12-2011

A Comparison and Evaluation of Three Different Pose Estimation Algorithms In Detecting Low Texture Manufactured Objects

Robert Kriener
Clemson University, rkriene@clemson.edu

A Comparison and Evaluation of Three Different Pose Estimation Algorithms In Detecting Low Texture Manufactured Objects

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

by
Robert Charles Kriener
Dec 2011

Abstract

This thesis examines the problem of pose estimation, which is the problem of determining the pose of an object in some coordinate system. Pose refers to the object's position and orientation in the coordinate system. In particular, this thesis examines pose estimation techniques using either monocular or binocular vision systems.

Generally, when trying to find the pose of an object the objective is to generate a set of matching features, which may be points or lines, between a model of the object and the current image of the object. These matches can then be used to determine the pose of the object which was imaged. The algorithms presented in this thesis all generate possible matches and then use these matches to generate poses.

The two monocular pose estimation techniques examined are two versions of SoftPOSIT: the traditional approach using point features, and a more recent approach using line features. The algorithms function in very much the same way with the only difference being the features used by the algorithms. Both algorithms are started with a random initial guess of the object's pose. Using this pose a set of possible point matches is generated, and then using these matches the pose is refined so that the distances between matched points are reduced. Once the pose is refined, a new set of matches is generated. The process is then repeated until convergence, i.e., minimal or no change in the pose. The matched features depend on the initial pose, thus the algorithm's output is dependent upon the initially guessed pose. By starting the algorithm with a variety of different poses, the goal of the algorithm is to determine the correct correspondences and then generate the correct pose.

The binocular pose estimation technique presented attempts to match 3-D point data from a model of an object, to 3-D point data generated from the current view of the object. In both cases the point data is generated using a stereo camera. This algorithm attempts to match 3-D point triplets in the model to 3-D point triplets from the current view, and then use these matched triplets to obtain the pose parameters that describe the object's location and orientation in space.

The results of attempting to determine the pose of three different low texture manufactured objects across a sample set of 95 images are presented using each algorithm. The results of the two monocular methods are directly compared and examined. The results of the binocular method are examined as well, and then all three algorithms are compared. Out of the three methods, the best performing algorithm, by a significant margin, was found to be the binocular method. The types of objects searched for all had low feature counts, low surface texture variation, and multiple degrees of symmetry. The results indicate that it is generally hard to robustly determine the pose of these types of objects. Finally, suggestions are made for improvements that could be made to the algorithms which may lead to better pose results.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pose estimation is the process of determining the pose of an object in space. The pose of an object is the object's translation and orientation, i.e., roll, pitch, and yaw in some coordinate system. This thesis will examine the problem of pose estimation using vision systems.

1.1 Motivation

Pose estimation is an important problem in autonomous systems. In the case of an industrial robot attempting to interact with or avoid an object, the robot must know where the object is located and how it is oriented. Typically, the problem of locating objects for grasping is avoided by ensuring that objects are always at the same location through some sort of tooling system. The objects with which the robot will interact are loaded into the tooling system by humans before the robot is able to interact with them. If the robot were capable of identifying where the objects were via its own pose estimation system it could, in theory, load the parts into the system itself. One reason why this technology is not prevalent in industry currently is that many manufactured objects, such as solid metal/plastic components, do not have many readily detectable features.

Pose estimation is also important in mobile robotic systems. If a robot is to retrieve an object it must be able to locate it in space first. Pose estimation can also be used in mobile robot localization. If the location of a known landmark can be determined then the robot can estimate its own position in space, much like how a human would look for a familiar building or sign to identify where they are.

1.2 Related Work

Many researchers have studied the pose estimation problem and developed algorithms to find the pose of objects.

Table 1.1 shows the relationship of a few of the pose estimation algorithms which will be discussed, specifically including the algorithms which will be examined in this thesis. Table 1.2 shows some of the different types of pose estimation problems which will be discussed and the common assumptions associated with them. The three categories of pose estimation problems shown in the table are pose estimation, pose tracking, and AR pose estimation techniques. The first category, pose estimation, addresses the problem of identifying an objects pose in space w.r.t. the camera, using a single image of the object. Pose tracking is the problem of tracking an objects pose from frame to frame in a video sequence, which is equivalent to finding the objects precise pose when the approximate pose is already known. The AR pose estimation techniques presented all work only with video sequences, and are related to structure from motion techniques. The AR techniques address the problem of finding the cameras pose in the world. This thesis focuses on the first category of problems, pose estimation.

Monocular Vision		Binocular Vision	
Known Correspondences	Unknown Correspondences	Known Correspondences	Unknown Correspondences
POSIT [10]	SoftPOSIT [8, 9]	Absolute Orientation [24]	Triplet Matching [21]
PnP Methods [18, 23]	RANSAC [12]		

Table 1.1: Classification of a few of the different pose estimation techniques discussed. Each unknown correspondence algorithm depends or builds upon the known correspondence algorithm to the left.

Algorithm Types	Pose Estimation	Pose Tracking	AR Pose Estimation
Algorithms	SoftPOSIT [8, 9] Triplet Matching [21]	RAPiD [20] [27] and [13]	[36] and [29]
Requirements	Model known	Model known Approx pose known	Moving camera
Applied To	Single image	Video or Single Image	Video

Table 1.2: Classifications of the different types of pose estimation algorithms discussed along with their requirements

Pose estimation, when the approximate pose is known, has been widely studied. These algorithms are generally used for pose tracking. In these instances the pose from one image to the next can only vary slightly, thus the approximate pose is known, and the problem is constrained. Some example algorithms for pose tracking include RAPiD [20], a method proposed by Lowe [27], and yet another method by Jurie [13].

Another common application of pose estimation is in augmented reality (AR) systems. These systems use pose to place objects in an image, such that the inserted object appears as if it were actually in the original scene. Often in these applications precise pose is not necessary because there is no physical interaction between the system and the world, and objects only need to appear as if they were actually in a scene. Also since AR is typically applied to video many of the algorithms take advantage of the cameras motion to help with the pose estimation problem. Some example AR pose estimation algorithms include [36, 29]. Lepetit gives a through survey of pose estimators for both AR and pose tracking applications in [25].

This thesis will focus on mathematical and geometrical methods of pose estimation, which rely on matching a model of the object to be found to some sort of image or sensor data. In all of these algorithms the true pose is assumed to lie within a large search space, the approximate pose is not known a priori, and the only image data available is a single image or a pair of stereo images.

One of the most common methods for estimating pose with a model and image data is to extract features from the image, such as lines, corners, or even circles and match the extracted features to the model features. If the correspondences/matches between the features of the model and the image are known the problem becomes nearly trivial.

One common algorithm for pose estimation with known point feature correspondences is POSIT (Pose from Orthogonality and Scaling with ITerations) [10]. This algorithm assumes that feature correspondences are known in advance and will fail when correspondences are incorrect. Other methods of pose estimation with known correspondences include [18, 23, 32, 1]. All of these algorithms are capable of generating pose estimates given a set of point, or in some cases line, correspondences and a cameras calibration matrix.

The POSIT algorithm was later updated to become SoftPOSIT [9] which combines the POSIT algorithm with a correspondence estimation algorithm softassign [15, 38]. This algorithm requires all of the point features in both the model and current image to be provided, along with a guess of the possible pose of the object. The algorithm matches the model and image features and estimates the pose to minimize the distance between all of the matched features. The pose output by the algorithm is dependent upon the initial pose guessed, and the algorithm is not guaranteed to converge. Even in cases where the algorithm does converge there is no way to know that the pose is correct without further evaluation. SoftPOSIT was extended to work with line features [8], but still has many of the same problems as the original SoftPOSIT.

