0.084
500	352.415	49.598	0.09
600	504.607	53.366	0.071
700	686.567	48.19	0.077
800	891.314	51.646	0.071

GPU and CPU changes. Our experiments indicate that CPU runs faster than GPU up to 160 matching pixels. After that cross-over point, GPU becomes faster and the difference is getting larger as the number of matching pixels gets larger. Please note that most of the processing time on the GPU is the memory transfer time. Without memory transfer, computation takes less than 1 millisecond, while memory transfer time is around 50 milliseconds on the average. In Figure 3.9, it can be seen that number of matches has no significant effect on the GPU performance. Numerical results of the same experiments can be seen in Table 3.3.

3.7 Summary

This chapter presents a GPU based image registration method to solve feature mapping problem in pose estimation. Proposed registration technique relies on the idea of Hough transform to calculate the transformation. Massively parallel execution capability of GPU is exploited to solve the combinations of matching image pixels extracted by SIFT descriptor.

Our algorithm is evaluated on real and synthetic images. Experimental results show that our method is robust to the outliers (incorrect matches) and it can achieve very accurate registration (numeric and visual) results with much faster (up to 20 times) than CPU implementation.

The complexity of our method depends on the number of features and possible combinations. For affine and perspective motion, the number of features and combinations may be reduced. For optimization algorithms, our four parameters may act as initial values before computing affine and perspective motion parameters.

CHAPTER 4

MIRAGE POSE ESTIMATION

Traditional techniques use direct 2D-3D correspondence of the feature points to calculate the (actual) camera pose. In this dissertation, we propose Mirage pose estimation method, in which an indirect approach is preferred by including an additional desired camera pose. A "desired pose" is an imaginary pose that is used as a "reference" to localize the camera in 3D space. It is commonly used in trajectory tracking applications to navigate the vehicle towards a particular (desired) position. In Mirage, we utilize this idea for pose estimation problem. Using the desired camera pose, the 3D feature points are projected into a desired image plane, and desired pixel coordinates of the feature points are generated [3]. Our approach takes advantage of the incremental relations [58] between 3D pose of the camera and the pixel coordinates of the feature points. This idea allows us to estimate the relation between the actual pose and the desired pose by minimizing the error between actual and desired pixel coordinates. Mirage, uses these pixel errors (incremental pixel coordinates) to calculate how the camera is located with respect to its desired pose (incremental 3D pose). This gives us the "pose error" or "relative pose" between the desired and actual poses. Finally, the actual pose can be calculated by adding the computed pose error

to the desired pose. However, computing the relative pose rather than the actual pose is advantageous in many robotic systems, since only the relative pose (or pose error) would be sufficient to follow a desired trajectory or complete a given task.

Our solution involves analytical derivations, which finally yield a linear equation system to estimate the pose. For multi-camera systems, this low complexity allows us to achieve a linear time ($O(n)$) solution that is beneficial for real time systems. Moreover, the extension procedure does not add a significant overhead into the system, while improving its reliability as the field-of-view is extended.

4.1 Single-Camera Systems

Mirage analytically solves 6 pose parameters in $O(n)$ time on multi-camera systems. However, in this subsection, we describe the mathematical derivations of our technique for a single camera case for the sake of simplicity. Figure 4.1 illustrates a sample scenario showing world (I), vehicle (B), camera (C) spaces, and a "reference (target) object" having 4 3D feature points (r^B) in the actual vehicle space. Similarly, desired vehicle space (Bd) and corresponding desired camera space (Cd) are given along with the 3D feature points in the *desired* vehicle space (r^{Bd}).

The actual pose is denoted by $\mathbf{q} = [x, y, z, \theta, \phi, \psi]$ and shown by solid lines in the figure. Similarly, the desired pose is denoted by $\mathbf{q^d} = [x^d, y^d, z^d, \theta^d, \phi^d, \psi^d]$ and shown by dashed lines. Mirage calculates the relative pose by determining a 4×4 transformation matrix $\tilde{\mathbf{T}}^B_{Bd}$, which comprise of rotational and translational parts such as $\tilde{\mathbf{T}}^B_{Bd} = [\mathbf{R}^B_{Bd}|\mathbf{t}^B_{Bd}]$, where $\mathbf{R} \in SO(3)$ and $\mathbf{t} \in R^3$.

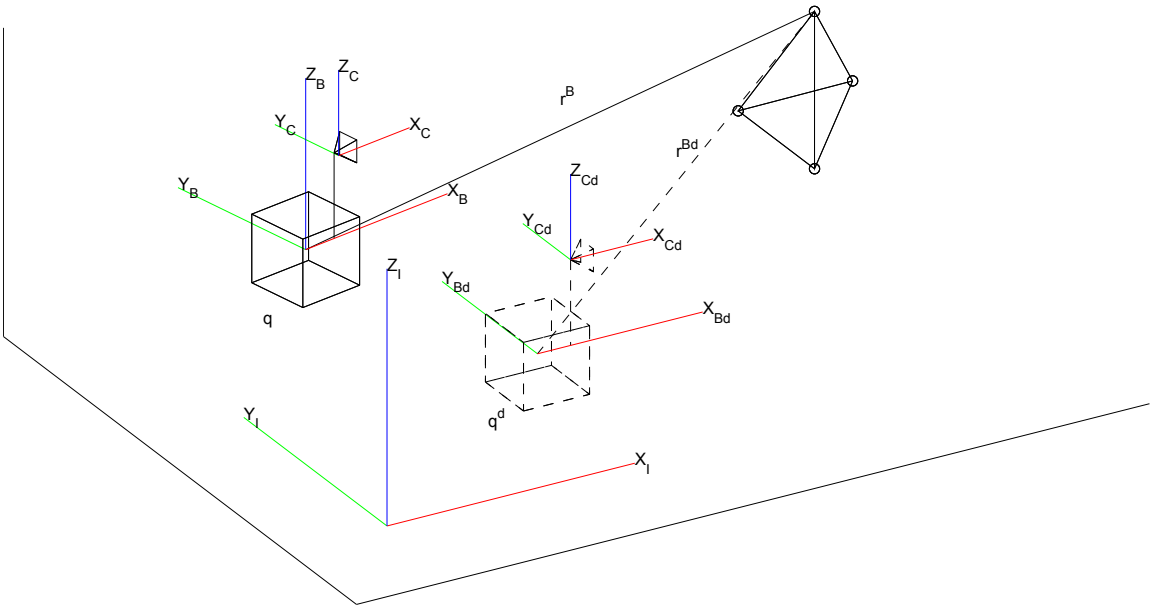


Figure 4.1: Sample Scene for Mirage. $\mathbf{q}$ is the actual pose of the vehicle and $\mathbf{r}^B$ is 3D position of a feature point with respect to the actual pose. Similarly, $\mathbf{q}^d$ is the desired pose of the vehicle and $\mathbf{r}^{Bd}$ is 3D position of the same feature point with respect to the desired pose. Mirage calculates the relative pose of the vehicle with respect to its desired position ($\tilde{\mathbf{T}}^B_{Bd}$). The desired pose $\mathbf{q}^d$, is determined beforehand, and actual pose of the vehicle $\mathbf{q}$ is calculated. I, B, and C represent *world*, *vehicle*, and *camera* spaces, respectively.

$\tilde{\mathbf{T}}^B_{Bd}$ transforms the vehicle's local frame at its actual pose (X_B, Y_B, Z_B) to its local frame at its desired pose X_{Bd}, Y_{Bd}, Z_{Bd}. This relation can be represented such that

$$\mathbf{T}^B_I = \tilde{\mathbf{T}}^B_{Bd}\mathbf{T}^{Bd}_I \tag{4.1}$$

where $\mathbf{T}_I^B = [\mathbf{R}_I^B|\mathbf{t}_I^B]$ is a transformation matrix that transform a given coordinate in world space into the actual vehicle space. Actual pose of the vehicle, $\mathbf{q}$, can be determined by manipulating rotation and translational parts of this matrix. Similarly, $\mathbf{T}_I^{Bd} = [\mathbf{R}_I^{Bd}|\mathbf{t}_I^{Bd}]$ is the transformation matrix to transform a given coordinate in world space into the desired vehicle space and it corresponds to $\mathbf{q^d}$, the desired pose of the vehicle. Eq. (4.1) states that the actual pose is conveniently determined if $\mathbf{T}_I^{Bd}$ is known and $\tilde{\mathbf{T}}_{Bd}^B$ is calculated.

Mirage solves $\tilde{\mathbf{T}}_{Bd}^B$ by minimizing the projection errors of 3D feature points. Assume that $\mathbf{r}^B = [r_1^B\ r_2^B\ r_3^B\ 1]^T$ is the 3D position of the object feature point with respect to the vehicle at its actual position. Similarly, $\mathbf{r}^{Bd} = [r_1^{Bd}\ r_2^{Bd}\ r_3^{Bd}\ 1]^T$ indicates the relative position of the same point when the vehicle is at its desired position. $\mathbf{r}^B$ is equal to $\mathbf{r}^{Bd}$ when the vehicle is at its desired pose. Otherwise a 3D translation and rotation is necessary for the vehicle to transform the desired points to the actual points. This transformation is done by $\tilde{\mathbf{T}}_{Bd}^B$ transformation matrix as

$$\mathbf{r}^B = \tilde{\mathbf{T}}_{Bd}^B \mathbf{r}^{Bd} \tag{4.2}$$

Let $\mathbf{p}^C$ be a 3D point in camera space and $\mathbf{M}^C$ be a 4×4 transformation matrix that transforms a point from the vehicle space to the camera space,

$$\mathbf{p}^C = \mathbf{M}^C \mathbf{r}^B \tag{4.3}$$

where $\mathbf{M}^C = [\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3\ 1]^T$ and $\mathbf{m}_i$'s are the rows of the matrix. $\mathbf{M}^C$ comprises two different transformation matrices $\mathbf{K}^C$ and $\mathbf{T}_B^C$,

$$\mathbf{M}^C = \mathbf{K}^C \mathbf{T}_B^C \tag{4.4}$$

where $\mathbf{T}_B^C$ is a 4×4 transformation matrix including extrinsic parameters of the camera that transforms a point from the vehicle space to the camera space. $\mathbf{K}^C$ is also a 4×4 matrix of intrinsic camera parameters (focal length, principle points and distortion parameters), which are available a priori via camera calibration. $\mathbf{K}^C$ projects the point in camera space into image plane coordinates. Similar to Eq. (4.3), desired points can be transformed from vehicle to camera space as follows

$$\mathbf{p}^{Cd} = \mathbf{M}^C \mathbf{r}^{Bd} \tag{4.5}$$

The error between points can be measured using the desired and actual 3D points in camera space. Normally, a simple subtraction operation $\mathbf{p}^C - \mathbf{p}^{Cd}$ would be performed to calculate 3D point error. However, our target is to obtain the projection error between points. So, the error can be derived as

$$\tilde{\mathbf{p}}^C = \begin{bmatrix} \tilde{p}_1^C \\ \tilde{p}_2^C \\ \tilde{p}_3^C \end{bmatrix} = \begin{bmatrix} p_3^{Cd} p_1^C - p_1^{Cd} p_3^C \\ p_3^{Cd} p_2^C - p_2^{Cd} p_3^C \\ p_3^{Cd} p_3^C \end{bmatrix} \tag{4.6}$$

where $\tilde{\mathbf{p}}^C$ represents the error between actual and desired values of the target points in the camera space. Expression in Eq. (4.6) is substituted with corresponding expressions in Eq. (4.3) and Eq. (4.5) as follows

$$\tilde{\mathbf{p}}^C = \begin{bmatrix} \tilde{p}_1^C \\ \tilde{p}_2^C \\ \tilde{p}_3^C \end{bmatrix} = \begin{bmatrix} \mathbf{m}_3\mathbf{r}^{Bd}\mathbf{m}_1\mathbf{r}^B - \mathbf{m}_1\mathbf{r}^{Bd}\mathbf{m}_3\mathbf{r}^B \\ \mathbf{m}_3\mathbf{r}^{Bd}\mathbf{m}_2\mathbf{r}^B - \mathbf{m}_2\mathbf{r}^{Bd}\mathbf{m}_3\mathbf{r}^B \\ \mathbf{m}_3\mathbf{r}^{Bd}\mathbf{m}_3\mathbf{r}^B \end{bmatrix} \tag{4.7}$$

The common factor $\mathbf{r}^B$ is moved out of the parenthesis and Eq. (4.8) is obtained.

$$\tilde{\mathbf{p}}^C = \begin{bmatrix} \tilde{p}_1^C \\ \tilde{p}_2^C \\ \tilde{p}_3^C \end{bmatrix} = \begin{bmatrix} \mathbf{m}_3\mathbf{r}^{Bd}\mathbf{m}_1 - \mathbf{m}_1\mathbf{r}^{Bd}\mathbf{m}_3 \\ \mathbf{m}_3\mathbf{r}^{Bd}\mathbf{m}_2 - \mathbf{m}_2\mathbf{r}^{Bd}\mathbf{m}_3 \\ \mathbf{m}_3\mathbf{r}^{Bd}\mathbf{m}_3 \end{bmatrix} \mathbf{r}^B \tag{4.8}$$

For the sake of simplicity, the rows of Eq. (4.8) are called $\mathbf{n}_i$. Using this notation, following form of the equation is obtained.

$$\tilde{\mathbf{p}}^C = \begin{bmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \mathbf{n}_3 \end{bmatrix} \mathbf{r}^B = \mathbf{N}^C\mathbf{r}^B \tag{4.9}$$

After substitution of $\mathbf{r}^B$ with $\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd}$ in Eq. (4.2), Eq. (4.10) can be derived.

$$\tilde{\mathbf{p}}^C = \mathbf{N}^C\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd} \tag{4.10}$$

Equation (4.10) implies that $\tilde{\mathbf{T}}_{Bd}^B$ can analytically be calculated if $\tilde{\mathbf{p}}^C$ is known, since all other parameters in the equation are known. $\tilde{\mathbf{p}}^C$ can be calculated indirectly

by comparing the desired and actual pixel coordinates. Because 2D image errors can be related to $\tilde{\mathbf{p}}^C$. The relation between 2D and 3D image errors is shown as

$$\mathbf{e}^C = \begin{bmatrix} e_x \\ e_y \end{bmatrix} = \begin{bmatrix} \tilde{p}_1^C/\tilde{p}_3^C \\ \tilde{p}_2^C/\tilde{p}_3^C \end{bmatrix} \tag{4.11}$$

where $\mathbf{e}^C$ represents the error between actual and desired 2D pixel coordinates of the points. In this case, $\mathbf{e}^C$ is a known vector because both the desired and actual images are known. If we separate the errors for both x and y dimensions on the image, we obtain

$$\tilde{p}_3^C e_x - \tilde{p}_1^C = 0 \tag{4.12}$$

$$\tilde{p}_3^C e_y - \tilde{p}_2^C = 0 \tag{4.13}$$

If we substitute $\tilde{p}_i^C$ with the corresponding value in Eq. (4.10), the new set of error equations, (4.14) and (4.15) are obtained.

$$\mathbf{n}_3 \tilde{\mathbf{T}}_{Bd}^B \mathbf{r}^{Bd} e_x - \mathbf{n}_1 \tilde{\mathbf{T}}_{Bd}^B \mathbf{r}^{Bd} = 0 \tag{4.14}$$

$$\mathbf{n}_3 \tilde{\mathbf{T}}_{Bd}^B \mathbf{r}^{Bd} e_y - \mathbf{n}_2 \tilde{\mathbf{T}}_{Bd}^B \mathbf{r}^{Bd} = 0 \tag{4.15}$$

We will consider derivations of only x component in the following parts of this section. Derivations regarding y component will be provided as needed. Observe that

Eq. (4.14) and Eq. (4.15) have multiplication with $\mathbf{n}_1^T$, $\mathbf{n}_2^T$, and $\mathbf{n}_3^T$ vectors, which are the components of $\mathbf{N}^C$ as mentioned in Eq. (4.9). Expanded form of $\mathbf{N}^C$ is shown below.

$$\mathbf{N}^C = \begin{bmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \mathbf{n}_3 \end{bmatrix} = \begin{bmatrix} n_{11} & n_{12} & n_{13} & n_{14} \\ n_{21} & n_{22} & n_{23} & n_{24} \\ n_{31} & n_{32} & n_{33} & n_{34} \end{bmatrix} \tag{4.16}$$