Another well known algorithm for estimating poses with features is RANSAC (RANdom SAmple Consensus) [12]. This algorithm matches, at random, the minimum number of point features from the model to features in the image to estimate a pose. The absolute minimum of matched features is three [18], which will provide up to four feasible pose estimates, while four matched features will yield a single pose estimate. By iterating through the possible sets of matches at random the actual pose can be generated. This algorithm has the advantage that it is guaranteed to yield the correct pose at some point; however, the correct pose must be extracted from all of the poses returned by the algorithm. The algorithm also is exponential (theoretically) in execution time as the number of features increases, making it a bad choice for feature rich scenes.

Some of the most robust pose estimation algorithms currently available [16, 17, 6] make use of Scale Invariant Feature Transform (SIFT) [28] features. These algorithms combine SIFT features with monocular, stereo, or Time of Flight (TOF) cameras to give highly accurate poses for objects. Although these algorithms work well, they are limited to use on highly textured objects. This is due to the fact that they rely on SIFT features which are only present on surfaces with high texture. Therefore, these algorithms are not suitable for use on many manufactured objects which have fairly consistent surfaces such as cardboard boxes, metal components, or plastics. These algorithms would also fail if the surfaces of the objects were changed even when their form remains the same, e.g., if a company redesigned its packaging art or decided to make its products in different colors.

Both SoftPOSIT and RANSAC can be applied to any set of image model point feature correspondences regardless of how they are generated. Besides SIFT, many other popular feature detectors exist including the Harris corner detector [19], SURF [3], FAST [33], and many others. See [31, 35] for a comprehensive review and comparison of common point feature detectors. However, as with SIFT other point features require certain types of surface texture variation to function well. If the object to be detected has few corners or reliable surface features, then there are no reliable features to match. This is true of many manufactured objects. Another drawback to feature based methods is that in order to match features they must first be extracted from the image, and as the image's content becomes increasingly complex the number of false matches and occluded features increases.

All of the pose estimation algorithms discussed up to this point are feature based, in that they require the matching of model and image features as a step in estimating a pose, and thus are restricted to being applied to objects which contain features. Another class of pose estimators uses only range data to estimate an object's pose.

All of these estimators [30],[34],[21] rely only range data, that is (x, y, z) point locations to estimate poses rather than feature extraction. These types of algorithms can work on objects of any shape, color, or texture provided accurate enough depth information can be extracted. Many devices exist which can generate depth information, including: stereo cameras, laser scanners, TOF cameras, sonar, and radar. Thus, these algorithms are not restricted to working only with stereo range data.

1.3 Outline

This paper compares and examines the effectiveness of SoftPOSIT with point features, SoftPOSIT with line features, and a 3-D point triplet matching algorithm in detecting the pose of low texture manufactured objects. The first two algorithms are directly comparable as they both are run on 2-D image data and rely on feature extraction. The third algorithm uses a stereo camera setup to reconstruct the scene's 3-D geometry as a point cloud and then examines this data to extract the pose of the object within. The overall performance of these algorithms will be compared over a sample set of images, but the reader should keep in mind the differences between the algorithms when comparing their performance.

Chapter 2 presents some background content including: basic concepts of imaging, 3-D reconstruction, and pose estimation with known correspondences. Chapter 3 examines in detail the three pose estimation algorithms presented in this thesis. Chapter 4 presents the experiments conducted to examine the effectiveness of the three pose estimation algorithms studied along with the experimental results. Finally Chapter 5 presents a review of the experimental findings along with possible future improvements and modifications that can be made.

Chapter 2

Background

2.1 Notation

All points in 3-D space will be defined by the capital letter P and a superscript letter C, M, or W will designate the frame of reference of the point . The letter C indicates the point is represented in the camera coordinate system, M indicates the point is represented with respect to the model coordinate system, and W indicates the point is represented with respect to the world coordinate system. Points will be enumerated by subscript numbers, or in the general case a subscript i. P_2^M for example would correspond to object point 2 in the model's coordinate system. The coordinates of a point P will be expressed by capital letters (X, Y, Z). Figure 2.1 gives an example of 3-D points expressed in different frames.

All image points will be designated by the lower case letter p. In the case of two cameras with separate images, superscript C_i's will be used to indicate the image which the point belongs to. All image points will be enumerated with subscript numbers. For example $p_3^{C_1}$ would indicate the third image point in the image generated by camera 1. The coordinates of a point p will be expressed as lowercase letters (x, y).

For both image points p and 3-D points P the homogeneous representation of the points will often need to be used. The homogeneous form is achieved by appending a 1 to the coordinates so that

$$\mathbf{p} = \lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{P} = \lambda \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The homogeneous form allows easier expressions of rotations and translations of points. Note the lambda term is included because homogeneous coordinates are scale invariant. When the last coordinate of the points is 1 the coordinates are referred to as normalized homogeneous coordinates. In any case the coordinate form (X, Y, Z) or homogeneous form $[X, Y, Z, 1]^T$ of points may be used throughout the thesis when referring to points.

It has been shown that points have a homogeneous form which is generated by appending a 1 to the coordinates. However, homogeneous coordinates also allow a alternate way to express lines. Specifically a line ℓ can be described in a Euclidean sense by the equation $ax + by + c = 0$ or in homogeneous form by $\ell = [a, b, c]$. The previous equation can then be expressed in a homogeneous sense by the equation $[a, b, c][x, y, 1]^T = 0$.

Matrices and vectors will both be indicated by **bold face** text. $\mathbf{R} \in \mathbb{R}^{3\times3}$ will be a rotation matrix which can be expressed as

$$\mathbf{R_C^M} = \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2} \\ \mathbf{r_3} \end{bmatrix}, \quad \mathbf{r_i} \in \mathbb{R}^{1\times3} \tag{2.1}$$

where $\mathbf{r_1}$ is the unit vector of the camera frame's X axis $\mathbf{\hat{e}_x^C}$ expressed in terms of the unit vectors of the model frame $\mathbf{\hat{e}_x^M}$, $\mathbf{\hat{e}_y^M}$, and $\mathbf{\hat{e}_z^M}$. Similarly $\mathbf{r_2}$ and $\mathbf{r_3}$ are the unit vectors $\mathbf{\hat{e}_y^C}$ and $\mathbf{\hat{e}_z^C}$ expressed in terms of the model coordinate systems unit vectors. This rotation matrix completely describes the rotation from the model to the camera coordinate system and satisfies

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I} \quad \text{and} \quad \det(\mathbf{R}) = 1$$

Note that the superscript on $\mathbf{R}_C^M$ indicates the source coordinate system and the subscript the destination coordinate system. So $\mathbf{R_C^M}$ is the rotation matrix that converts coordinates in the model frame to coordinates in the camera coordinate frame, assuming the origins of the two systems coincide. In the case where the origins of the two systems do not coincide an additional translation $\mathbf{T}_C^M \in \mathbb{R}^3$ must be applied to the points to shift them to the correct location. Where $\mathbf{T}_C^M$ is the vector from the origin of the camera coordinate system to the origin of the model coordinate system in the camera's frame of reference.

To convert a point from one coordinate system to another, the rotation and translation transforms can be applied to the point to generate the new coordinates. For example to convert point P^M from the model frame to the camera frame the following equation would be used

$$\mathbf{P}^C = \mathbf{R_C^M}\mathbf{P^M} + \mathbf{T}_C^M$$

This equation first rotates the point then shifts it to the proper position in the camera's frame.