By substituting $\mathbf{n}_3$ in the first term of Eq. (4.14) with the 3$^{\text{rd}}$ row of $\mathbf{N}^C$, we obtain

$$\mathbf{n}_3\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd}e_x = [n_{31}\ n_{32}\ n_{33}\ n_{34}] \left[\begin{array}{cccc} \tilde{\mathbf{t}}_1 & \tilde{\mathbf{t}}_2 & \tilde{\mathbf{t}}_3 & \tilde{\mathbf{t}}_4 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \mathbf{r}^{Bd}e_x \tag{4.17}$$

where $\tilde{\mathbf{t}}_i$ represents $[\tilde{t}_{1i}\ \tilde{t}_{2i}\ \tilde{t}_{3i}]^T$ vector of $\tilde{\mathbf{T}}_{Bd}^B$. If we separate n_{34} from the other n_{ij} values, we get

$$\mathbf{n}_3\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd}e_x = [n_{31}\ n_{32}\ n_{33}\ 0]\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd}e_x + [0\ 0\ 0\ n_{34}]\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd}e_x \tag{4.18}$$

The second term can be simplified as

$$n_{34}e_x = [0\ 0\ 0\ n_{34}]\tilde{\mathbf{T}}_{Bd}^B\mathbf{r}^{Bd}e_x \tag{4.19}$$

By substituting new value of the term in Eq. (4.18), we obtain Eq. (4.20)

$$\mathbf{n}_3\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd}e_x = [n_{31}\ n_{32}\ n_{33}\ 0]\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd}e_x + n_{34}e_x \tag{4.20}$$

Then, the terms with the matrix $\tilde{\mathbf{T}}^B_{Bd}$ is moved to the same side of equation yielding

$$n_{34}e_x = \mathbf{n}_1\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd} - [n_{31}\ n_{32}\ n_{33}\ 0]\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd}e_x \tag{4.21}$$

Now, we can divide each side of the equation by n_{34} to isolate e_x

$$e_x = \frac{\mathbf{n}_1}{n_{34}}\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd} - \frac{[n_{31}\ n_{32}\ n_{33}\ 0]}{n_{34}}\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd}e_x \tag{4.22}$$

For further simplification, assume

$$\mathbf{o}_1 = \frac{\mathbf{n}_1}{n_{34}}, \tag{4.23}$$

$$\mathbf{o}_2 = \frac{\mathbf{n}_2}{n_{34}}, \tag{4.24}$$

$$\mathbf{o}_3 = \frac{[n_{31}\ n_{32}\ n_{33}\ 0]}{n_{34}} \tag{4.25}$$

where $\mathbf{o}_i$ is a 1×4 vector. And, by substituting the $\mathbf{o}_1$ and $\mathbf{o}_3$ values in Eq. (4.23) and Eq. (4.25) with the corresponding terms in Eq. (4.22), we obtain

$$e_x = \mathbf{o}_1\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd} - \mathbf{o}_3\tilde{\mathbf{T}}^B_{Bd}\mathbf{r}^{Bd}e_x \tag{4.26}$$

Similarly, the error in y component may be presented using the same derivations,

$$e_y = \mathbf{o}_2 \tilde{\mathbf{T}}^B_{Bd} \mathbf{r}^{Bd} - \mathbf{o}_3 \tilde{\mathbf{T}}^B_{Bd} \mathbf{r}^{Bd} e_y \tag{4.27}$$

Now, if we multiply $\mathbf{o}_i$ vectors with the matrix $\tilde{\mathbf{T}}^B_{Bd}$, we obtain

$$e_x = \begin{bmatrix} \mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_1 \\ \mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_2 \\ \mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_3 \\ \mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_4 + o_{14} \end{bmatrix}^T \begin{bmatrix} r_1^{Bd} \\ r_2^{Bd} \\ r_3^{Bd} \\ 1 \end{bmatrix} - e_x \begin{bmatrix} \mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_1 \\ \mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_2 \\ \mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_3 \\ \mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_4 \end{bmatrix}^T \begin{bmatrix} r_1^{Bd} \\ r_2^{Bd} \\ r_3^{Bd} \\ 1 \end{bmatrix} \tag{4.28}$$

Similarly, after the multiplication with the $\mathbf{r}^{Bd}$, we obtain

$$\begin{aligned} e_x =& (r_1^{Bd}\mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_1 + r_2^{Bd}\mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_2 \\ &+ r_3^{Bd}\mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_3 + \mathbf{o}_{1,(1\to3)}\tilde{\mathbf{t}}_4 + o_{14}) \\ &- e_x(r_1^{Bd}\mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_1 + r_2^{Bd}\mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_2 \\ &+ r_3^{Bd}\mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_3 + \mathbf{o}_{3,(1\to3)}\tilde{\mathbf{t}}_4) \end{aligned} \tag{4.29}$$

Now, we can reorder the terms of Eq. (4.29) and then group the common terms as follows:

$$
\begin{aligned}
e_x - o_{14} =& \Big(r_1^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_1^{Bd}\mathbf{o}_{3,(1\to3)} \Big)\tilde{\mathbf{t}}_1 \\
&+ \Big(r_2^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_2^{Bd}\mathbf{o}_{3,(1\to3)} \Big)\tilde{\mathbf{t}}_2 \\
&+ \Big(r_3^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_3^{Bd}\mathbf{o}_{3,(1\to3)} \Big)\tilde{\mathbf{t}}_3 \\
&+ \Big(\mathbf{o}_{1,(1\to3)} - e_x \mathbf{o}_{3,(1\to3)} \Big)\tilde{\mathbf{t}}_4
\end{aligned}
\tag{4.30}
$$

This form of Eq. (4.30) enables us to rewrite $\vec{r_i}$ and $\tilde{\mathbf{t}}_4$ parameters as vector format which is actually equal to $\tilde{\mathbf{t}}_{Bd}^{B}$ as mentioned in Eq. (4.37). In this way, unknown parameters of matrix $\tilde{\mathbf{t}}_{Bd}^{B}$ is separated from known variables as

$$
e_x - o_{14} = \begin{bmatrix}
r_1^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_1^{Bd}\mathbf{o}_{3,(1\to3)} \\
r_2^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_2^{Bd}\mathbf{o}_{3,(1\to3)} \\
r_3^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_3^{Bd}\mathbf{o}_{3,(1\to3)} \\
\mathbf{o}_{1,(1\to3)} - e_x \mathbf{o}_{3,(1\to3)}
\end{bmatrix}^T
\begin{bmatrix}
\tilde{\mathbf{t}}_1 \\ \tilde{\mathbf{t}}_2 \\ \tilde{\mathbf{t}}_3 \\ \tilde{\mathbf{t}}_4
\end{bmatrix}
\tag{4.31}
$$

In a similar way, the equation regarding y component can be written as

$$
e_y - o_{24} = \begin{bmatrix}
r_1^{Bd}\mathbf{o}_{2,(1\to3)} - e_y r_1^{Bd}\mathbf{o}_{3,(1\to3)} \\
r_2^{Bd}\mathbf{o}_{2,(1\to3)} - e_y r_2^{Bd}\mathbf{o}_{3,(1\to3)} \\
r_3^{Bd}\mathbf{o}_{2,(1\to3)} - e_y r_3^{Bd}\mathbf{o}_{3,(1\to3)} \\
\mathbf{o}_{2,(1\to3)} - e_y \mathbf{o}_{3,(1\to3)}
\end{bmatrix}^T
\begin{bmatrix}
\tilde{\mathbf{t}}_1 \\ \tilde{\mathbf{t}}_2 \\ \tilde{\mathbf{t}}_3 \\ \tilde{\mathbf{t}}_4
\end{bmatrix}
\tag{4.32}
$$

In this form of equation, we clearly isolated unknowns from the known values. In order to simplify the notation, let $\mathbf{v}_x$ and $\mathbf{v}_y$ be 1×12 vectors which composes of known values in the right side of Eq. (4.31) and Eq. (4.32)

$$\mathbf{v}_x = \begin{bmatrix} r_1^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_1^{Bd}\mathbf{o}_{3,(1\to3)} \\ r_2^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_2^{Bd}\mathbf{o}_{3,(1\to3)} \\ r_3^{Bd}\mathbf{o}_{1,(1\to3)} - e_x r_3^{Bd}\mathbf{o}_{3,(1\to3)} \\ \mathbf{o}_{1,(1\to3)} - e_x\mathbf{o}_{3,(1\to3)} \end{bmatrix}^T \tag{4.33}$$

$$\mathbf{v}_y = \begin{bmatrix} r_1^{Bd}\mathbf{o}_{2,(1\to3)} - e_y r_1^{Bd}\mathbf{o}_{3,(1\to3)} \\ r_2^{Bd}\mathbf{o}_{2,(1\to3)} - e_y r_2^{Bd}\mathbf{o}_{3,(1\to3)} \\ r_3^{Bd}\mathbf{o}_{2,(1\to3)} - e_y r_3^{Bd}\mathbf{o}_{3,(1\to3)} \\ \mathbf{o}_{2,(1\to3)} - e_y\mathbf{o}_{3,(1\to3)} \end{bmatrix}^T \tag{4.34}$$

By substituting $\mathbf{v}_x$ and $\mathbf{v}_y$ values with corresponding fields in Eq. (4.31) and Eq. (4.32), we reach the final representations of error in x and y components

$$e_x - o_{14} = \mathbf{v}_x \tilde{\mathbf{t}}_{Bd}^{B} \tag{4.35}$$

$$e_y - o_{24} = \mathbf{v}_y \tilde{\mathbf{t}}_{Bd}^{B} \tag{4.36}$$

Consequently, the combination of these 2 equations gives us Eq. (4.37).

$$\bar{\mathbf{e}}^C = \mathbf{V}^C \tilde{\mathbf{t}}^B_{Bd} \tag{4.37}$$

where

$$\bar{\mathbf{e}}^C = \begin{bmatrix} e_x - o_{14} \\ \\ e_y - o_{24} \end{bmatrix} \tag{4.38}$$

In Eq. (4.37), $\bar{\mathbf{e}}^C$ refers to pixel errors between actual and desired image pixels. $\tilde{\mathbf{t}}^B_{Bd} = [\tilde{t}_{11}\ \tilde{t}_{12}\ \tilde{t}_{13}\ ...\ \tilde{t}_{33}\ \tilde{t}_{34}]^T$ is a 12×1 vector that is actually the reshaped form of $\tilde{\mathbf{T}}^B_{Bd}$ matrix. And finally, $\mathbf{V}^C$ is 2×12 matrix that satisfies the corresponding equations. Eq. (4.37) is a linear equation system. 12 unknowns in $\tilde{\mathbf{t}}^B_{Bd}$ can be solved if there are at least 12 independent equations in the system. In a single camera system, one target point generates two equations since one equation exists per image dimension x and y as in Eq. (4.11). Therefore, in theory, a minimum of 6 distinctive target points are necessary in a single camera system to satisfy all 12 equations. However, due to calibration error and dimensional inaccuracies, 12 equations do not lead to the correct result using single camera. Adding extra equations ($\mathbf{t}_i\mathbf{t}_j = 0$, for $1 \leq i, j \leq 3$, $i \neq j$) or camera(s) is necessary to compensate these errors. Finally n target points generate $2n$ different equations where $2n > 12$. Eq. (4.37) is stacked n times to obtain:

$$\begin{bmatrix} \mathbf{V}_1^C \\ \mathbf{V}_2^C \\ \vdots \\ \mathbf{V}_n^C \end{bmatrix} \tilde{\mathbf{t}}_{Bd}^B = \begin{bmatrix} \bar{\mathbf{e}}_1^C \\ \bar{\mathbf{e}}_2^C \\ \vdots \\ \bar{\mathbf{e}}_n^C \end{bmatrix} \tag{4.39}$$

In a real application, it is most likely to have a large number of matching points, which may also have some amount of noise. For such cases, direct solutions are not applicable since the number of rows of $\mathbf{V}^C$ matrix will be much larger than 12. To overcome this limitation, a "least squares" optimization method is beneficial. Let us rewrite Eq. (4.39) such that $\mathbf{Q}$ be a $2n \times 12$ coefficient matrix on the left hand side and $\mathbf{W}$ be $2n \times 1$ constant vector in the right hand side,

$$\mathbf{Q}\tilde{\mathbf{t}}_{Bd}^B = \mathbf{W} \tag{4.40}$$

where symbols $\mathbf{Q}$ and $\mathbf{W}$ refer to corresponding matrices in Eq. (4.39). Considering large number of points, $\mathbf{Q}$ will not be a square matrix, thus Eq. (4.40) may be solved using the least squares method. The error,

$$E = \frac{1}{2}(\mathbf{Q}\tilde{\mathbf{t}}_{Bd}^B - \mathbf{W})^T(\mathbf{Q}\tilde{\mathbf{t}}_{Bd}^B - \mathbf{W}), \tag{4.41}$$

is minimized using the formula in Eq. (4.42), which also solves the unknowns in $\tilde{\mathbf{t}}_{Bd}^B$ vector.

$$\tilde{\mathbf{t}}_{Bd}^B = (\mathbf{Q}^T\mathbf{Q})^{-1}(\mathbf{Q}^T\mathbf{W}) \tag{4.42}$$

Note that $\tilde{\mathbf{t}}_{Bd}^{B}$ includes 12 elements of the transformation matrix $\tilde{\mathbf{T}}_{Bd}^{B}$. If the purpose of the application was to estimate the relative pose of the vehicle, then using directly $\tilde{\mathbf{T}}_{Bd}^{B}$ would be sufficient, but we seek the actual pose of the vehicle. Thus additional operation is necessary such that

$$\mathbf{T}_{I}^{B} = \tilde{\mathbf{T}}_{Bd}^{B}\mathbf{T}_{I}^{Bd} \tag{4.43}$$

which was described in Eq. (4.1) earlier. The multiplication in Eq. (4.43) returns the actual pose parameters of the vehicle as a transformation matrix,

$$\mathbf{T}_{I}^{B} = \left[\begin{array}{ccc|c} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.44}$$

At this stage, a conversion is necessary to calculate the actual pose $\mathbf{q} = [x, y, z, \theta, \phi, \psi]$ using 12 elements of $\mathbf{T}_{I}^{B}$ matrix. The values t_{14}, t_{24}, and t_{34} are conveniently assigned to translation parameters x, y, and z, respectively. For the rotational parameters, the following conversions are necessary to compute the angles. Note that in this conversion, the quadrant-dependent function *atan2* is used instead of a simple *arctangent* function. The sign of $\cos(\theta)$ determines the quadrant of ϕ and ψ, so it must not be dropped from the denominators.

$$\theta = \text{atan2}(-t_{31}, \sqrt{{t_{11}}^2 + {t_{21}}^2}) \tag{4.45}$$

$$\phi = \text{atan2}(\frac{t_{32}}{\cos\theta}, \frac{t_{33}}{\cos\theta}) \tag{4.46}$$

$$\psi = \text{atan2}(\frac{t_{21}}{\cos\theta}, \frac{t_{11}}{\cos\theta}) \tag{4.47}$$

4.2 Generalization to Multi-Camera Systems

Single camera case is a special condition requiring additional equations into the system as described above. However, our method can easily be adapted to multi-camera systems without the need of major modifications and significant overhead (e.g., bundle adjustment [59]). Since it requires fixed number of equations, it is not crucial how many cameras are employed. Moreover, our derivations yield a significant advantage, in which increasing the number of cameras decreases the number of target points required for the solution. Based on the fact that 1 camera can provide 2 equations from a target point, theoretically, the inequality $mn \geq 6$ must be satisfied, where m is the number of cameras and n is the number of target points. However, in practice at least 4 non-planar points are necessary to localize the vehicle and perform reliable pose calculation.

Our system treats the multi-camera system as "one" [24]. However, there are no major restrictions on the setup and specifications of cameras. The cameras do not have to be identical, and they can be installed on the vehicle at known arbitrary poses. All cameras can capture different target points, and non-overlapping setup as in [24] is not required for Mirage. Mirage allows overlapping cameras and can benefit

from overlaps by reducing the number of features per camera. If n target points are required for a single camera system, multi-camera system with m cameras may work with $\lceil n/m \rceil$ target points. Equation (4.39) formulates this logic for a single camera system. If we generalize this formula for m number of cameras, we have

$$\begin{bmatrix} \mathbf{V}_1^{C_1} \\ \mathbf{V}_1^{C_2} \\ \vdots \\ \mathbf{V}_1^{C_m} \\ \mathbf{V}_2^{C_1} \\ \mathbf{V}_2^{C_2} \\ \vdots \\ \mathbf{V}_2^{C_m} \\ \vdots \\ \mathbf{V}_n^{C_1} \\ \mathbf{V}_n^{C_2} \\ \vdots \\ \mathbf{V}_n^{C_m} \end{bmatrix} \tilde{\mathbf{t}}_{Bd}^{B} = \begin{bmatrix} \bar{\mathbf{e}}_1^{C_1} \\ \bar{\mathbf{e}}_1^{C_2} \\ \vdots \\ \bar{\mathbf{e}}_1^{C_m} \\ \bar{\mathbf{e}}_2^{C_1} \\ \bar{\mathbf{e}}_2^{C_2} \\ \vdots \\ \bar{\mathbf{e}}_2^{C_m} \\ \vdots \\ \bar{\mathbf{e}}_n^{C_1} \\ \bar{\mathbf{e}}_n^{C_2} \\ \vdots \\ \bar{\mathbf{e}}_n^{C_m} \end{bmatrix} \tag{4.48}$$

where $\mathbf{V}_j^{C_i}$ represents the $\mathbf{V}$ matrix for i^{th} camera and j^{th} target point. Similarly, symbol $\bar{\mathbf{e}}_j^{C_i}$ represents $\bar{\mathbf{e}}$ pixel errors for i^{th} camera and j^{th} target point. Using the equation system in Eq. (4.48) instead of Eq. (4.39), the pose parameters can be conveniently solved for a multi-camera system having m cameras.