Using homogeneous coordinates this transform can be expressed as a single homogeneous rigid body transform of the form:

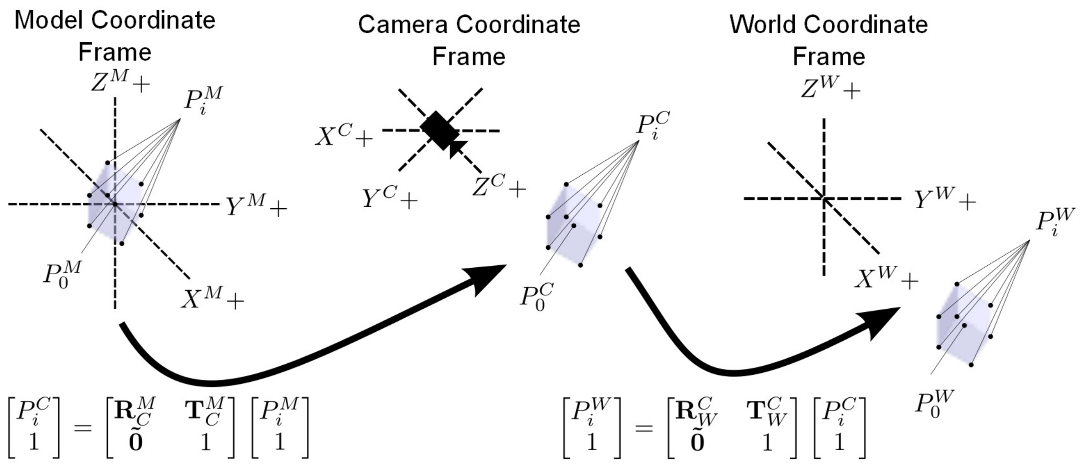
$$\begin{bmatrix} P^C \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_C^M & \mathbf{T}_C^M \\ \tilde{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} P^M \\ 1 \end{bmatrix}$$

This rigid body transform allows a set of points belonging to an object which are expressed in a model coordinate frame to be expressed in the camera systems coordinate frame.

2.2 What is meant by pose?

As discussed in Chapter 1, the goal of pose estimation is to generate a pose that describes an object's position and orientation in space with respect to some coordinate system. Pose in this instance will be a translation $\mathbf{T}_C^M$ and rotation $\mathbf{R}_\mathbf{C}^\mathbf{M}$ which fully describes the position and orientation of an object in the camera's frame of reference. If the relationship between the camera's coordinate system and a world coordinate system is known ($\mathbf{R}_W^C$,$\mathbf{T}_\mathbf{W}^\mathbf{C}$), the overall pose of the object in the world can be determined see (2.2). Figure 2.1 shows the relationship of three coordinate systems.

In Figure 2.1, P_i^M are the object points expressed in the model coordinate frame, and P_0^M is the centroid of the model and the origin of the model coordinate system. P_i^C are the object points expressed in the camera coordinate frame, and P_0^C is the centroid of the object in the camera's coordinate frame. P_i^W are the object points in the world coordinate frame. The equation relating the coordinates of the



Model Coordinate Frame
Z^M+
P_i^M
Y^M+
P_0^M
X^M+
Camera Coordinate Frame
X^C+
Y^C+
Z^C+
P_i^C
P_0^C
World Coordinate Frame
Z^W+
Y^W+
X^W+
P_i^W
P_0^W
$\begin{bmatrix} P_i^C \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_C^M & \mathbf{T}_C^M \\ \tilde{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} P_i^M \\ 1 \end{bmatrix}$
$\begin{bmatrix} P_i^W \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_W^C & \mathbf{T}_W^C \\ \tilde{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} P_i^C \\ 1 \end{bmatrix}$

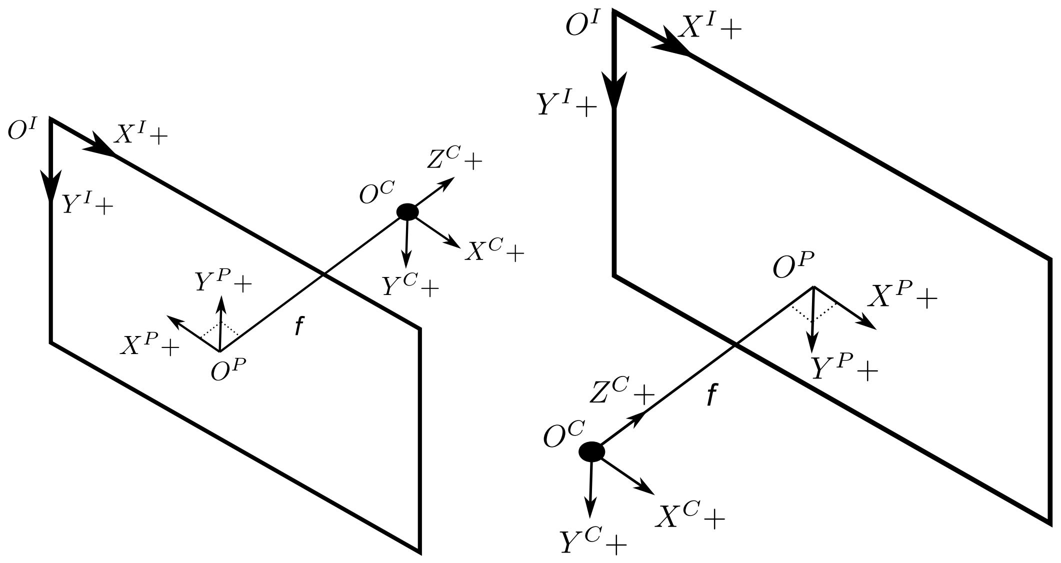
Figure 2.1: The relationship of the model, camera, and world coordinate systems.

points in the model frame to the points in the world frame is given by

$$\mathbf{P_i^W} = \begin{bmatrix} \mathbf{R}_W^C & \mathbf{T}_W^C \\ \tilde{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_C^M & \mathbf{T}_C^M \\ \tilde{\mathbf{0}} & 1 \end{bmatrix} \mathbf{P_i^M} \tag{2.2}$$

If it is assumed that the camera's relationship to the world is constant, i.e., the camera does not move or the camera and world frame move synchronously, then the transform relating the camera coordinate system and world coordinate system $(\mathbf{R}_W^C, \mathbf{T}_W^C)$ can be calculated once and will remain constant.

Assuming the camera to world transform is known the goal of pose estimation is to find the rotation matrix $\mathbf{R}_C^M$ and translation vector $\mathbf{T}_C^M$ which will locate the object in the camera frame of reference.



(a) Camera model (b) Frontal camera model

Figure 2.2: Mathematically identical camera models

2.3 How does imaging work?

2.3.1 Modeling a camera

The simplest model to examine the behavior of a camera is the pinhole model. This model treats the camera as a single point and a plane. In an actual camera light in the world travels through a lens which focuses the light through a point and onto film or a CCD. The point in the model is equivalent to the center of the lens, the optical center, and the plane is equivalent to the CCD or film in a camera.

The optical center of the camera will be defined as the origin of the camera's coordinate system, O^C. The Z-axis will be defined by the location where the plane normal passes through O^C, and the X and Y axes will be parallel to the image plane with the X-axis left to right and the Y-axis pointing up and down as in Figure 2.2(a).

This geometry generates an inverted image which digital cameras correct by inverting the image data. To achieve the same result with the model, the imaging plane can be moved in front of the focal point as in Figure 2.2(b). Figure 2.3 illustrates the projection of a point onto the image plane for both models. Notice the frontal plane model gives a non-inverted image.

The length of the perpendicular line between the camera and the optical center is the focal length *f*. It is related to the length between the CCD/film and the lens of a camera. The units used for the length will determine the correspondence between pixel lengths and real world lengths. In this thesis all lengths will be in meters. Thus, f has units of pixels/meters.

2.3.2 The geometry of image formation

Using this frontal model the geometry of how an image is formed can be explained. Figure 2.3 shows the projection of a point onto the image plane for both the "real" and frontal camera models. Notice that 2 similar triangles are formed with lengths Y, Z and y, f. Using the relationship of similar triangles the y coordinate and similarly the x coordinate of the projected point $p^P = (x, y)$ can be calculated. Note that P indicates the coordinates are with respect to the projected image coordinate system. The relationship between the two coordinate systems is as follows.

$$x^P = f\frac{X^C}{Z^C} \quad y^P = f\frac{Y^C}{Z^C} \tag{2.3}$$

At this stage the transform necessary to project points from the camera's coordinate system onto the image plane and into the projected image coordinate system has been shown. Since images typically assume that the origin of the image coordinate system is at the top left of the image an additional transform must be applied to these projected points coordinates to shift the origin to the top left. This transform is a simple translation in the x and y coordinates of the image. With the

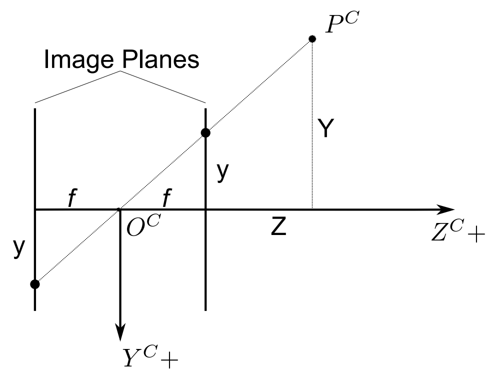


Figure 2.3: The projection of a point onto the image plane

previous transformation equation (2.3) the change was from camera coordinates in meters to image coordinates in pixels; however, this transform is within the same space thus the translations units are in pixels. Specifically the translation $\mathbf{T}_{\mathbf{I}}^{\mathbf{P}} = [u_o, v_o]$ where u_o, v_o are the coordinates of the center of the image in pixels w.r.t. the image coordinate system origin O^I.

Thus, the complete transform to convert from camera coordinates to image coordinates is given by the equation

$$x^I = f\frac{X^C}{Z^C} + u_o^I \quad y^I = f\frac{Y^C}{Z^C} + v_o^I \tag{2.4}$$

This equation can be simplified by using homogeneous coordinates and some simple matrix algebra.

$$\underbrace{\lambda \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p^I}} = \underbrace{\begin{bmatrix} f & 0 & u_o & 0 \\ 0 & f & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{P^C}} \tag{2.5}$$

The λ factor is in the equation to ensure that the result of the matrix multiplication is indeed a normalized homogeneous coordinate i.e. its third coordinate is one. This factor appears because any point along a ray from the optical center through a pixel on the image plane will project down to that pixel.

The **H** matrix in equation (2.5) is commonly referred to as the camera matrix or calibration matrix and this is its simplest form. In reality the two f terms are slightly different because of varying pixel dimensions in the X and Y directions. Additionally, there is a skew term which can be added to the matrix. There are also distortion terms which can be used to correct lens distortion in the projection, but for most simple applications all of the distortions can be ignored along with the higher complexity terms in the camera matrix.

Without in-depth knowledge of the construction of the camera it is not possible to know the value of f, u_o, or v_o. Thus methods have been developed to determine these parameters through calibration. With proper calibration all of the parameters in the calibration matrix along with the distortion terms can be estimated with a high level of accuracy.

2.4 Camera calibration

There exist many different methods for performing camera calibration. In this implementation the built-in method, cv::calibrateCamera, in the OpenCV library was used. Camera calibration requires a series of differing views of a calibration pattern, in this case a checkerboard, to be fed into the function along with the dimensions of the checkers on the pattern. The checkerboard pattern makes it easy to find the corners of the squares and if the dimensions of the squares are known a model for the checkerboard can be easily generated. With a known model of the calibration pattern and with the detected squares of the imaged calibration pattern, correspondences

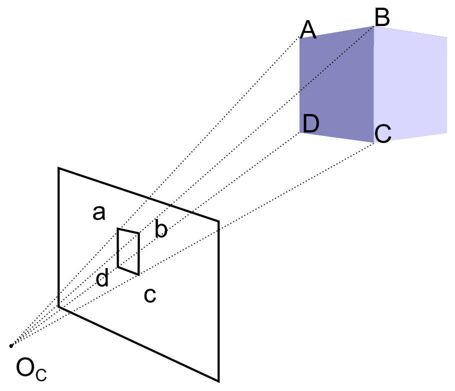


Figure 2.4: Estimating pose with known correspondences

between the detected image corners and the model corners can be generated. Using these correspondences a homography matrix can be generated that represents the transform the model goes through to create the image. Using a number of these homographies from different images of the calibration pattern, the parameters of the calibration matrix can be empirically determined. Thus the matrix $\mathbf{H}$ can be determined. A more detailed explanation of the calibration process can be found in [40]. A survey of calibration methods and their approaches can be found in [2].

2.5 Pose From Correspondences

It has been shown that any point in space which lies along a ray that intersects the image plane can project down to the plane at that intersection point. If a series of correspondences between a geometric model and an image of that model can be determined, then a pose estimate which aligns the model points in space along the rays passing through the image points can be generated. Figure 2.4 shows a possible pose generated from four correspondences between an image and a model.

Normally four points is sufficient to recover the correct actual pose of the object as long as the four points are not co-planar. In this example, Figure 2.4, the four points are co-planar. For three non co-planar points or four co-planar points, there are multiple poses for the object which will result in the same image. Using four non-coplanar points avoids this problem.

There are many methods to solve for the pose of an object given a model and a set of image correspondences including the P3P (Perspective 3 Point) [18] problem, POSIT [10], and others [23][1]. In this paper the focus will be on the POSIT algorithm as it is an integral part of the SoftPOSIT algorithm.

2.6 POSIT

2.6.1 Overview of POSIT

POSIT [10] uses known image model point correspondences and a known camera calibration to reconstruct the pose of an object. The goal of POSIT is to relate a model's geometry, a scaled orthographic projection of the model, and an actual image of the modeled object to recover all of the parameters which define the pose.

The algorithm initially assumes that the object is at some depth which is relatively far away from the camera as compared to the depth of the actual object its self, and then fits the pose as best it can at this depth by trying to align image and model features. This is the POS (Pose from Orthogonality and Scaling) algorithm. Based upon the error of the fit a better depth estimate is created and the process is repeated. The repeated application of the POS algorithm is the POSIT (POS with ITerations) algorithm. After iteratively improving the pose, the algorithm will eventually converge and return the pose of the object.

2.6.2 Scaled Orthographic Projection

The Scaled Orthographic Projection (SOP) of a model is an approximation of the perspective transform. In fact, the SOP is a special case of the perspective transform where all of the points in the scene of an image lie in a plane parallel to the image plane.

To generate the scaled orthographic projection all of the points in a scene are orthogonally projected onto a plane parallel to the image plane at at distance Z_o from the camera's origin. Then these point coordinates are scaled by Z_o/f to generate the SOP.