4.3 Performance of Mirage on Synthetic Data

In this section, we investigate the pose calculation accuracy and processing time of Mirage by comparing with state-of-the-art in the literature. In all experiments, we used the intrinsic parameters of a real camera (Logitech HD Webcam C525) and image size of 640×480 pixels. Our code is implemented in MATLAB 2013a.

In the synthetic experiments, the actual camera is positioned on a pre-defined pose in the world space and n number of feature (target) points are randomly generated (in the interval of [-5,5] in x, y, and z dimensions). Then, 2D projection of each point is calculated. 3D points and their projections are provided to all methods, and their results (6 pose parameters) are compared to the real pose parameters. Only in Mirage, a desired pose is also defined to be used in calculations.

Since we generate the points randomly, we run every simulation 20 times and get the average error for the final evaluation. We compare Mirage with available state-of-the-art techniques, Efficient PnP (EPnP), Efficient PnP with Gauss Newton (EPnP-G), Classic POSIT (Posit-C), Modern POSIT [1] (Posit-M), Direct Linear Transform (DLT), Lu-Hager-Mjolsness (LHM), Robust PnP (RPnP), Direct Least Squares (DLS). Each method is evaluated with respect to processing time, robustness to noise and number of feature points via three experiments: noise rate vs. pose error, the number of points vs. pose error, and the number of points vs. processing time.

[1] Projection center is adjusted to object origin

4.3.1 Noise Rate vs. Pose Error

In real applications, it is a common problem to have fair amount of noise on feature extraction stage. A desirable pose estimation method should be able to handle this problem without sacrificing the accuracy significantly. The first experiment targets to simulate this condition, where 2D pixel coordinates cannot be perfectly detected. The results show the noise resilience analysis of each method. In the experiments, the number of feature points n is set to 50, and 20 different gaussian noise rate ($\sigma = 0$ to 20) is added to the pixel coordinates of each point. The noise is generated randomly for each feature point and varies. In other words, these experiments do not measure a static system error. Figure 4.2 shows translational (cm) and rotational (radian) pose errors with respect to the image pixel noise.

According to the results, DLT and Posit-C return very high error rates, which imply they are very sensitive to the noise. Particularly DLT is very unstable with respect to the noise. LHM and DLS give low rotational errors but their translational error rapidly increases as the noise increases. EPnP and EPnP-G generate acceptable results, yet a few spikes can be seen around $\sigma = 12$, 16, and 20, which make them unstable on these cases. RPnP yields relatively better results in translation but is still not as good as the Posit-M and Mirage in rotation. Interestingly, an iterative approach Posit-M generates low error rate and more stable results than other compared methods in both translational and rotational parameters. However, Posit-M has another limitation that is not shown in the figure. During our experiments, on some specific cases (special combinations of random points), Posit methods have fallen to

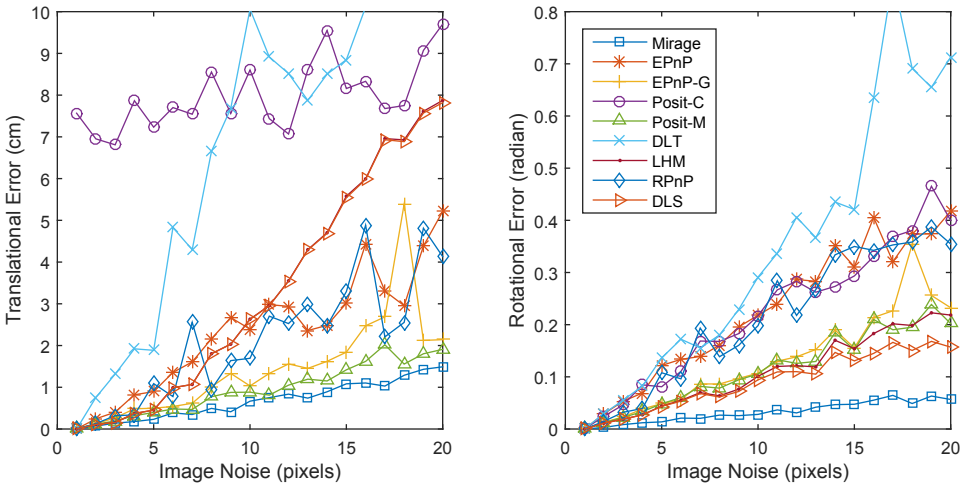


Figure 4.2: Translational and rotational pose error with respect to the pixel noise for $n = 50$ feature points and $\sigma = 0$ to 20 noise rate

local minima and never converged to a solution. This is a critical limitation that makes most of the iterative methods impractical. Our method, on the other hand, yields stable results and the lowest translational and rotational errors.

4.3.2 Number of Points vs. Pose Error

Number of feature points is another critical factor in pose estimation problem. It is desirable to estimate the pose with less number of points. In the second experiment, we analyze the effect of number of feature points to the pose error. We evaluate the number of matching points, n, between 10 and 200 for a fixed amount of gaussian noise, where $\sigma = 10$.

As expected, all methods generate lower error results as the number of points increases. Figure 4.3 shows that DLT gives unstable results and the error rate merely decreases by increasing the number of points. Posit-C also returns high error rate,

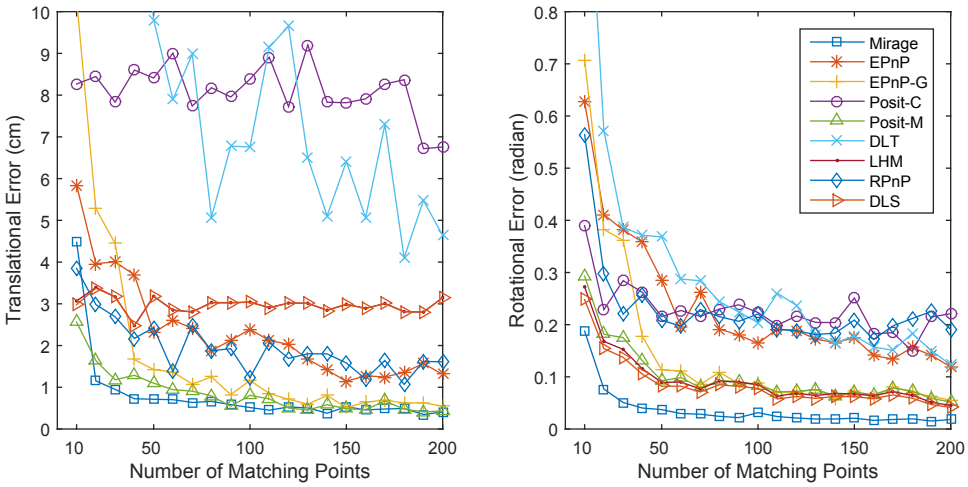


Figure 4.3: Translational and rotational error with respect to the number of points $n = 10$ to 200 and $\sigma = 10$ noise rate

which is more than 8 times of Mirage. Increasing the number of points does not affect the results of LHM and DLS in translational error, however, DLS yields better improvement than LHM in rotational error. EPnP and RPnP generate relatively good results, yet they are not stable enough compared to the remaining methods. EPnP-G and Posit-M yield good results in translational error, but not as good as Mirage in rotational error. Mirage generates significantly good and stable results in both translational and rotational pose error. Our experiments show that using 30-40 number of points would be sufficient to calculate the pose for gaussian noise with $\sigma = 10$ noise rate, which is a realistic assumption for real applications.

4.3.3 Number of Points vs. Processing Time

Previous experiment results show that increasing the number of points decreases the pose error. Although there is a clear advantage of using more points,

the running time of the algorithm is negatively affected from large data size. In this experiment, our goal is to analyze the processing times of the methods with respect to number of matching points. So, we measured the run-time of each method for increasing number of points from $n = 10$ to 200, under fixed amount of pixel noise, where $\sigma = 10$.

According to the results in Figure 4.4, LHM requires very high computational time (0.07 seconds for 200 points), which is larger than the plot range. DLS gives slightly better results than LHM but it is not as efficient as the others. Remaining methods run in reasonable time. EPnP, EPnP-G, and Mirage show similar performance and RPnP is slightly better. It is hard to benefit from fast convergence of Posit-C, Posit-M, and DLT methods to a solution, since Posit methods may fall into local minima with the probability of high error and DLT yields high pose errors. The results clearly show that Mirage is a linear solution since the processing time is linearly increasing. In addition, it runs faster than EPnP and EPnP-G until 100 points that is larger than the sufficient number of points shown in previous results. In a typical real application using the same image size, it is expected to extract around 100-150 key features on the average. In such a system, Mirage can calculate the pose around 4 ms, which is feasible for real time requirements.

4.4 Performance of Mirage on Real Data for a Multi-Camera System

In order to test the performance of Mirage on real environment, we have made tests on real images. In the experiment, we employed a multi-camera system (having 2 Logitech HD Webcam C525 cameras) as shown in Figure 4.5. The image size for

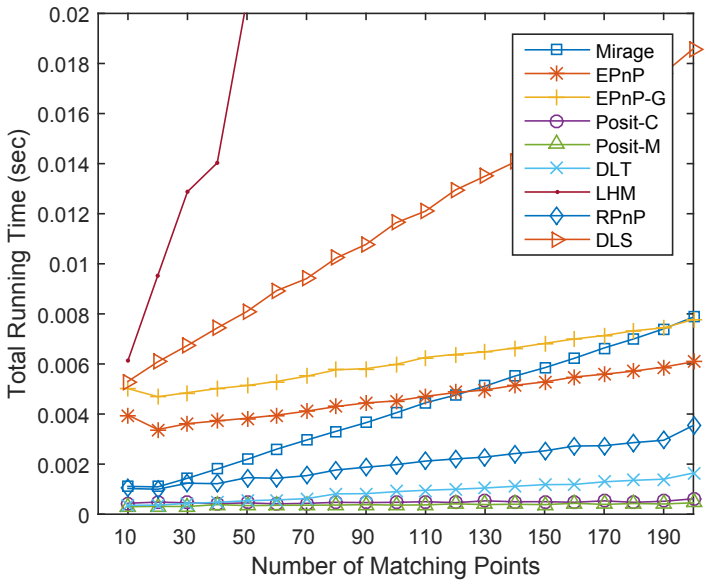


Figure 4.4: Total processing time with respect to number of points $n = 10$ to 200 and $\sigma = 10$ noise rate

Figure 4.5: 2 Cameras are employed in out real data experiments

the experiments was 640×480 pixels. We constructed a target object (Figure 4.6) containing 4 matching points.

The target object is placed to a known (actual) pose on grid (table with markers) and corresponding camera desired pose is defined. Using the real images of the target object and the desired pose, Mirage estimates the camera pose. Finally, the actual and estimated camera poses are compared.

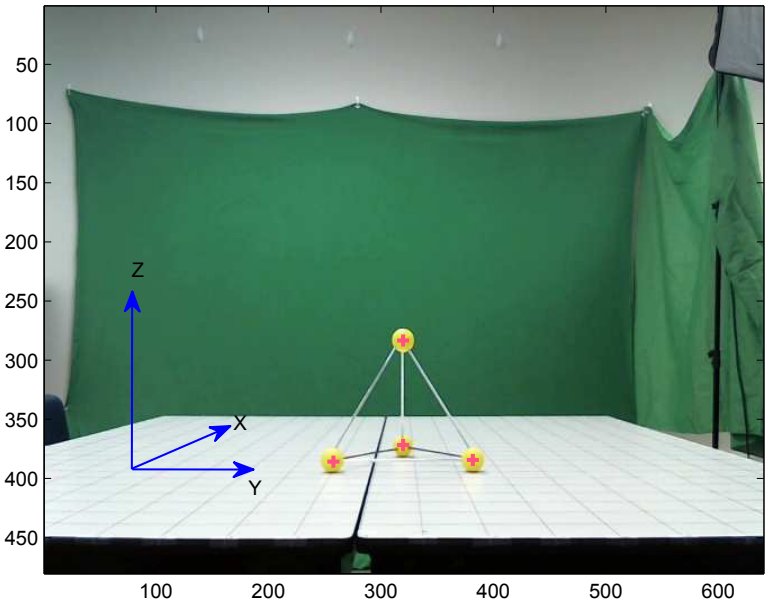


Figure 4.6: Sample target image used in our experiments. Target contains 4 matching points (yellow balls) to calculate the pose

We present the results for each dimension and repeated the experiments several times for different poses. Figure 4.7 shows the results for rotational pose parameters (θ, ϕ, ψ) of the camera in degrees. Results show a small pose error ($5 - 10\%$), which is acceptable for such experimentation.

Figure 4.8 shows the results for translational parameters x, y and z in cm. Results show that Mirage can calculate the pose of the camera with very minor translational error.

4.5 Preconditions and Limitations

In the previous sections, the performance of Mirage pose estimation is presented for various conditions. Although the results are very accurate and promising, there are several preconditions and limitations in the methodology that may require

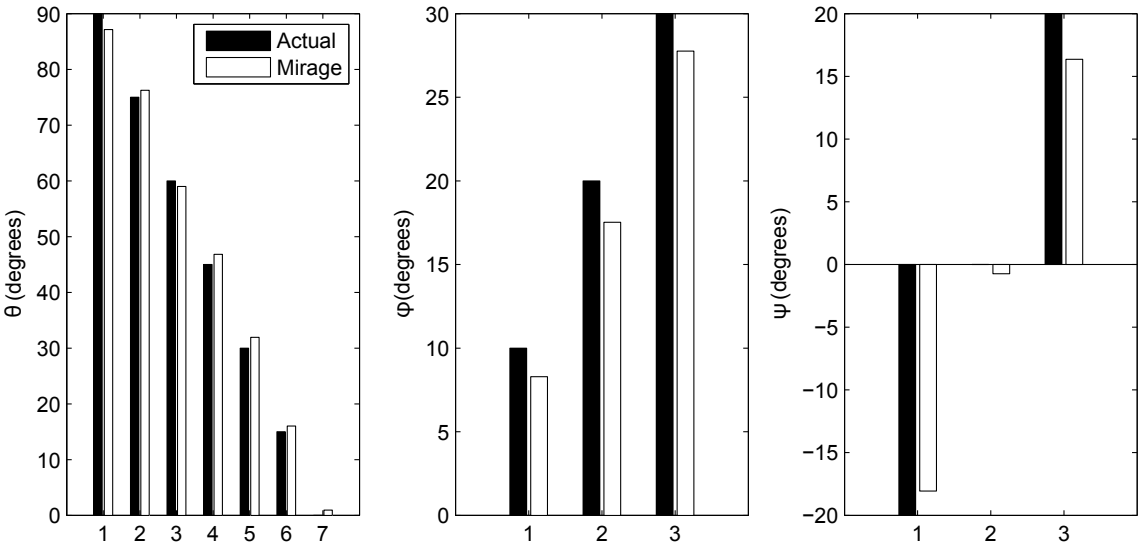


Figure 4.7: Rotational pose calculation results. From left to right: θ, rotation around x axis, ϕ, rotation around y axis, ψ, rotation around z axis

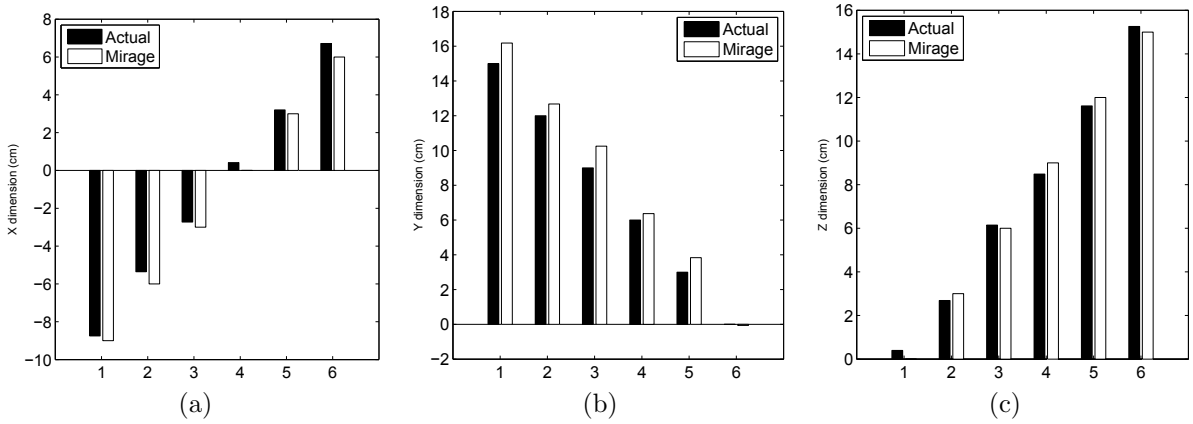


Figure 4.8: Translational pose calculation results. (a) x dimension, (b) y dimension, (c) z dimension

a precaution to prevent unexpected results. Most of these cases can be avoided with simple solutions, however, some of them require special attention.