In POSIT the model undergoes the SOP to generate a simulated image. If there are N number of model points $P_0^M ... P_N^M \in \mathbb{R}^{3\times 1}$ where P_0^M coincides with the origin of the model coordinate system then a perspective projection of these points would have the form of the equation in (2.3), i.e.

$$x_i^P = f\frac{X_i^C}{Z_i^C} \quad y_i^P = f\frac{Y_i^C}{Z_i^C}$$

Assuming the plane for the orthographic projection is located at the z coordinate of P_0^M in the camera coordinate system i.e. $Z_o = Z_0^C$ then the SOP image coordinates p_i' of a point P_i^M are given by

$$x_i' = f\frac{X_i^C}{Z_0^C} \quad y_i' = f\frac{Y_i^C}{Z_0^C}$$

Combining these forms, a more desirable form of the SOP image coordinates p_i' which relates the known image coordinates and desired model coordinates in the camera's coordinate system is generated.

$$x'_i = x_0^P + s(X_i^C - X_0^C) \quad y'_i = y_0^P + s(Y_i^C - Y_0^C) \tag{2.6}$$

$$s = \frac{f}{Z_0^C}$$

2.6.3 POS

The prior definition of the rotation matrix (2.1) will be used as the unknown rotation matrix $\mathbf{R}_C^M$ we seek to find with the POSIT algorithm.

Using this notation the pose of the object can be fully recovered with the parameters $\mathbf{r_1}$,$\mathbf{r_2}$,$\mathbf{r_3}$,and the coordinates of P_0^C.

The following two equations relate the known parameters the model and image features to the unknown parameters $\mathbf{r_1}$,$\mathbf{r_2}$, and Z_0^C.

$$(P_i^M - P_0^M) \cdot \frac{f}{Z_0^C}\mathbf{r_1} = x_i^P(1 + \epsilon_i) - x_0^P \tag{2.7}$$

$$(P_i^M - P_0^M) \cdot \frac{f}{Z_0^C}\mathbf{r_2} = y_i^P(1 + \epsilon_i) - x_0^P \tag{2.8}$$

where ϵ_i is defined as

$$\epsilon_i = \frac{f}{Z_0^C}(P_i^M - P_0^M) \cdot \mathbf{r_3} \tag{2.9}$$

and $\mathbf{r_3}$ is calculated from taking $\mathbf{r_1} \times \mathbf{r_2}$ since $\mathbf{R}_C^M$ is required to have orthogonal rows.

These equations relate the image coordinates of the SOP and the actual perspective projection to the model, with the coordinates of the SOP expressed in terms of the perspective projection. Looking with more detail it can be shown that $x'_0 = x^P_0$ because the plane used in generating the SOP is located at the Z coordinate of P^M_0. Thus, the SOP and perspective projection of P^M_0 are the same point. Examining the term $x^P_i(1+\epsilon_i)$, it can be shown that this term is the image coordinate p'_i of the SOP of P^M_i full proof of this fact is show in [10]. Intuitively this makes sense because ϵ_i is the ratio of the distance between the Z coordinates of the model points, and the distance between the camera's origin and the orthographic projection plane. Thus, if an object is far away ϵ_i is small and $x_i \approx x'_i$ but when an object is close ϵ_i is large and the disparity between x_i and x'_i increases thus the coordinate must be shifted a greater distance. The left hand side of equation (2.7) is the projection of a vector in the model coordinate system onto the vector $\mathbf{r_1}$ which is the image X-axis expressed in the model coordinate system, this projection is then scaled by the SOP scaling factor. Thus, the result of the left hand side of the equation is the length between the two model points P^M_0 and P^M_i along the X-axis in the SOP coordinate system, which is equal to the distance between the points $x'_i = x^P_i(1+\epsilon_i)$ and $x'_0 = x^P_0$.

Since $\mathbf{r_1}$, $\mathbf{r_2}$, Z^C_0 will be chosen to optimize the fit of all of N model points the equations (2.7) (2.8) will need to be re-written in a form which lends it self to developing a linear system. The equations are rewritten

$$(P^M_i - P^M_0) \cdot \mathbf{I} = \xi_i$$

$$(P^M_i - P^M_0) \cdot \mathbf{J} = \eta_i$$

with

$$\mathbf{I} = \frac{f}{Z^C_0}\mathbf{r_1} \quad \mathbf{J} = \frac{f}{Z^C_0}\mathbf{r_2} \quad \xi_i = x^P_i(1+\epsilon_i) - x^P_0 \quad \eta_i = y^P_i(1+\epsilon_i) - y^P_0 \tag{2.10}$$

These equations can be rewritten to a linear system of the form

$$\mathbf{AI} = \mathbf{x}' \quad \mathbf{AJ} = \mathbf{y}' \tag{2.11}$$

$\mathbf{A}^{(N-1)\times 3}$ is the matrix of model points $P^M_{1\ldots N}$ in the model coordinate system which does not change. $\mathbf{I}$ is the same as in equation 2.10 while $\mathbf{x'}^{(N-1)\times 1}$ and $\mathbf{y'}^{(N-1)\times 1}$ are vectors containing ξ_i and η_i respectively.

The equation (2.11) can be solved in a simple least squares sense to give values for $\mathbf{I}$ and $\mathbf{J}$. Looking back at the definitions of $\mathbf{I}$ and $\mathbf{J}$ it can be seen that $\mathbf{r_1}$ and $\mathbf{r_2}$ can be recovered by normalizing $\mathbf{I}$ and $\mathbf{J}$. The amount by which are $\mathbf{r_1}$ and $\mathbf{r_2}$ are scaled is $\frac{f}{Z_0^C}$. Thus the average of the magnitude of $\mathbf{I}$ and $\mathbf{J}$ gives a good estimate of $s = \frac{f}{Z_0^C}$. Since f is known in the algorithm Z_0^C can be readily calculated. The last parameters to be calculated are $\mathbf{r_3}$ and ϵ_i. $\mathbf{r_3}$ can be quickly generated by taking $\mathbf{r_1} \times \mathbf{r_2}$ and ϵ_i is now dependent on already calculated parameters.

2.6.4 POS with ITerations

By using the results of the first application of POS to generate new values of ϵ_i, and then repeating the POS algorithm with the new ϵ_i values the POSIT algorithm is developed.

Up to now the POSIT algorithm has been developed. Now the problem of how to start the algorithm is addressed. After all, the linear system (2.11) requires an initial value for ϵ_0. Making the assumption that the Z dimensions of the object are small, compared to the distance to the object from the camera, the algorithm can be started with $\epsilon_0 = 0$. This initial seeding of the algorithm works well when the assumption is true, but can cause the algorithm to diverge from the correct answer when the assumption is false. Because of this the POSIT algorithm is only useful when the assumption is indeed true, which for many real applications this assumption is acceptable.

If the POSIT algorithm is run in a loop until $\left|\epsilon_{i(n)} - \epsilon_{i(n-1)}\right| < \delta$ then the algorithm can be considered to have converged. Once the algorithm has converged the pose parameters can be recovered from the values returned by POSIT. $\mathbf{R}_C^M$ can be recovered from $\mathbf{r_1}$, $\mathbf{r_2}$, and $\mathbf{r_3}$ and the translation vector $\mathbf{T}_C^M = \left[p_0^P/s, s/f\right]$, which is the image point p_0^P projected back into space at a depth Z_0^C. Now the objects pose has been reconstructed using the model coordinates, corresponding image coordinates, and the camera's focal length.