Pose estimation methods assume the availability of 2D-3D feature correspondence, which means feature extraction stage is not part of the pose estimation. This condition is also true for Mirage. Although we have a separate feature mapping stage, Mirage user must employ a extraction technique (SIFT, SURF, etc.) to generate initial feature set from images. Similarly in this stage, any other challenges such as illumination changes and occlusion should be considered under feature extraction. Nevertheless, the pose recovery algorithm presented in Chapter 6 may be beneficial to reduce the effect of these problems.

Another precondition for Mirage is the availability of colored 3D model of the desired object. It is recommended to use high resolution 3D models, since the desired images are generated using 3D model. At least the resolution of the model is expected to be *good enough* to have reliable and comparable desired images.

One limitation of Mirage is its sensitivity to planar objects. 3D points on the object model must be non-planar to avoid under-determined equation system. However, today almost all 3D object models are non-planar, thus, this is not a significant limitation for real world applications.

Finally, it would be beneficial to mention the special case for single camera structure in Mirage. Single camera system requires an additional equation into the system to solve the unknowns. This is not necessary in two or more camera systems.

CHAPTER 5

TRAJECTORY TRACKING USING MIRAGE

In vision based trajectory tracking problems, a moving vehicle is expected to follow a pre-defined (desired) trajectory (path) using actual and desired views of a reference object. Visual features are extracted from images, and a control law is employed to compute necessary velocities to manipulate the vehicle's pose. Depending on the type of the system, the control law may be image based or position based controller. As discussed in more detail in the related work, the major difference between these systems originates from the pose estimation method. Image based systems only minimize the pixel error in 2D image plane, while position based methods can determine the vehicle pose in 3D space.

5.1 Using Mirage for Trajectory Tracking

Mirage is employed in a position based system with some variations. Figure 5.1 shows the overview of our tracking system. Please note that Mirage does not require direct pose measurement via additional equipment and avoids numerical calculations. Also unlike position based methods, our method calculates the pose error, not the actual pose. Estimated pose error is used by a closed-loop feedback controller to de-

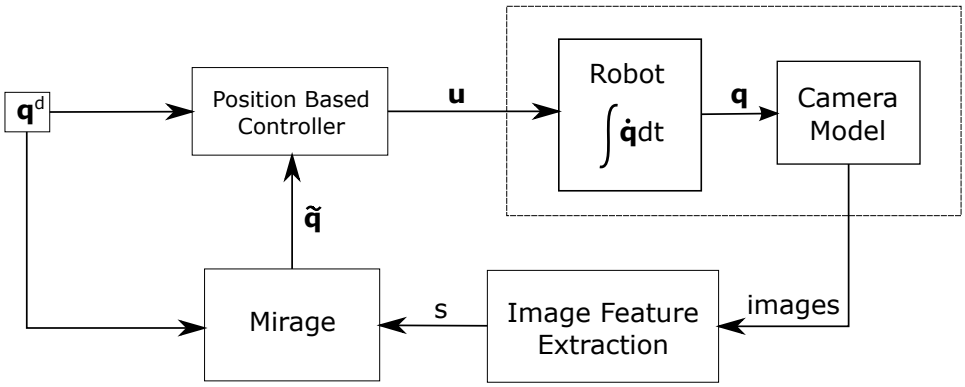


Figure 5.1: Overview of the proposed (Mirage based) trajectory tracking systems. $\mathbf{q}$, s, and $\mathbf{u}$ represent the 3D pose, set of image features, and velocity of the vehicle, respectively.

termine $\mathbf{u} = [\upsilon, \omega]^T$, the linear and angular velocity of the robotic vehicle. Repeating this process for each time instant, the vehicle follows a given trajectory. No assumptions are made on the characteristics of desired trajectory as long as the target points are visible. It can be any arbitrary set of continuous positions in time domain.

The proposed system offers a generic framework that is capable of controlling the vehicle in 3D with 6 degrees of freedom. So our approach can be applied to robotic systems with no modifications. General form of the kinematic model for a robotic system can be expressed as,

$$\dot{\mathbf{q}} = \mathbf{S}\mathbf{u} \tag{5.1}$$

where $\dot{\mathbf{q}}$ is the time derivative of the vehicle pose and $\mathbf{S}$ is the vehicle specific matrix. Depending on the motion constraints of the system, $\mathbf{S}$ changes. In the following, characteristics of two sample systems are described. The first system is a 6DOF vehicle with no motion constraints. And the latter is a 3DOF two-wheeled nonholonomic

mobile robot. Both systems are selected to demonstrate particular difficulties to be solved.

5.2 6DOF Unconstrained System

We prefer to start with a 6DOF robotic vehicle (e.g., an underwater robot). In such a system, **S** matrix in Eq. (5.1) is an identity matrix since no motion constraints are specified. Hence the new form of system model becomes $\dot{\mathbf{q}} = \mathbf{u}$. The controller of the system can be specified as

$$\mathbf{u} = \lambda \tilde{\mathbf{q}} \tag{5.2}$$

where $\mathbf{u}$ the velocity vector fed into system model, λ is small negative constant and $\tilde{\mathbf{q}}$ is the pose error generated by Mirage. After substituting $\mathbf{u}$ in given equations, the new form of the system model becomes $\dot{\mathbf{q}} = \lambda \tilde{\mathbf{q}}$, which suggests that the motion is realized by direct integration of pose error after multiplying by λ. Please note that this system is applicable to limited real applications; however, it has a simple and straightforward structure, which makes these systems suitable for simulation purposes.

5.3 2-Wheeled Nonholonomic Mobile Robot

Planar vehicles can also benefit from our tracking system. We applied our approach to a nonholonomic two-wheeled robot in a virtual 2D system with 3 degrees of freedom. Figure 5.2 demonstrates the schematic view of the chosen robot, where

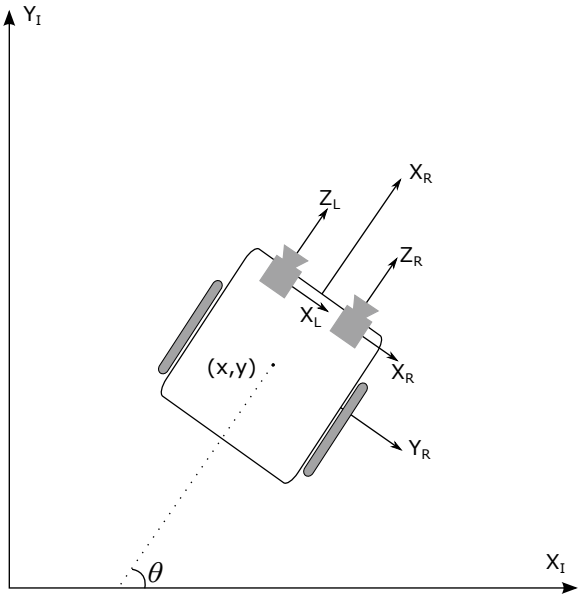


Figure 5.2: Two-wheeled nonholonomic mobile robot

x, y are coordinates of the axis center, and θ is the orientation of the robot, such that $\mathbf{q} = [x, y, \theta]^T$.

This system may also be represented by the kinematic model given in Eq. (5.1) with modified $\mathbf{S}$ matrix such that

$$\dot{\mathbf{q}} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \mathbf{u} \tag{5.3}$$

where $\dot{\mathbf{q}}$ is the time derivative of the pose and $\mathbf{u} = [v, \omega]^T$. In simulation, we consider the robot moving along a desired trajectory using the vision system containing two calibrated cameras (not stereo vision mode) mounted on the robot body. For each time instant, robot checks its pose with respect to the desired pose, then moves with the objective of reducing pose error. In this paper, we briefly describe the derivations

of controller. The detailed information is provided in [60]. In order to derive the $\mathbf{u}$ vector, the following equation may be used [61]:

$$\mathbf{u} = \begin{bmatrix} v \\ \\ w \end{bmatrix} = \begin{bmatrix} v^d \cos\tilde{q}_\theta + v^b \\ \\ w^d + w^b \end{bmatrix} \tag{5.4}$$

where v^d and w^d are the desired input velocities and v^b and w^b are feedback signals which are explained in [60]. The inertial posture error $\tilde{\mathbf{q}}_I = [\tilde{q}_{I_x}\ \tilde{q}_{I_y}\ \tilde{q}_{I_\theta}]^T$ is necessary for computation but the errors calculated by Mirage are only available in local coordinate system not in global coordinate system. Hence, following transformation is necessary:

$$\begin{bmatrix} \tilde{q}_{I_x} \\ \\ \tilde{q}_{I_y} \\ \\ \tilde{q}_{I_\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ \\ -\sin\theta & \cos\theta & 0 \\ \\ 0 & 0 & 1 \end{bmatrix} \tilde{\mathbf{q}} \tag{5.5}$$

where $\tilde{\mathbf{q}}$ is the local pose error calculated by Mirage and $\theta = \theta^d + \tilde{\theta}$. In this manner, the posture error model can be shown as

$$\begin{bmatrix} \dot{\tilde{q}}_{I_x} \\ \\ \dot{\tilde{q}}_{I_y} \\ \\ \dot{\tilde{q}}_{I_\theta} \end{bmatrix} = \begin{bmatrix} \cos\tilde{q}_{I_\theta} & 0 \\ \\ \sin\tilde{q}_{I_\theta} & 0 \\ \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v^d \\ \\ w^d \end{bmatrix} + \begin{bmatrix} -1 & \tilde{q}_{I_y} \\ \\ 0 & -\tilde{q}_{I_x} \\ \\ 0 & -1 \end{bmatrix} \mathbf{u} \tag{5.6}$$

It is shown in [60] that the error can vanish by employing the feedback signals v^b and w^b as follows:

$$v^b = k_x \tilde{q}_x \tag{5.7}$$

$$w^b = k v^d \tilde{q}_y (1 + \frac{e_{cos}}{a})^2 + k_s e_{sin} [(1 + \frac{e_{cos}}{a})^2]^n \tag{5.8}$$

where $\tilde{q}_x$ and $\tilde{q}_y$ are local errors, k_x and k_s are positive values and $n \in Z$. [60] also suggests that n should be chosen as a small number such as -2, -1, 0, 1 or 2 for practical reasons. Parameters k and a are constants larger than zero. And finally directional error parameters, e_{sin} and e_{cos} in Eq. (5.8) are presented as follows:

$$e_{sin} = \sin(\tilde{q}_{I_\theta}) \tag{5.9}$$

$$e_{cos} = \cos(\tilde{q}_{I_\theta}) - 1 \tag{5.10}$$

5.4 Application to 6DOF Unconstrained System

The first simulation is applied to the 6DOF vehicle. Using such an unconstrained system, we address a well-known benchmark problem [62] on which image based systems usually generate undesirable motion. This is an unavoidable issue in such systems due to their limited solution space (2D image plane). Optimal minimization on the image plane does not yield optimal solution in 3D space.

The problem involves a square shape target object that is placed on a fixed position and the robotic vehicle is rotated around only one dimension with respect to

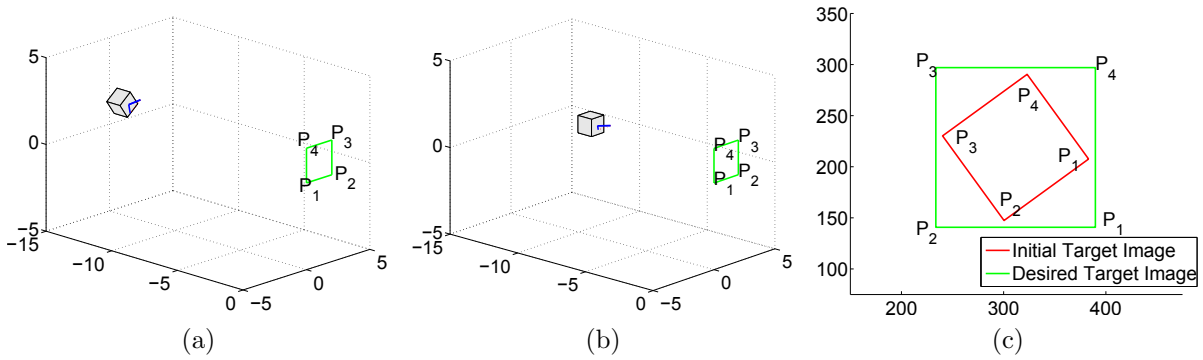


Figure 5.3: Initial and desired pose of the robot: a) initial position: rotated 30 degrees and placed 2 m behind the desired position, b) desired position, c) initial and desired target images

its desired position. When the system runs, the motion of the vehicle is analyzed until convergence to the desired pose. To test this problem on the proposed method, the initial position of our robot is rotated 30 degrees and placed 2 m behind the desired position. An illustration of this scenario can be seen in Figure 5.3 with initial and desired positions of the vehicle, and target images for both cases.

This scenario is simulated using traditional image based, position based, virtual visual servoing approach[1] [39], and the proposed approach. Detailed comparison results are shown in Figure 5.4. Note that only differences between initial and desired pose are in x dimension and ϕ angle. Therefore, the expected motion change must be in only x and ϕ. According to the results, all 4 methods converge to the desired pose. However, there are significant differences in the image based and virtual visual servoing results. First, they converge later than the other 2 methods. Second, the robot makes undesirable motion in almost all dimensions due to 2D error minimiza-

[1]Source Code can be downloaded at http://www.irisa.fr/lagadic/visp/visp.html

tion process. Particularly, on x direction, the robot moves back for some time to correct its position. On the other hand, the position based method yields satisfactory results for all dimensions. Majority of position based methods calculate the 3D pose of the robot via numeric (iterative) solutions, which are usually slow for real time implementation. Also there is a possibility to be trapped in local minima during iterative process [63]. Since the 3D pose of the robot is directly fed into the controller, the position based solution generates good results. Our proposed approach provides very similar results with the position based method, while only using the 2D image pixels of the target and obtaining the motion parameters in Euclidean Space with low computational complexity. This is a noticeable advantage of our method over other solutions.