2.7 3D Reconstruction From Stereo Images

One last topic to explore related to the algorithms which will be presented is 3D reconstruction from stereo images. The goal of 3-D reconstruction is to re-project an images points back into space at the appropriate depth so that a 2-D image can be used to recreate a 3-D point cloud which approximates the continuous surface which was imaged. If there are two cameras in a world looking at the same object then each camera will project the same point P in the object down to different points, $p_i^{C_1}$ and $p_i^{C_2}$, in each camera's image coordinate systems. Using the camera models as shown in Figure 2.5, for each camera the line of sight from the camera origin through the image plane at the pixel corresponding to the model point P can be reconstructed. If noise is non existent, then in theory, both of the lines of sight rays from both cameras will intersect at the object point in space. If the distance and orientation between the two cameras is known then the location of the object point in space w.r.t. the cameras can be determined via triangulation.

Two major assumptions are made above which must be explored further. First

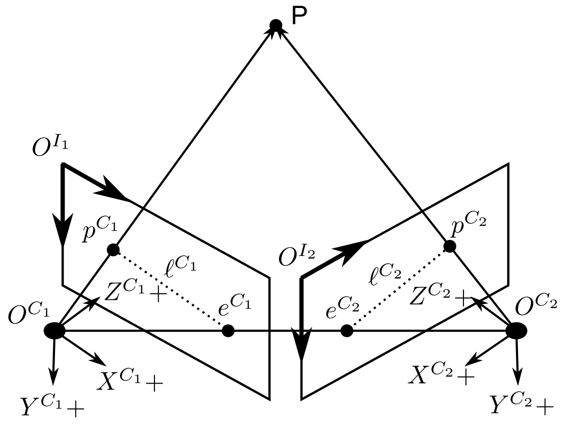


Figure 2.5: Example of two cameras in space

it was assumed that the rotation $\mathbf{R}_{C2}^{C1}$ and translation $\mathbf{T}_{C2}^{C1}$ between the two cameras was known. In reality this is almost never the case. Thus this relationship must be determined via some method. Thankfully due to the geometry of two cameras looking at a point, the rotation and translation between the two cameras can be calculated relatively easily.

2.7.1 Finding the Essential Matrix

Looking at Figure 2.5 the line drawn between the two cameras origins is called the baseline, and it intersects each cameras image plane at e^{C_1} and e^{C_2}. These two points are referred to as the epipolar points and the lines between $e^{C_1}, p_i^{C_1}$ and $e^{C_2}, p_i^{C_2}$ are epipolar lines $\ell_{\mathbf{i}}^{\mathbf{C_1}}$,$\ell_{\mathbf{i}}^{\mathbf{C_2}}$. The baseline is the common edge of the triangle formed between the cameras origins and any point in space P_i. Any point lying on this triangle in space will project onto the image plane of camera one somewhere along the line between $p_i^{C_1}$ and e^{C_1} and image plane of camera two somewhere along $p_i^{C_2}$ and e^{C_2}.

The essential matrix $\mathbf{E}$ captures the relationship of a normalized homogeneous image point $p_i^{C_1}$ and its epipolar line $\ell_{\mathbf{i}}^{\mathbf{C_1}}$ in image one to the corresponding epipolar line $\ell_{\mathbf{i}}^{\mathbf{C_2}}$ in image two, specifically $\ell_{\mathbf{i}}^{\mathbf{C_2}} = \mathbf{E}\mathbf{p}_{\mathbf{i}}^{\mathbf{C_1}}$ and $\ell_{\mathbf{i}}^{\mathbf{C_1}} = \mathbf{E}^T\mathbf{p}_{\mathbf{i}}^{\mathbf{C_2}}$. Looking at point P_i in Figure 2.5, $P_i^{C_1}$ are the coordinates of P_i in camera ones coordinate system. The coordinates of $P_i^{C_2}$ are $\mathbf{P}_{\mathbf{i}}^{\mathbf{C_2}} = \mathbf{R}_{C2}^{C1}\mathbf{P}_{\mathbf{i}}^{\mathbf{C_1}} + \mathbf{T}_{C2}^{C1}$. Converting to normalized homogeneous image coordinates this relationship becomes

$$\lambda_2(\mathbf{p}_{\mathbf{i}}^{\mathbf{C_2}}) = \mathbf{R}_{C2}^{C1}\lambda_1(\mathbf{p}_{\mathbf{i}}^{\mathbf{C_1}}) + \mathbf{T}_{C2}^{C1}$$

Multiplying this equation by $\hat{\mathbf{T}}$ gives

$$\hat{\mathbf{T}}\lambda_2(\mathbf{p}_{\mathbf{i}}^{\mathbf{C_2}}) = \hat{\mathbf{T}}\mathbf{R}_{C2}^{C1}\lambda_1(\mathbf{p}_{\mathbf{i}}^{\mathbf{C_1}}) + 0$$

with

$$\hat{\mathbf{T}} = \begin{bmatrix} 0 & -T_3 & T_2 \\ T_3 & 0 & -T_1 \\ -T_2 & T_1 & 0 \end{bmatrix}$$

Taking the inner product of both sides with $p_i^{C_2}$

$$(\mathbf{p}_{\mathbf{i}}^{\mathbf{C_2}})^T\hat{\mathbf{T}}\mathbf{R}_{C2}^{C1}(\mathbf{p}_{\mathbf{i}}^{\mathbf{C_1}}) = 0 \tag{2.12}$$

This equation is known as the epipolar constraint and the essential matrix $\mathbf{E}$ is given by

$$\mathbf{E} = \hat{\mathbf{T}}\mathbf{R}_{C2}^{C1}$$

$\mathbf{E}$ is a function of $\mathbf{R}_{C2}^{C1}$ and $\mathbf{T}_{C2}^{C1}$ and if $\mathbf{E}$ can be calculated $\mathbf{R}_{C2}^{C1}$ and $\mathbf{T}_{C2}^{C1}$ can be recovered.

If a number of point correspondences between images from camera one and images from camera two can be generated then by exploiting the epipolar constraint and the properties of the matrix $\mathbf{E}$ a precise numerical approximation of $\mathbf{E}$ can be calculated. A common algorithm which does this is the 8-Point algorithm [26]. In brief the algorithm sets up a linear system of equations using the point correspondences and $\mathbf{E}$ that conforms to the epipolar constraint. This system is then solved in a least squares sense to give a best fit $\mathbf{E}$. Using SVD the rank of $\mathbf{E}$ is forced to be two, which is the form required for an essential matrix. The result is an accurate approximation of $\mathbf{E}$. With $\mathbf{E}$ known, $\mathbf{R}_{C2}^{C1}$ and $\mathbf{T}_{C2}^{C1}$ can be recovered using SVD.

It was shown that for any point in image one, the corresponding point in image two will lie along the line defied by $\ell_{\mathbf{i}}^{\mathbf{C_2}} = \mathbf{E}\mathbf{p}_{\mathbf{i}}^{\mathbf{C_1}}$. A method to calculate $\mathbf{E}$ and find the location of camera two in relationship to camera one has also been developed. Using all of these knowns a point in image one can be chosen, then the corresponding point in image two can be found along the line $\ell_{\mathbf{i}}^{\mathbf{C_2}}$, which allows the triangulation of point P using the known correspondences, $\mathbf{R}_{C2}^{C1}$, and $\mathbf{T}_{C2}^{C1}$.