5.5 Application to 3DOF Planar System

Second simulation is presented using 3DOF planar system described in Section 3.2. The desired trajectory is defined as a function of time in a 2D environment and the initial pose of the robot is set. The objective is to reduce the pose error by moving the robot as time passes. A sine wave function is selected to represent the trajectory, since it has sufficient complexity for such systems. The initial position of the desired trajectory is $\mathbf{q}_0^d = [-30, 2, 0.5610]$ and the initial position of the robot is $\mathbf{q}_0 = [-32, -2, 0]$. The simulation is run for 25 seconds. This experiment has been repeated for noise-free and noisy scenarios and the results of 5 state-of-art pose estimation techniques are compared on noisy environment

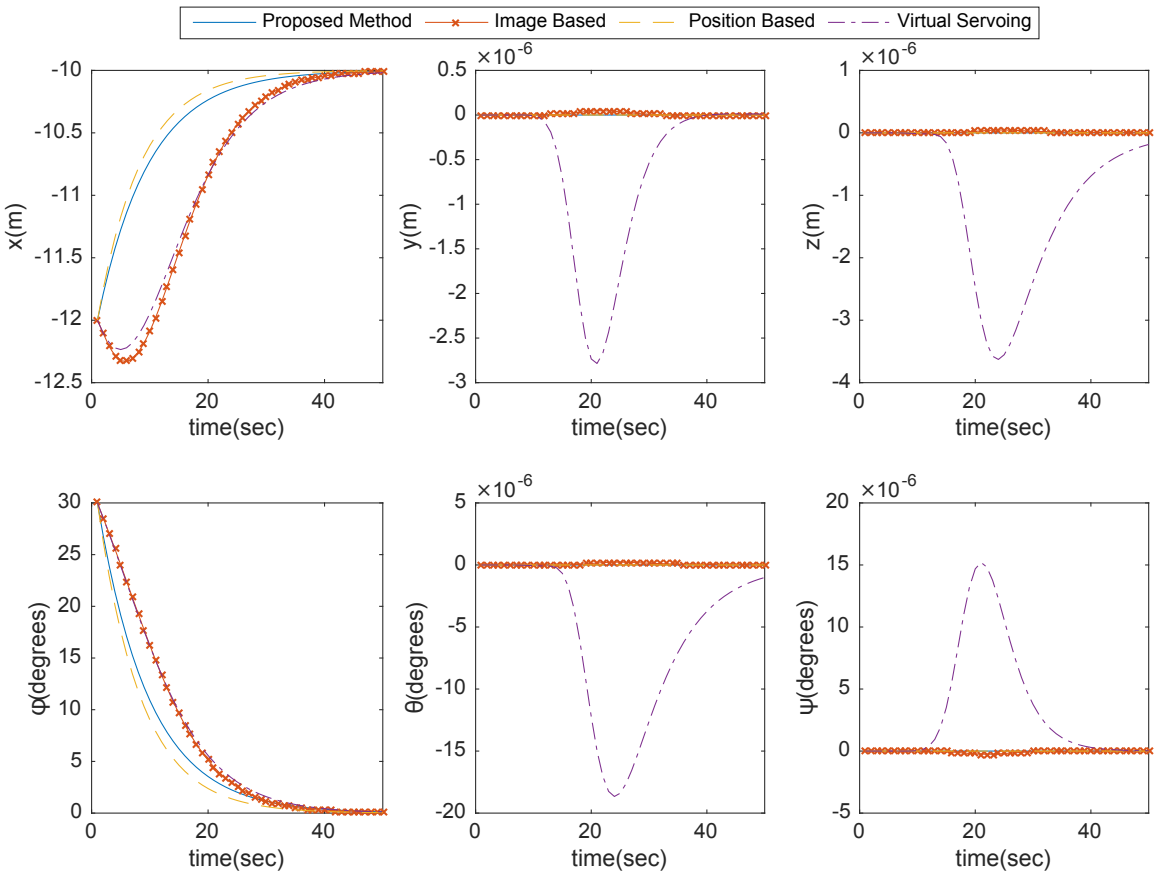


Figure 5.4: Comparison of 3 methods with respect to 6 degrees of freedom. In image based system and virtual servoing, undesirable motion exist on x, y, and z dimensions, also on θ and ψ angles.

5.5.1 Noise-Free Environment

In the first case, we consider an ideal condition, where there is no noise in the environment. We intend to see whether our method generates unexpected results under perfect conditions. The controller gains are selected as $k = 1$, $k_x = 1$, $k_s = 1$, $a = 3$, and $n = 0$. Figure 5.5 illustrates the simulation results.

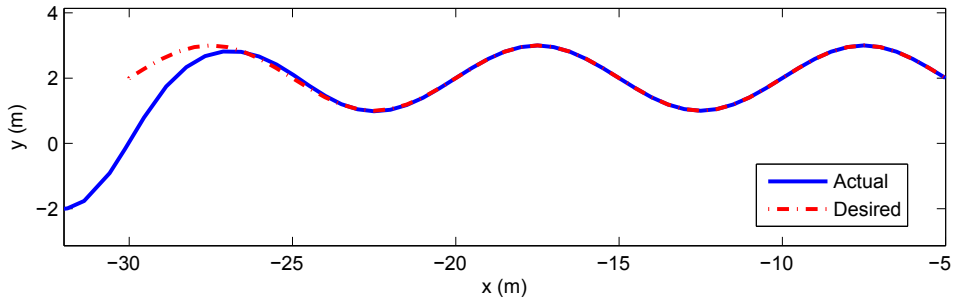
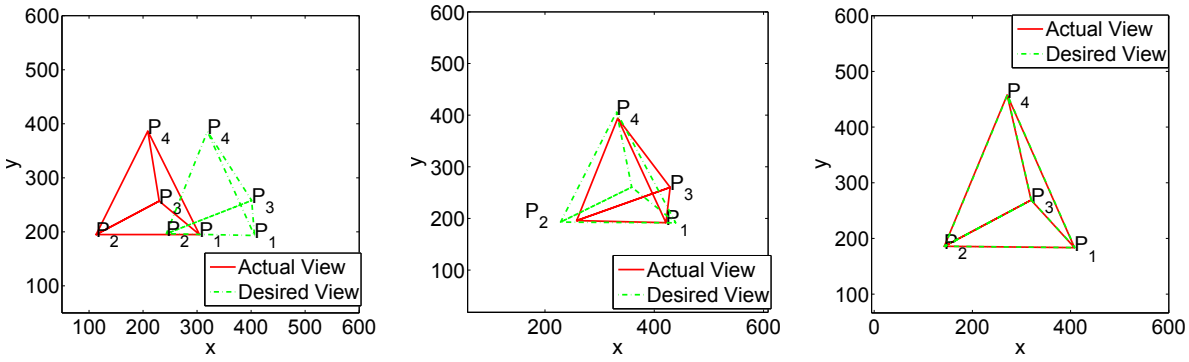


Figure 5.5: Tracking results under noise-free environment. Convergence is achieved at 9^{th} second of the motion.

As shown in Figure 5.5, mobile robot converges to the desired trajectory and finally follows it with no error. Figure 5.6 shows how the image of the target in the left camera looks like when the robot is at actual and desired positions in 3 different frames. The actual image of the target converges to the desired image of the target approximately at the 9^{th} second of the simulation.



(a) (b) (c)

Figure 5.6: Target image in the left camera; a) at time 0, b) at time 5, c) at time 9

5.5.2 Noisy Environment

In the second case, we added certain amount of gaussian noise ($\sigma = 20$) to the target pixel coordinates and ran the same simulation on 3DOF mobile robot. This time we also tested 5 pose estimation methods: Direct Linear Transform (DLT) [20], Lu-Hager-Mjolsness (LHM) [14], Modern POSIT (Posit) [64], Efficient PnP (EPnP) [21], and Robust PnP (RPnP) [23]. Selected methods include numeric and analytic solutions with varying complexities. The controller gains are selected as $k = 1$, $k_x = 1$, $k_s = 1$, $a = 3$, and $n = 0$. Figure 5.7 shows the tracking results of each method.

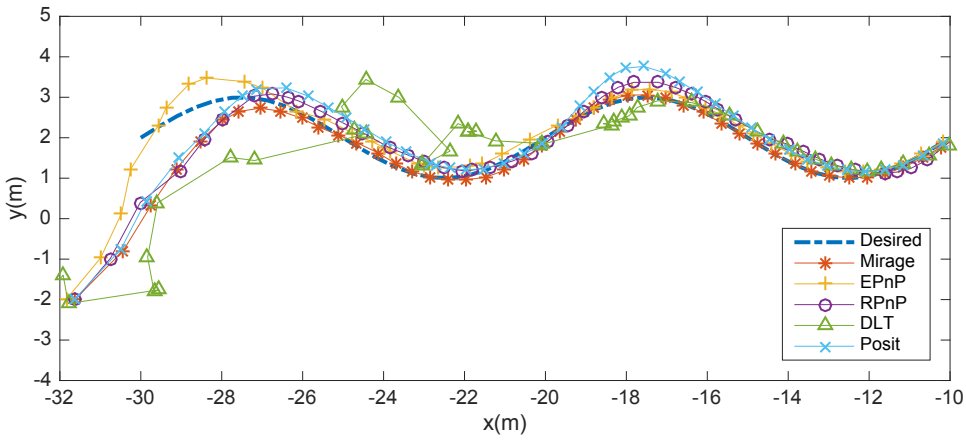


Figure 5.7: Tracking results under noisy environment

In Figure 5.7 it can be seen that, DLT generated inconsistent motion, which cannot be applicable to a real system. EPnP, RPnP, and LHM can converge to the desired trajectory with high level of error, which may be critical for some applications. Posit returns relatively good results but it is not as good as Mirage. Our method is the most robust pose estimation method in the given scenario.

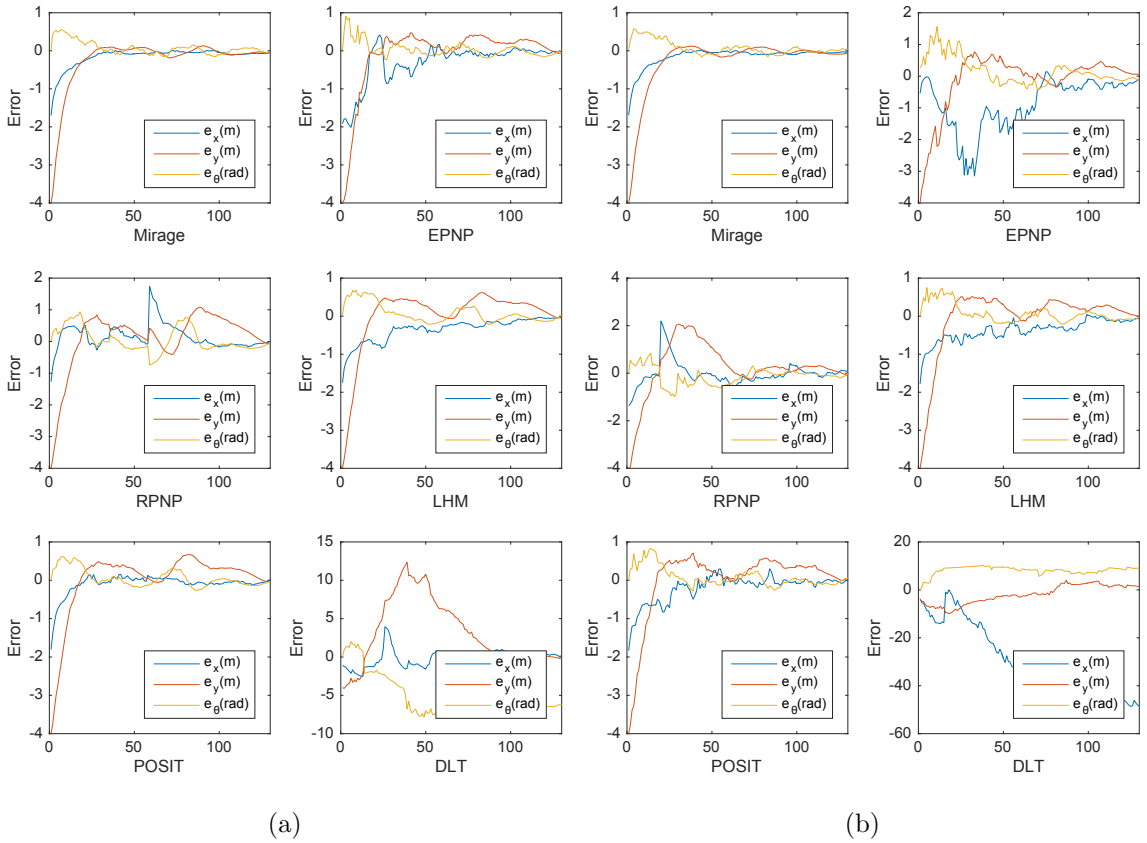


Figure 5.8: Pose error minimization during the simulation using (a) 20 target points, (b) 5 Target Points

We also track the minimization of pose errors by the time. Figure 5.8 (a) shows the results of each method using 20 target points. The results clearly indicate that Mirage is very stable under the noise while others produce significant errors. Also, using 20 points is much more than necessary points for Mirage. When we reduce the number of points to 4 (minimum for Mirage) the error rate of some methods becomes very high. Particularly, Posit cannot even calculate the pose in 4 point case. We found that 5 is the minimum number that runs for all methods. So, the same scenario is

Table 5.1: Sum of squared errors of pose estimation methods for each dimension with respect to the number of target points

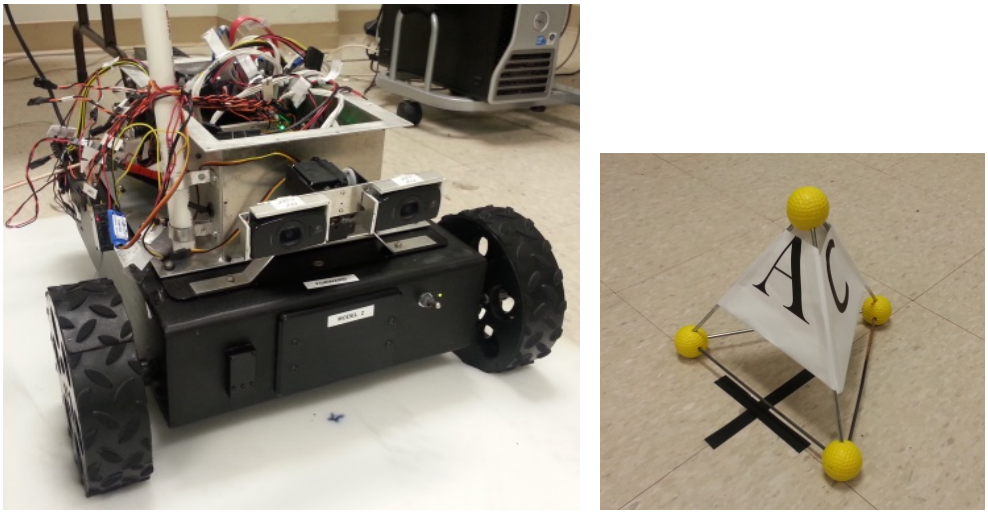
	4 Points			20 Points			50 Points			100 Points		
	e_x	e_y	e_θ	e_x	e_y	e_θ	e_x	e_y	e_θ	e_x	e_y	e_θ
Mirage	3.24	8.78	2.31	3.35	8.82	2.16	3.41	8.73	2.17	3.12	8.90	2.21
EPnP	50.01	36.72	15.97	8.57	10.44	4.12	6.15	10.01	2.98	4.67	9.99	2.84
RPnP	7.40	11.91	3.36	3.73	9.60	3.17	4.00	10.26	2.83	2.86	9.78	2.85
LHM	9.58	23.23	7.33	5.80	9.80	2.69	6.08	9.89	3.01	5.421	9.73	2.80
Posit	N/A	N/A	N/A	4.19	9.56	2.67	3.04	9.71	2.79	3.29	9.58	2.51
DLT	143.62	47.68	70.48	15.98	17.50	8.97	10.22	12.35	3.81	5.52	10.53	4.81

tested using 5 points and the results are provided in Figure 5.8 (b). The results show that Mirage generate very similar results using low number of target points while some other methods, such as EPnP and RPnP, generate significant errors compared to the 20 point case.

To investigate the effect of the number of points, we conducted further experiments from 5 to 100 points and provide the results in Table 5.1. Table entries show the sum of squared errors (SSE) of a method for given number of points. The results indicate that the error rate of the Mirage is very low for any number of target points and it is independent of the number of points. This is a significant advantage when only limited number of features can be extracted from the images. On the other hand, EPnP and DLT produce very high error on small set of target points and the error decreases by increasing the target points. RPnP and LHM are more robust to the change of target points but the errors are still not as low as Mirage. Posit cannot calculate the pose using 4 points, but then generates relatively good results.

5.6 Real Time Tracking using 3DOF Non-holonomic Vehicle

Experimental Setup. Proposed tracking algorithm has also been applied on a real non-holonomic robotic system to present its efficiency with real equipment. In the experiments, a 2 wheeled ground vehicle holding 2 cameras is utilized (Figure 5.9(a)). 640 $\times$ 480 resolution images are captured from the cameras and then images are processed by onboard computer to detect the target points. The points are fed into Mirage to calculate the pose errors that are converted to velocity using the controller system provided in Section 3.2. Our experiments show that the controller gains in Eq. (5.7) and Eq. (5.8) should be $k = 0.3$, $k_x = 1.5$, $k_s = 1.5$, $a = 3$, and $n = 0$ in order to receive accurate results for our robotic system. The time consumed for image acquisition and feature extraction stages is about $285ms$. The time for Mirage pose estimation using 4 feature points and two cameras is about $15ms$.