2.7.2 Stereo rectification

Up to now one of the two assumptions which was made earlier has been addressed, which is that $\mathbf{R}_{C2}^{C1}$ and $\mathbf{T}_{C2}^{C1}$ were known. The second assumption was that there was no noise in the image. In reality noise is unavoidable in imaging due to the fact that points in continuous space are projected into pixels which have discrete coordinates. A second level of noise is added due to imperfections and distortions in the lens of the camera. With noise added into the images the two rays projected out from each cameras origin through the corresponding image points will not intersect in space. Thus an approximate intersection must be chosen which minimizes some sort of error metric, such as the re-projection error in both images.

Avoiding the complexities of approximating the intersection of the two lines

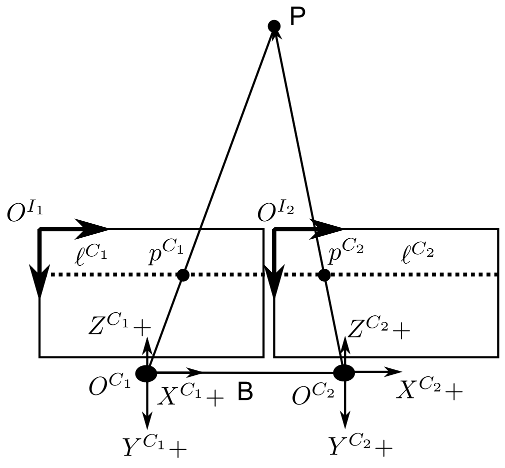


Figure 2.6: Example of two stereo rectified cameras

and continuously calculating search lines $\ell_i^{C_2}$ to find correspondences, the images from each camera can first be rectified. In a rectified stereo pair the cameras have the layout shown in Figure 2.6. In this camera layout the baseline between the cameras does not intersect the image plane because the image planes are parallel. Since the baseline does not intersect the image planes the epipolar points are now at infinity. When this happens the corresponding epipolar lines in each image are the same and are all parallel. This simplifies the search for correspondences because now a pixel at location $p^{C_1} = (x, y)$ will correspond to a pixel in image two at $p^{C_2} = (x - d, y)$. The value d is known as the disparity for the pixel between the two images. Correspondences can be easily generated by comparing the sum of the color values in a window around a point $p_i^{C_1}$ in image one to the sum of the color values in a window around a point $p_i^{C_2}$ in image two where the two points are related by a disparity d. The value of d which minimizes the difference of these two sums is the optimum disparity for the pixel.

Looking at the geometry between the two cameras the depth of a point is directly related to the length of the baseline, the focal length of the camera, and the disparity. This relationship is given by

$$Z_i^{C_1} = f\frac{B}{d}$$

Where B is the length of the baseline, d is the disparity, f is the focal length, and Z is the distance of the world point from the cameras origin, along the Z axis. Figure 2.7 shows this relationship. Ignoring the fact that noise causes the projection rays from each image to not intersect at an exact point, but instead choosing to re project the point along the ray corresponding to image one, then the coordinates of point P_i can be reconstructed.

$$P_i = \left(\frac{Z_i^{C_1}}{f}x, \frac{Z_i^{C_1}}{f}y, Z_i^{C_1}\right)$$

To convert the cameras geometry to the geometry of stereo rectified cameras the image plane of the two cameras can be rotated in space so that they become co-planar. If the two planes are only rotated then the baseline will remain the same and the above calculations will hold. Once the rotation is found which aligns the two image planes a transformation can be calculated which converts the pixel coordinates in the original image plane to the proper coordinates in the new image plane. The result is two stereo rectified images. OpenCV includes a function which can perform this transformation which is based upon the method in [14]. If the object points are reconstructed with respect to this rectified image plane the reconstructed points can be transfered back to the original coordinate system by using the inverse of the rotation used to generate the new image plane. Thus it has been shown how the locations of 3D points of an object can be recovered from two images of the points.

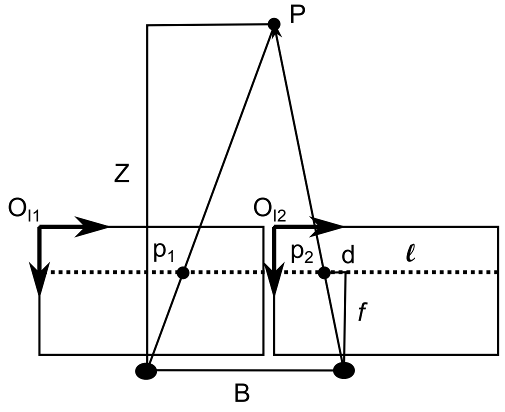


Figure 2.7: Geometry of the disparity to depth relationship

Chapter 3

Methods

This thesis will focus on the implementation and comparison of three pose estimation algorithms. The first of the three algorithms is the SoftPOSIT [9] algorithm. The second algorithm is an extension of the SoftPOSIT algorithm designed to work with line features [8] instead of point features. The last of the three algorithms is one proposed by Ulrich Hillenbrand in a paper called Pose Clustering From Stereo Data [21].

3.1 SoftPOSIT

3.1.1 The SoftPOSIT algorithm

The SoftPOSIT algorithm is an extension of POSIT which is designed to work with unknown correspondences. The algorithm develops correspondences while updating the estimate of the pose. The algorithm takes an initial guess of the pose and then develops possible correspondences based upon the initial pose guessed. With the set of guessed correspondences the pose can be refined and then new correspondences generated. This process is repeated until a final set of correspondences and the pose fitting the correspondences is generated. First the method used to update the pose is changed slightly from the original POSIT algorithm.

3.1.1.1 Updating POSIT

The previous definition of a rotation matrix $\mathbf{R}_C^M$ from equation (2.1) will again be used. The vector $\mathbf{T}_C^M = [T_x, T_y, T_z]^T$ is the translation from the origin of the camera C^O to the origin of the model P_0^C, which need not be a visible point. The rigid body transform relating the model frame to the camera frame is then given by the combination of $\mathbf{R}_C^M$ and $\mathbf{T}_C^M$. The image coordinates of the N model points $P_{i=0\ldots N}^M$ with the model pose given by $\mathbf{R}_C^M$ and $\mathbf{T}_C^M$ are

$$\begin{bmatrix} w_i x_i^P \\ w_i y_i^P \\ w_i \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_C^M & \mathbf{T}_C^M \\ \bar{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} P_i^M \\ 1 \end{bmatrix}$$

Notice that the camera matrix $\mathbf{H}$ here assumes the image coordinates are with respect to the principal point, not the shifted image origin as in equation (2.5). The previous expression can be rewritten to take the form

$$\begin{bmatrix} w_i x_i^P \\ w_i y_i^P \\ w \end{bmatrix} = \begin{bmatrix} f\mathbf{r_1} & fT_x \\ f\mathbf{r_2} & fT_y \\ \mathbf{r_3} & T_z \end{bmatrix} \begin{bmatrix} P_i^M \\ 1 \end{bmatrix}$$

Setting $s = f/T_z$ and remembering that homogeneous coordinates are scale invariant, the previous equation can be re-written

$$\begin{bmatrix} w_i x_i^P \\ w_i y_i^P \end{bmatrix} = \begin{bmatrix} s\mathbf{r_1} & sT_x \\ s\mathbf{r_2} & sT_y \end{bmatrix} \begin{bmatrix} P_i^M \\ 1 \end{bmatrix} \tag{3.1}$$

$$w_i = \mathbf{r_3} \cdot P_i^M / T_z + 1$$

Notice that w is similar to the $(\epsilon + 1)$ term in equations (2.7) and (2.8) from the POSIT algorithm. Similar to the term in POSIT, w is the projection of a model line onto the cameras Z axis plus one. That is, w is the ratio of the distance from the camera origin to a model point over the distance from the camera origin to the SOP plane, or simply the ratio of the Z coordinate of a model point over the distance to the SOP plane .