(a) (b)

Figure 5.9: a) Non-holonomic vehicle employed in the real experiments, b) 4 point target object

We defined a 3-meter path and desired trajectory. The vehicle moves on x direction with varying velocity by the time. On this planar space, initial pose of the vehicle is $\mathbf{q}_0 = [-4, 0, 0]$ and a 36-second desired trajectory is defined such that the vehicle moves forward with sinusoidally increasing velocity for 9-second to the pose $\mathbf{q}_1 = [-2.5, 0, 0]$ and then continues with sinusoidally decreasing velocity for 9-second to zero, the pose $\mathbf{q}_2 = [-1, 0, 0]$. After that it stops and moves backwards to $\mathbf{q}_0$ following the same pattern in the remaining 18 *sec*. A 4 point target object in Figure 5.9(b) is constructed and placed at the origin.

Results. The experiment is repeated 2 times and in total of 150 seconds. The robot is initially at rest at $x = -4m$ and $y = 0m$. The control is switched to automatic tracking at $t = 11$ and $t = 95$ seconds, when the robot starts moving for a period of 36 seconds. The robot is commanded to move straight forward and backward along in the x direction for 3-meter, while increasing and decreasing its speed in a sinusoidal pattern. Considering two 36-second periods in Figure 5.10, the vehicle follows the trajectory in all 3 dimensions without significant sum of squared errors (SSE) such that $e_x = 4.5204m^2$, $e_y = 0.8261m^2$, and $e_\theta = 0.4942rad^2$.

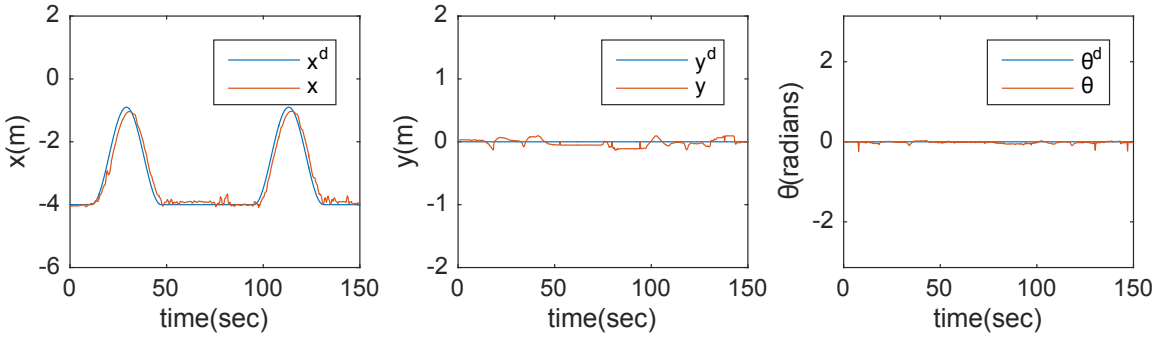


Figure 5.10: Experiment results with real robotic system

5.7 Summary

This chapter presents the technical details of how to apply Mirage pose estimation method to the trajectory tracking problem. Unlike image based methods, proposed system analytically calculates the pose in 3D Euclidean space with 6 motion parameters. This is a noticeable advantage that provides high accuracy in Euclidean trajectory tracking problems as in position based methods. Moreover, unlike position based methods, the Euclidean pose is calculated analytically without iterative solutions.

Mirage based trajectory tracking system can be applied to various types of mobile vehicles having spatial motions (e.g., aircraft, spacecraft, underwater robots). However, it can also be easily applied to vehicles with planar motion. In this work, we demonstrated two sample applications, and provide real environment and simulation results that verify the effectiveness of our method. According to the results Mirage based system converges to the trajectory with less error in shorter time than other methods and it is still robust in noisy environments. Both simulations and actual experiments successfully confirm the advantages of the proposed methodology.

CHAPTER 6

POSE RECOVERY

Pose estimation is a challenging problem. An accurate pose of the camera (or the carrier vehicle) needs to be calculated for every time instant during the motion. Aside from the algorithmic and computational difficulties, there are also challenges arise from the environmental changes or instantaneous imaging errors. In real world applications, it is not a rare case that the lighting of the environment changes, the camera view is occluded, or the camera fails to capture an accurate image for that instant. Usually, in such cases, a temporary miscalculation happens, and it causes the determination of incorrect motion directives to the robotic vehicle. Moreover, motion on incorrect direction often amplifies the error and that may end up with a complete failure of the task. Particularly trajectory tracking applications are susceptible to this problem since the vehicle pose is calculated every time instant and new calculations depend on previous motion. In order to compensate the pose error at the early stages, a pose recovery procedure is necessary, where an emergency algorithm intervenes in the system to calculate the correct pose.

6.1 Pose Recovery Problem

Pose recovery is the procedure that involves two stages: detection of incorrect pose calculation and then recovering the pose. First step is necessary to decide when to start recovery calculations. A common way to solve this problem is considering a series of previous pose calculations and estimating the next pose. A comparison between the estimated future pose and the calculated pose would give a clue to decide whether the calculation is correct or not. Because, for incorrect calculations, the pose is usually much different (or unrealistic) than the previous poses. Once an incorrect calculation is detected, the recovery algorithm can be activated to recover correct pose. Later, the system may run normally using the correct pose.

6.2 Overview of the Proposed Method

The terminology of pose recovery in research community conveys broad meanings for different applications. In this research, we are narrowing down this definition into our particular problem, "trajectory tracking". In our case, the *vehicle pose* needs to be *recovered*, if the actual view and the desired view of the reference object are very different and 2D features of the images do not match. Therefore, Mirage pose calculation may not be performed and the vehicle moves off the desired path. Figure 6.1 illustrates a sample case, where an actual vehicle and the desired vehicle are at very different positions, thus the images taken from their positions do not have sufficient number of common feature points.

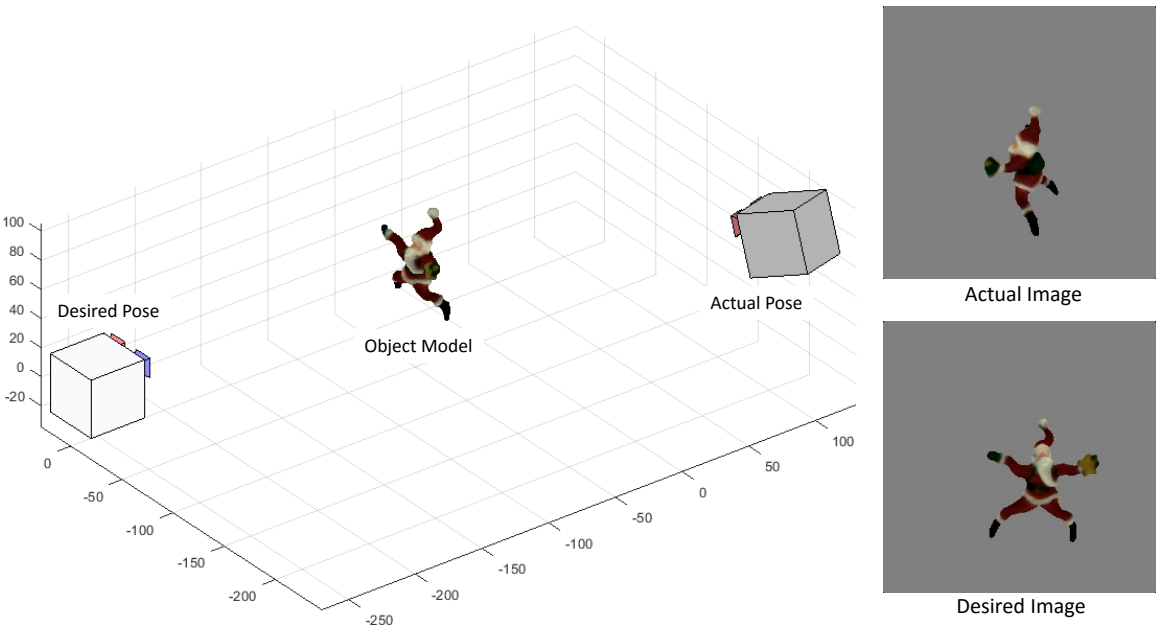


Figure 6.1: A problematic pose estimation case. Actual and desired vehicle poses are very far from each other. Therefore, their images cannot be matched properly.

In order to solve this problem, we propose an alternative recovery algorithm, where multiple *virtual images* of the reference object are employed to calculate the correct pose. A virtual image is taken from a virtual camera pose, which is *systematically* selected to cover all views of the object. According to the algorithm, if we have enough number of virtual images covering sufficient number of object views, then at least a group of images will closely match with the actual image as shown in Figure 6.2. For those cases, Mirage will return the correct pose. However, for all other cases, the results will be irrelevant and far from the others in the solution space. Therefore, clustering the solutions based on the neighboring information can lead us to a solution.

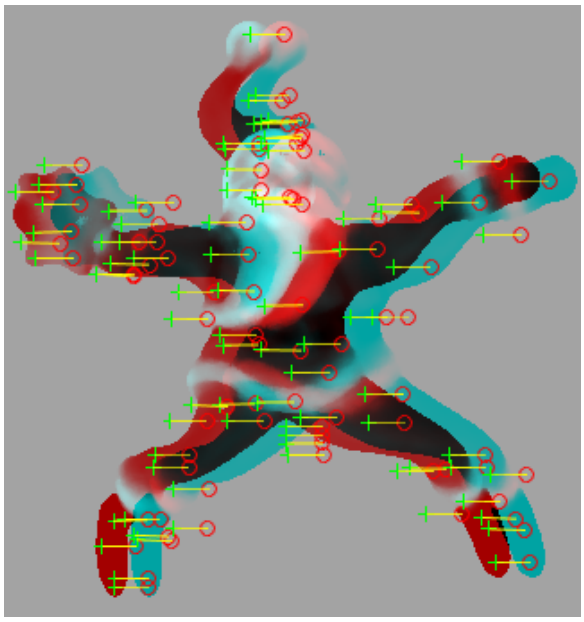


Figure 6.2: SIFT feature matching results of the actual and desired images

6.3 Related Work

In the literature, recovery algorithms developed for variety of applications. Krupa et al. [65] propose a methodology that moves a surgical robotic arm from an unknown position to the desired 3D position. They utilize special optical markers generated by laser lights to recover the pose of the robotic arm. Ozuysal et al. [66] propose a keypoint recognition system using Random Ferns. Their system starts with ferns that have been trained with known 2D features. Later during the trajectory tracking, a second set of ferns are trained to unknown features in the environment in case of a failure in tracking. In Diosi et al. [67], two consecutive reference images are used to reconstruct local 3-D scene. If a set of tracked features are lost, they are projected back into the current image using 3D information. Morward et al. [68] employ special distinctive features placed on the reference object to recover the pose.

The idea of generating multiple virtual poses in the proposed method resembles to an algorithm called "characteristic views" (CV) [69], which aims to determine minimum number of representative images of an object to cover a complete 3D view. It is commonly used in 3D model retrieval [70] and object recognition [71] problems. Besides, many other studies have been proposed for more reliable and efficient CV generation. Theetten et al. [72] proposed a novel CV generation method using the visual hulls [73], which are the intersected 3D regions of back-projection cones of 2D images. Hausdorff distance is used as a metric to measure similarity between the visual hulls. Ansary et al. [74] proposed an alternative CV generation technique based on the adaptive clustering algorithm and probabilistic Bayesian approach. He et al. [75] propose a cluster based CV selection approach, in which multiple views of the object is generated to cover every angle. Later, the views are clustered using support vector machines (SVM) with radial basis function (RBF) kernel. Finally, every cluster center is considered as a single CV.

Our pose recovery algorithm uses the virtual views to estimate the camera pose, rather than targeting to generate more representative views. The characteristic views can be beneficial at the initial stage of our pose recovery algorithm. It can be utilized to reduce the number of virtual poses, which lowers the burden in calculations. In the future, we plan to include this technique in our system to determine an optimum number of virtual poses.

6.4 Method

Recovery process starts with defining multiple virtual views of the reference object. It is beneficial to determine the virtual poses systematically, so that every view of the reference object is covered. In this research, we used a spherical mesh of 3D points to determine the positions of virtual vehicle poses. Each 3D point is assumed to be the position of a single virtual pose. Viewing direction of the camera is always through the center of the sphere, where the reference object is located. Finally, *upward* direction of the camera is selected as $+z$ axis. Figure 6.3 illustrates a sample scene including few number of poses. In order to obtain more robust estimation to significant distance changes, the idea can be extended to more poses, or even multiple spherical meshes having different radii.

By generating multiple positions, we target to acquire a group views that are similar to the actual view of the reference object. If a virtual view is not matching to the actual view, Mirage will generate irrelevant results. These irrelevant results indicate that those virtual views are not close to the actual view. Mirage will generate correct results for only the views that are similar to the actual view. In this manner, if there are multiple views similar to the actual view, then there will be a group of correct results. This group can be detected using their inter-point distances in the 6 dimensional solution space.

Consider a problematic case similar to the one in Figure 6.1. Let $\mathbf{s}$ be the actual image, $\mathbf{q}$ be the actual vehicle pose to be calculated, and $\mathbf{q}^d$ be the corresponding

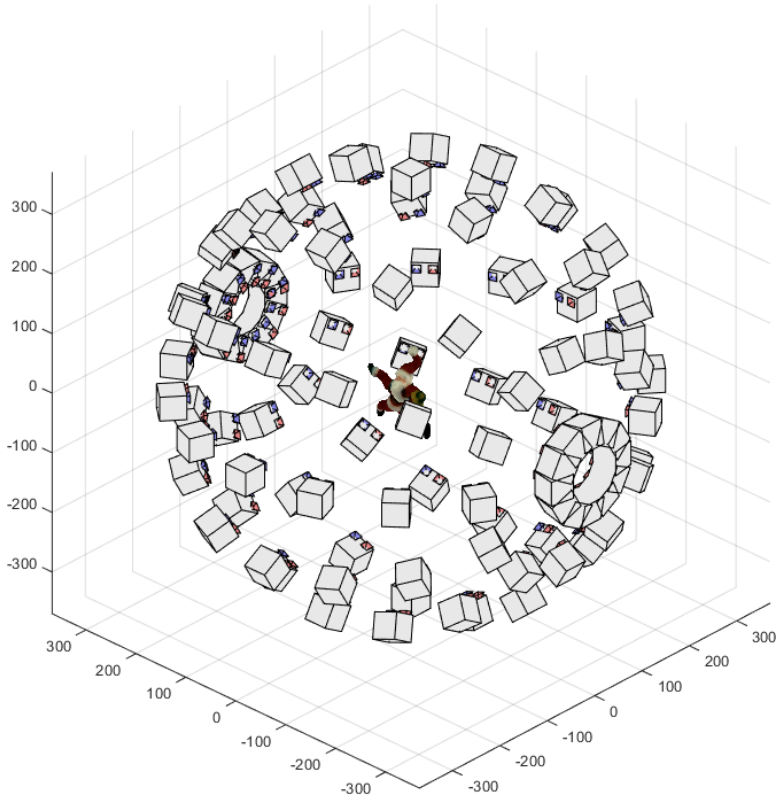


Figure 6.3: Multiple virtual poses of the vehicle are generated systematically using a spherical mesh

desired pose. Since the desired image is not enough to determine the pose, virtual views of the reference object are generated such that

$$\mathbf{Q} = \{\mathbf{q}_1^v, \mathbf{q}_2^v, \ldots, \mathbf{q}_n^v\} \tag{6.1}$$

$$\mathbf{S} = \{\mathbf{s}_1^v, \mathbf{s}_2^v, \ldots, \mathbf{s}_n^v\} \tag{6.2}$$

where $\mathbf{Q}$ and $\mathbf{S}$ are a set of n virtual poses and corresponding virtual images (2D projections w.r.t. virtual poses), respectively. To calculate the vehicle pose, Mirage

employs the actual image, virtual image, and the virtual pose. Next, each virtual image and virtual pose are fed into the Mirage along with the actual image.

$$\mathbf{q_i'} = Mirage(\mathbf{s}, \mathbf{s}_i^v, \mathbf{q}_i^v), \quad \text{where } i\text{=1,2,...,}n \tag{6.3}$$

The result of each virtual view is stored in an estimated pose $\mathbf{q_i'}$. The final step is analyzing the results and searching for common solution that at least a group of virtual views agree. A convenient way to find this group is to search k nearest neighbors of each $\mathbf{q_i'}$ and to calculate the total inter-point distances of each group. Ideally, if a set of points are very close to the correct solution, then their total inter-point distances would be the minimum. Because all incorrect solutions would have very large distances to each other. Using this idea, we find the closest k points in n and find the median of k points (center of the group) as the final solution.

6.5 Complexity and Parallelization

Generating virtual views is not necessarily an online step. All virtual pose and image generation can be done off-line before the tracking is started. Moreover, image processing and feature extraction procedures can also be done off-line, since only 2D features and corresponding 3D coordinates are necessary for Mirage calculation. In this way, a reduction on computational load can be achieved.

The number of virtual views n can be bottleneck in the system. For more complex environments, more virtual views would be necessary, but increasing n will

also increase the computational complexity of the solution. To solve this problem, an efficient optimization can be achieved by exploiting "data level parallelism". Our algorithm requires computation of every virtual pose independently to determine the recovered pose. This is a typical SIMD (Single Instruction Multiple Data) structure, where a single instruction (Mirage calculation) is applied to the same type of multiple data (virtual poses). Since the calculation of each virtual view is completely independent, a separate processor thread can be conveniently assigned to a virtual view, and the work can be completed in parallel. This application can also be very suitable for many-core processors such as GPUs (Graphical Processing Units).

6.6 Experiments

In this section we present the experimental results of our algorithm. In the first experiment, proposed algorithm is tested on a similar problematic case as shown in Figure 6.1. In this simulated environment, the reference object is located at the origin of the 3D space and the actual pose is selected as $\mathbf{q} = [-350, 5, 10, 0.1, 0.2, 0.3]$. Since the desired pose is very different, regular Mirage calculation is not possible, therefore, a total of 441 virtual views of the reference object are generated for pose recovery. Figure 6.3 can be seen for a reference illustration.

All virtual views are processed and their results are stored. Some of the virtual images do not match with the actual image as expected. As a result, a sufficient number of matching features cannot be extracted, and Mirage does not return a solution. These cases can easily be eliminated from the list, since these virtual views are not similar to the actual view. After removing these cases, all remaining results

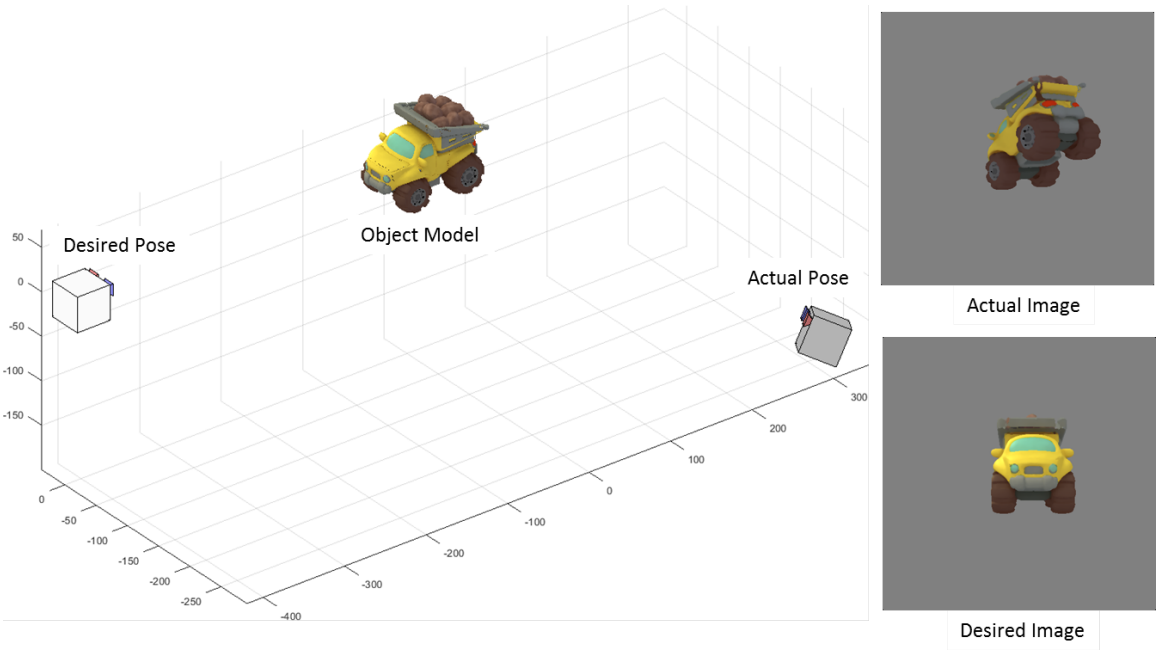


Figure 6.4: The initial condition of *Truck* experiment

are grouped using *k-nearest-neighboring algorithm*, where k is selected as 20. In order words, we search for closest 20 points in the solution space. These 20 points are the most likely the group of solutions that we are targeting, since all other incorrect solutions can not be closer. Median of the selected group is marked as the final solution. In this experiment, the final solution is calculated as $\mathbf{q_f} = [-353.1, 4.8, 9.8, 0.1, 0.04, 0.3]$, which clearly shows that our pose recovery algorithm calculates the pose very close to the actual pose of the vehicle.

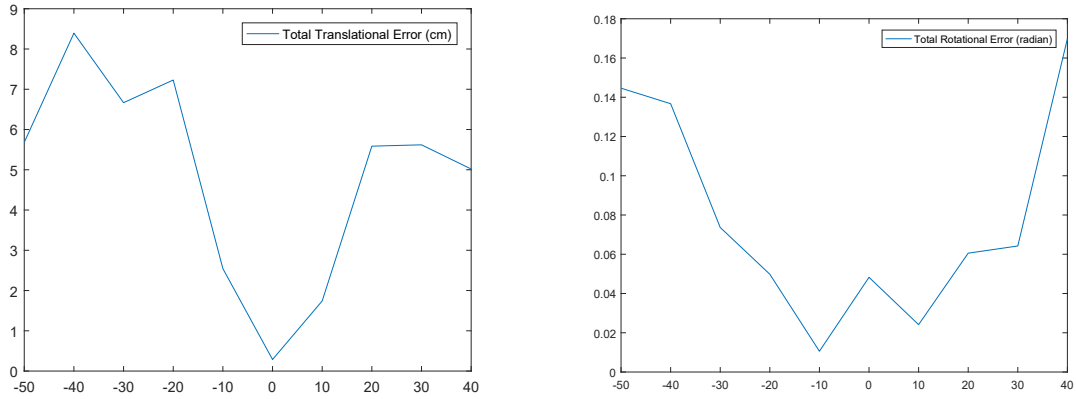
In the second experiment, we used a new reference object *Truck* in a scenario illustrated in Figure 6.4. This experiment is designed to show the performance of our algorithm under the conditions, where no virtual pose perfectly matches to the actual pose. Thus, a series of actual poses are determined on the x axis. The initial pose of the actual vehicle pose is selected as $\mathbf{q} = [-350, 0, 0, 0, 0.5, 2.5]$ and the x position

Table 6.1: Pose recovery error of each dimension for varying distances

Distance(cm)	e_x	e_y	e_z	e_ϕ	e_θ	e_ψ
-50	2.4354	0.1975	3.0484	0.0001	0.0843	0.0601
-40	5.6104	0.0639	2.7185	0.0022	0.0721	0.0623
-30	4.3450	0.2424	2.0791	0.0046	0.0410	0.0279
-20	5.1895	0.1406	1.3992	0.0047	0.0245	0.0204
-10	1.7957	0.0731	0.6755	0.0006	0.0006	0.0092
0	0.0278	0.1902	0.0689	0.0022	0.0311	0.0148
10	0.9587	0.0124	0.7707	0.0037	0.0017	0.0185
20	3.9286	0.1461	1.5117	0.0038	0.0133	0.0433
30	3.1490	0.1002	2.3705	0.0009	0.0018	0.0615
40	1.7833	0.2108	3.0245	0.0162	0.0634	0.0900

is changed from -350 to -450 using 10cm intervals. For each case, we employed the virtual poses on a spherical mesh having radius of 400cm and located at the origin of 3D space. The proposed algorithm is employed to recover the pose and the results are shown in Table 6.1. The first column in the table shows the distance of the actual vehicle to the sphere of virtual poses. Remaining columns show the dimension errors for each case.

According to the results, the error rate is very low and does not have a significant effect in the system, even for the worst case, where the error of x dimension is calculated as 5.61cm. This error is acceptable for such a problematic case. These results clearly show that our method is a promising solution for pose recovery problems. Another important conclusion can be derived from Figure 6.5, where total translational and rotational errors are plotted with respect to the distance. Error rate is increasing as the vehicle moves further from the spherical mesh of virtual poses. The



(a) (b)

Figure 6.5: Results of second experiment: a) Total translational error, b) Total rotational error of actual vehicle pose with respect to distance of original position

results suggest that using more levels of virtual poses may be necessary depending on the problem.

In our final experiment, we made a comprehensive test to see how our algorithm performs when the actual pose is located very different directions. For this purpose, we generated an additional spherical mesh for the actual vehicle poses. This is an exhaustive comparison that covers majority of possible cases for the pose recovery problem. Figure 6.6 illustrates the experimental setup, where the reference object is located at the origin of 3D space. First, we generated 484 virtual poses on a spherical mesh centered at the origin, where radius = 400cm (red mesh in the figure). Next, we generated 484 actual poses on a second spherical mesh at the origin with radius = 410cm (green mesh in the figure).

Every actual pose is recovered using the mesh of all virtual poses. Total translational and rotational errors calculated for each pose, and the results are shown

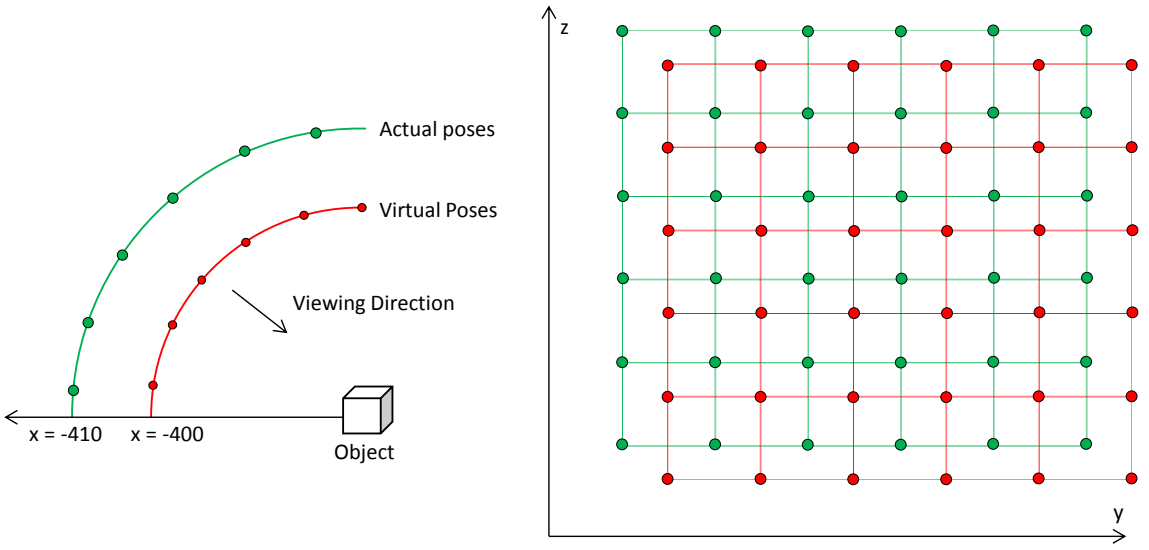


Figure 6.6: A spherical mesh of actual poses are generated and then tested with all virtual poses. Figure on the left shows the location of the viewpoints in x dimension. Figure on the right is the unwrapped version of spherical mesh, and it shows structured placement of the views in y and z dimensions.

by a sphere representation as well as by a 2D *heatmap* representation. A convenient 2D representation is possible since there is already a structured placement of the actual poses as shown in the right image of Figure 6.6.

The error values are visualized on a blue-red color spectrum, where blue color tones represent low errors and red color tones represent high errors. Figure 6.7 a) shows the translational errors on spherical representation, where each cell on the sphere refers to the translational error located at that particular actual pose. Figure 6.7 b) shows the same results on a 2D map (unwrapped version of sphere) that presents the complete visualization. According to the results, the proposed algorithm gives very low translational error for majority of positions. There are only few positions, for which the translational pose parameters could not be calculated. We

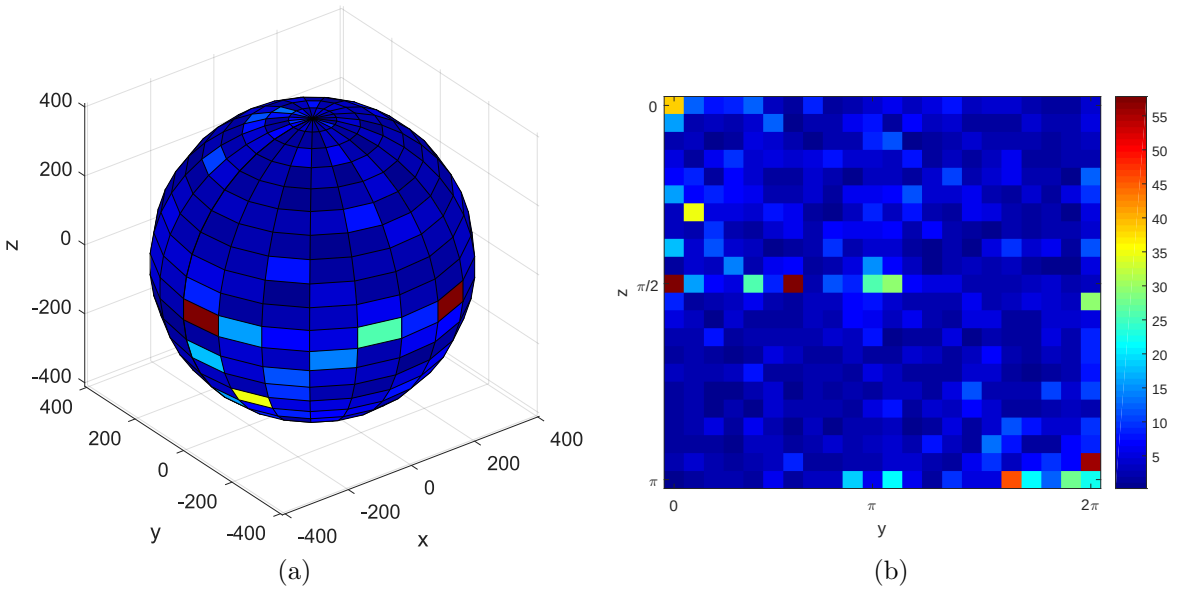


Figure 6.7: Transnational errors for experiment 3. a) Error in spherical mesh representation. b) Error in 2D heat map representation. Color bar on the right shows the color coding for error rates (in cm)

investigated those problematic cases and noticed that sufficient number of 2D features cannot be extracted from those images due to the limited view of the reference object.

Second part of this experiment investigates the rotational error results. Figure 6.8 a) shows the spherical representation of total rotational errors. And Figure 6.8 b) shows 2D map representation of the same experiment. According to the results, very reliable results can be achieved when the actual pose is located at northern hemisphere. Similar to the translational errors, there is a problematic region at the central orbit of the sphere. However, in general, the proposed algorithm can calculate the actual pose with very low rotational error.

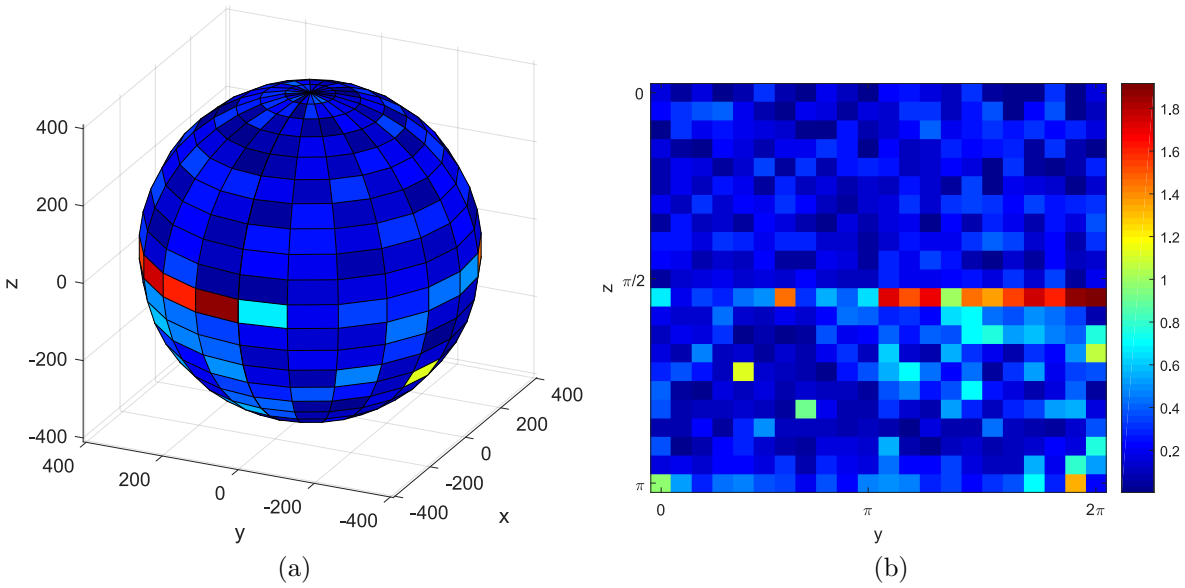


Figure 6.8: Rotational errors for experiment 3. a) Error in spherical mesh representation. b) Error in 2D heat map representation. Color bar on the right shows the color coding for error rates (in radians)

6.7 Summary

This chapter presents a new pose recovery algorithm for pose estimation and trajectory tracking problems. Our method takes the advantage of generating multiple virtual poses to recover the pose when the desired and the actual views of the reference object are not similar. All virtual poses are tested with Mirage pose estimation and a group of common solutions, which has the minimum inter-point distances, are determined as the final solution.

We evaluated our algorithm on 3 different experiments using 2 different reference objects. The results are presented in terms on translational and rotational errors. According to the results, our pose recovery algorithm can be effective for majority of pose failure cases. A small number of experimental cases returned unsatisfactory

results due to the limited views of the object. This problem can be overcome by utilizing higher resolution reference object and including more virtual poses into the calculations.

CHAPTER 7

CONCLUSIONS

This dissertation investigates the problem of camera pose estimation and introduces our novel method Mirage to the research community. Our algorithm is an analytic $O(n)$ time solution that supports multi-camera systems. Compared to other state-of-the-art techniques, our method utilizes the reference camera pose, rather than using 2D-3D point correspondence. Mirage estimates the camera pose by minimizing the 2D projection error between reference and actual pixel coordinates using a linear solution. We conducted our experiments on both simulated and real images, and compared Mirage with 8 well-known techniques in the literature. Promising results have been reached in all experiments and comparisons. The translational and rotational pose errors were lowest for Mirage in the presence of varying noise. Moreover, Mirage was able to solve pose estimation with significantly low error even if there are low number of feature points.

Since Mirage is a platform independent solution, it can be applied to various types of mobile vehicles having spatial motions (e.g., aircraft, spacecraft, underwater robots). However, it can also be easily applied to vehicles with planar motion. We demonstrated two sample applications, and provide real environment and simulation

results that verify the effectiveness of our method. Our method converges to the trajectory with less error in shorter time than other methods and it is still robust in noisy environments. Both simulations and actual experiments successfully confirm the advantages of the proposed methodology.

In the rest of this dissertation, two additional problems of pose estimation (feature mapping and pose recovery) are investigated and alternative solutions are proposed. For the first problem, a GPU based feature mapping methodology is presented. And for the second problem, a new pose recovery algorithm is introduced, where multiple virtual poses are utilized in the case of unmatching actual and desired views. Both proposed solutions are evaluated on variety of test conditions and their performance results are provided.

7.1 Future Work

Based on these promising results, in the future, we aim to improve our system for more realistic and challenging environments. Also, we target to address the limitations of the proposed techniques to achieve more reliable and robust system. Some of the future works in our agenda can be listed as below:

- improve feature extraction stage for low illumination and poor detection or undetected target points
- eliminate the requirement of additional equations for single camera system
- reduce the number of virtual views in Pose Recovery stage

REFERENCES

[1] Carl C Liebe, Kenneth A Brown, Suraphol Udomkesmalee, Curtis W Padgett, Michael P Brenner, Ayanna M Howard, Terry R Wysocky, David I Brown, and Steven C Suddarth. {VIGIL}: a {GPS}-based target-tracking system. *Proc. SPIE*, 3365:10–21, 1998.

[2] X I E Shao-Rong, L U O Jun, R A O Jin-Jun, and GONG Zhen-Bang. Computer vision-based navigation and predefined track following control of a small robotic airship. *Acta Automatica Sinica*, 33(3):286–291, 2007.

[3] Semih Dinc, Farbod Fahimi, and Ramazan Aygun. Vision-based trajectory tracking approach for mobile platforms in {3D} world using {2D} image space. In *ASME International Mechanical Engineering Congress and Exposition, Proceedings (IMECE)*, volume 4 B, San Diego, CA, United states, 2013.

[4] Thomas Petersen. A Comparison of {2D}-{3D} Pose Estimation Methods. *Master's thesis, Aalborg University-Institute for Media Technology Computer Vision and Graphics, Lautrupvang*, 15:2750, 2008.

[5] Tobias Nöll, Alain Pagani, and Didier Stricker. Real-time camera pose estimation using correspondences with high outlier ratios. In *VISAPP 2010: International Conference on Computer Vision Theory and Applications*, pages 381–386, 2010.

[6] Carlos Jaramillo, Ivan Dryanovski, Roberto G Valenti, and Jizhong Xiao. {6-DOF} pose localization in {3D} point-cloud dense maps using a monocular camera. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1747–1752. IEEE, 2013.

[7] Bodo Rosenhahn. *Pose estimation revisited*. PhD thesis, Inst. f{ü}r Informatik und Praktische Mathematik, 2003.

[8] Daniel Grest, Thomas Petersen, and Volker Krüger. A comparison of iterative {2D}-{3D} pose estimation methods for real-time applications. In *Image Analysis*, pages 706–715. Springer, 2009.

[9] Tobias Nöll, Alain Pagani, and Didier Stricker. Markerless camera pose estimation-an overview. In *OASIcs-OpenAccess Series in Informatics*, volume 19. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[10] Luis Ferraz, Xavier Binefa, and Francesc Moreno-Noguer. Very fast solution to the {PnP} problem with algebraic outlier rejection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 501–508. IEEE, 2014.

[11] Yang Guo. A novel solution to the {P4P} problem for an uncalibrated camera. *Journal of mathematical imaging and vision*, 45(2):186–198, 2013.

[12] Jianliang Tang, Wen-Sheng Chen, and Jie Wang. A novel linear algorithm for {P5P} problem. *Applied Mathematics and Computation*, 205(2):628–634, 2008.

[13] Yinqiang Zheng, Yubin Kuang, Shigeki Sugimoto, Kalle Astrom, and Masatoshi Okutomi. Revisiting the {PnP} problem: A fast, general and optimal solution. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2344–2351. IEEE, 2013.

[14] C-P Lu, Gregory D Hager, and Eric Mjolsness. Fast and globally convergent pose estimation from video images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(6):610–622, 2000.

[15] Ralf Vandenhouten, Thomas Kistel, and Ole Wendlandt. A method for optical indoor localization of mobile devices using multiple identifiable landmarks. *Transaction on IoT and Cloud Computing*, 1(1), 2015.

[16] Adnan Ansar and Konstantinos Daniilidis. Linear pose estimation from points or lines. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):578–589, 2003.

[17] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(8):774–780, 1999.

[18] Changhyun Choi and Henrik I Christensen. {3D} pose estimation of daily objects using an RGB-D camera. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3342–3349. IEEE, 2012.

[19] Gim Hee Lee, Bo Li, Marc Pollefeys, and Friedrich Fraundorfer. Minimal solutions for pose estimation of a multi-camera system. In *International Symposium on Robotics Research (ISRR)*, 2013.

[20] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[21] 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.

[22] Joel Hesch, Stergios Roumeliotis, and Others. A direct least-squares {(DLS)} method for {PnP}. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 383–390. IEEE, 2011.

[23] Shiqi Li, Chi Xu, and Ming Xie. A robust {O(n)} solution to the perspective-n-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1444–1450, 2012.

[24] Laurent Kneip, Paul Furgale, and Roland Siegwart. Using multi-camera systems in robotics: Efficient solutions to the {nPnP} problem. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3770–3776. IEEE, 2013.

[25] Chong Chen and Dan Schonfeld. Robust {3D} pose estimation from multiple video cameras. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 541–544. IEEE, 2009.

[26] Henrik Stewenius, Christopher Engels, and David Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[27] Wen Yan Chang and Chu Song Chen. Pose estimation for multiple camera systems. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 262–265. IEEE, 2004.

[28] J Fabian and G M Clayton. Error Analysis for Visual Odometry on Indoor, Wheeled Mobile Robots With {3-D} Sensors. *Mechatronics, IEEE/ASME Transactions on*, 19(6):1896–1906, December 2014.

[29] Ivan Dryanovski, Roberto G Valenti, and Jizhong Xiao. Fast visual odometry and mapping from RGB-D data. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2305–2310. IEEE, 2013.

[30] L Svarm, O Enqvist, M Oskarsson, and F Kahl. Accurate Localization and Pose Estimation for Large {3D} Models. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 532–539, June 2014.

[31] Yi Hu Huang, Man Hu, Hong Lei Chong, Xi Mei Jia, Ji Xiang Ma, and Wen Lon Liu. A Survey of Robot Visual Tracking. *Key Engineering Materials*, 501:577–582, 2012.

[32] B Bethke, M Valenti, and J How. Cooperative vision-based estimation and tracking using multiple {UAV}s. In *Conference of Cooperative Control and Optimization*, Gainesville, FL, January 2007.

[33] R Kelly, R Carelli, O Nasisi, B Kuchen, and F Reyes. Stable visual servoing of camera-in-hand robotic systems. *IEEE/ASME Transactions on Mechatronics*, 5(1):39–48, March 2000.

[34] J T Feddema, C S G Lee, and O R Mitchell. Weighted selection of image features for resolved rate visual feedback control. *IEEE Transactions on Robotics and Automation*, 7(1):31–47, 1991.

[35] G Palmieri, Matteo Palpacelli, Matteo Battistelli, and Massimo Callegari. A Comparison between Position-Based and Image-Based Dynamic Visual Servoings in the Control of a Translating Parallel Manipulator. *Journal of Robotics*, 2012, 2012.

[36] Tao Zhao and Ram Nevatia. Tracking multiple humans in complex situations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1208–1221, 2004.

[37] Huajun Song and Meili Shen. Target tracking algorithm based on optical flow method using corner detection. *Multimedia Tools and Applications*, 52(1):121–131, 2011.

[38] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean-Marc Lavest. Monocular Vision for Mobile Robot Localization and Autonomous Navigation. *Int. J. Comput. Vision*, 74(3):237–260, September 2007.

[39] Éric Marchand and François Chaumette. Virtual Visual Servoing: a framework for real-time augmented reality. In *Computer Graphics Forum*, volume 21, pages 289–297. Wiley Online Library, 2002.

[40] G L Mariottini, G Oriolo, and D Prattichizzo. Image-based visual servoing for nonholonomic mobile robots using epipolar geometry. *IEEE Transactions on Robotics*, 23(1):87–100, 2007.

[41] Changhyun Choi, Seung-min Baek, Sukhan Lee, and Fellow Member. Real-time {3D} object pose estimation and tracking for natural landmark based visual servo. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3983–3989, 2008.

[42] W MacKunis, N Gans, A Parikh, and W E Dixon. Unified tracking and regulation visual servo control for wheeled mobile robots. *Asian Journal of Control*, 16(3):669–678, May 2014.

[43] Chieh-Chih Wang, Charles Thorpe, Sebastian Thrun, Martial Hebert, and Hugh Durrant-Whyte. Simultaneous Localization, Mapping and Moving Object Tracking. *International Journal of Robotic Research*, 26(9):889–916, September 2007.

[44] Patrick Benavidez and Mo Jamshidi. Mobile robot navigation and target tracking system. In *2011 6th International Conference on System of Systems Engineering*, pages 299–304. IEEE, June 2011.

[45] Ali Elqursh and Ahmed Elgammal. Line-based relative pose estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3049–3056, 2011.

[46] Vincenzo Lippiello, Bruno Siciliano, and Luigi Villani. Position-based visual servoing in industrial multirobot cells using a hybrid camera configuration. *Robotics, IEEE Transactions on*, 23(1):73–86, 2007.

[47] Lisa Gottesfeld Brown. A Survey of Image Registration Techniques. *ACM Comput. Surv.*, 24(4):325–376, December 1992.

[48] Yudong Zhang and Lenan Wu. Rigid Image Registration by PSOSQP Algorithm. *Advances in Digital Multimedia*, 1(1):4–8, 2012.

[49] A Kubias, F Deinzer, T Feldmann, D Paulus, B Schreiber, and Th. Brunner. {2D/3D} image registration on the {GPU}. *Pattern Recognition and Image Analysis*, 18(3):381–389, 2008.

[50] Alexander Köhn, Johann Drexl, Felix Ritter, Matthias König, and Heinz-Otto Peitgen. {GPU} accelerated image registration in two and three dimensions. In *Bildverarbeitung f{ü}r die Medizin 2006*, pages 261–265. Springer, 2006.

[51] Sudipta N Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications*, 22(1):207–217, 2011.

[52] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable Parallel Programming with {CUDA}. *Queue*, 6(2):40–53, March 2008.

[53] David G Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[54] D H Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[55] Yi Chen and Ramazan Savas Aygün. Synthetic Video Generation for Evaluation of Sprite Generation. *International Journal of Multimedia Data Engineering and Management*, 1(2):34–61, January 2010.

[56] Yi Chen and Ramazan Savas Aygün. Synthetic Video Generation with Complex Camera Motion Patterns to Evaluate Sprite Generation. In *2009 11th IEEE International Symposium on Multimedia*, pages 657–662. IEEE, 2009.

[57] Hannes Fassold and Jakub Rosner. A real-time {GPU} implementation of the SIFT algorithm for large-scale video analysis tasks. In *IS&T/SPIE Electronic Imaging*, page 940007. International Society for Optics and Photonics, 2015.

[58] Simon Leonard. *Learning Feed-Forward Control for Vision-Guided Robotics*. {PhD} thesis, University of Alberta, 2008.

[59] Chris Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. *Photogrammetric computer vision*, 2:124–131, 2006.

[60] SašO Blaič. A novel trajectory-tracking control law for wheeled mobile robots. *Robot. Auton. Syst.*, 59(11):1001–1007, November 2011.

[61] Y Kanayama, Y Kimura, F Miyazaki, and T Noguchi. A stable tracking control method for an autonomous mobile robot. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 384 –389 vol.1, May 1990.

[62] F Chaumette and S Hutchinson. Visual servo control, {P}art {I}: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, December 2006.

[63] Semih Dinc, Farbod Fahimi, and Ramazan Aygun. Mirage: An {O(n)} time Analytical Solution to {3D} Camera Pose Estimation with Multi-Camera Support. *Robotica (Submitted)*, 2016.

[64] Daniel F Dementhon and Larry S Davis. Model-based object pose in 25 lines of code. *International journal of computer vision*, 15(1-2):123–141, 1995.

[65] Alexandre Krupa, Jacques Gangloff, Christophe Doignon, Michel F De Mathelin, Guillaume Morel, Joël Leroy, Luc Soler, and Jacques Marescaux. Autonomous 3-d positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. *Robotics and Automation, IEEE Transactions on*, 19(5):842–853, 2003.

[66] Mustafa Özuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):448–461, 2010.

[67] Albert Diosi, Siniša Šegvić, Anthony Remazeilles, and François Chaumette. Experimental evaluation of autonomous driving based on visual memory and image-based visual servoing. *Intelligent Transportation Systems, IEEE Transactions on*, 12(3):870–883, 2011.

[68] Thomas Mörwald, Johann Prankl, Michael Zillich, and Markus Vincze. Advances in real-time object tracking. *Journal of Real-Time Image Processing*, 10(4):683–697, 2015.

[69] Ruye Wang and Herbert Freeman. Object recognition based on characteristic view classes. In *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, volume 1, pages 8–12. IEEE, 1990.

[70] Anan Liu, Zhongyang Wang, Weizhi Nie, and Yuting Su. Graph-based characteristic view set extraction and matching for 3d model retrieval. *Information Sciences*, 320:429–442, 2015.

[71] Arthur R Pope and David G Lowe. Learning object recognition models from images. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, pages 296–301. IEEE, 1993.

[72] Adrien Theetten, Jean-Philippe Vandeborre, and Mohamed Daoudi. Determining characteristic views of a 3d object by visual hulls and hausdorff distance. In *3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on*, pages 439–446. IEEE, 2005.

[73] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.

[74] Tarik Filali Ansary, Mohamed Daoudi, and Jean-Philippe Vandeborre. A bayesian 3-d search engine using adaptive views clustering. *Multimedia, IEEE Transactions on*, 9(1):78–88, 2007.

[75] Qizhen He, Zhiwu Lu, and Horace HS Ip. View topics: automatically generated characteristic view for content-based 3d object retrieval. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, page 4. ACM, 2009.