The equation for the SOP of a model point takes a similar form

$$\begin{bmatrix} x_i \\ yi \end{bmatrix} = \begin{bmatrix} s\mathbf{r_1} & sT_x \\ s\mathbf{r_2} & sT_y \end{bmatrix} \begin{bmatrix} P_i^M \\ 1 \end{bmatrix} \tag{3.2}$$

This is identical to equation (3.1) if and only if $w = 1$. If $w = 1$ then $\mathbf{r_3} \cdot P_i^M = 0$ which means that the model point lies on the SOP projection plane and the SOP is identical to the perspective projection.

Rearranging equation (3.1) gives

$$\underbrace{\begin{bmatrix} P_i^M & 1 \end{bmatrix} \begin{bmatrix} s\mathbf{r_1}^T & s\mathbf{r_2}^T \\ sT_x & sT_y \end{bmatrix}}_{p_i'} = \underbrace{\begin{bmatrix} w_i x_i^P & w_i y_i^P \end{bmatrix}}_{p_i''} \tag{3.3}$$

Assuming there are at least four correspondences between model points P_i^M and image points p_i and that w_i for each correspondence is known, a system of equations can be set up to solve for the unknown parameters in equation (3.3).

The left half of equation (3.3) will be defined as p_i' which is the SOP of model point P_i^M for the given pose, as in Figure 3.1. This definition is straightforward as the left half of equation (3.3) is simply the transpose of equation (3.2) which was the equation to find the image coordinates of the SOP of a model point. The right hand side of equation (3.3) will be defined as p_i'' which is the SOP of model point P_i^M constrained to lie along the true line of sight L of P_i^C, which is the line passing through the camera origin and the actual image point p_i. The point lying along the line of sight will be referred to as P_{Li}^C and will be constrained to have the same Z coordinate as P_i^C. Figure 3.1 illustrates the relative layout of the points. Its been shown that $p_i'' = w_i p_i$ which can be proven by observing the geometry of the points. It was shown before that w_i is the ratio of the Z coordinate of a model point over the distance to the SOP plane T_z. Therefore, $w_i T_z$ is the Z coordinate of a point P_i^C. Using this fact $P_{Li}^C = w_i T_z p_i / f$ which is the re projection of image point p_i to a depth $w_i T_z$. This gives the camera coordinates of point P_{Li}^C.

When the correct pose is found, the points p_i' and p_i'' will be identical because P_i^C will already lie along L, the line of sight of P_i^C. Thus the goal of the algorithm is to find a pose such that the difference between the actual SOP and the SOP constrained to the lines of sight is zero.

An error function which defines the sum of the squared distances between p_i' and p_i'' as the error is given by

$$E = \sum_i^N d_i^2 = \sum_i^N |p_i' - p_i''|^2 = \sum_i^N (\mathbf{Q_1} \cdot \mathbf{P_i^M} - w_i x_i^P)^2 + (\mathbf{Q_2} \cdot \mathbf{P_i^M} - w_i y_i^P)^2 \tag{3.4}$$

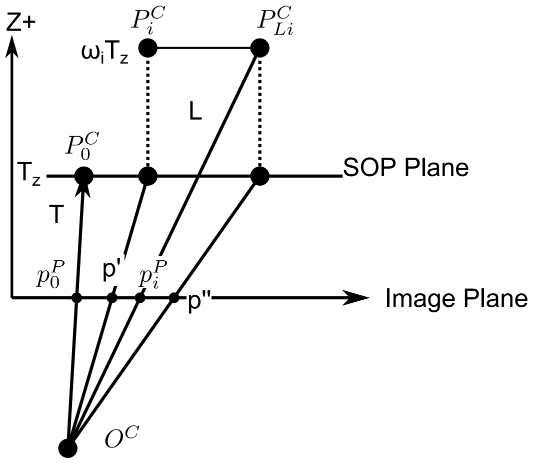


Figure 3.1: Point relationships in SoftPOSIT

with

$$\mathbf{Q_1} = s \begin{bmatrix} \mathbf{r_1} & T_x \end{bmatrix}$$

$$\mathbf{Q_2} = s \begin{bmatrix} \mathbf{r_2} & T_y \end{bmatrix}$$

Iteratively minimizing this error will eventually lead to the right pose.

To minimize the error the derivative of equation (3.4) is taken, which can be expressed as a system of equations

$$\mathbf{Q_1} = \left(\sum_i^N \mathbf{P_i^M} \mathbf{P_i^{M^T}} \right)^{-1} \left(\sum_i^N w_i x_i^P \mathbf{P_i^M} \right) \tag{3.5}$$

$$\mathbf{Q_2} = \left(\sum_i^N \mathbf{P_i^M} \mathbf{P_i^{M^T}} \right)^{-1} \left(\sum_i^N w_i y_i^P \mathbf{P_i^M} \right) \tag{3.6}$$

Like POSIT, at the start of the loop it can be assumed that $w_{i=0...N} = 1$ calculate new values for $\mathbf{Q_1}$ and $\mathbf{Q_2}$, then update $w_{i=0...N}$ using the new estimated pose. What we have developed up to here is simply a variation on the original POSIT algorithm, now it can be extend to work with unknown correspondences.

3.1.1.2 POSIT with unknown correspondences

For the case of unknown correspondences there are N model points and M image points. Model points will be indexed with the subscript i and image points with the subscript j, thus there are $P^M_{i=0\ldots N}$ model points and $p_{j=0\ldots M}$ image points. If correspondences are unknown then any image point can correspond to any model point and there are a total of MN possible correspondences.

With w_i defined as before in (3.1). The new SOP image points are

$$p''_{ji} = w_i p_j \tag{3.7}$$

and

$$p'_i = \begin{bmatrix} \mathbf{Q_1} \cdot \mathbf{P^M_i} \\ \mathbf{Q_2} \cdot \mathbf{P^M_i} \end{bmatrix} \tag{3.8}$$

Where equation (3.7) is the SOP of model point P^C_i constrained to the line of sight L of image point p_j and equation (3.8) is identical to the original p'_i in (3.3) but adding the new Q notation.

The distance between points p''_{ji} and p'_i is given by

$$d^2_{ji} = \left|p'_i - p'_{ji}\right|^2 = (\mathbf{Q_1} \cdot \mathbf{P^M_i} - w_i x^P_j)^2 + (\mathbf{Q_2} \cdot \mathbf{P^M_i} - w_i y^P_j)^2 \tag{3.9}$$

which can be used to update the previous error equation (3.4) giving a new error equation

$$E = \sum_i^N \sum_j^M m_{ij} \left(d^2_{ji} - \alpha\right) = \sum_i^N \sum_j^M m_{ij} \left((\mathbf{Q_1} \cdot \mathbf{P^M_i} - w_i x^P_j)^2 + (\mathbf{Q_2} \cdot \mathbf{P^M_i} - w_i y^P_j)^2 - \alpha\right) \tag{3.10}$$

where m_{ji} is a weight in the range $0 \leq m_{ji} \leq 1$ expressing the likelihood that model point P_i^M corresponds to image point p_j. The term α is here to bump the error away from setting all the weights to zero and to account for noise is the locations of feature points in the images so that slightly mis-aligned model and image points can still be matched. In the case that all correspondences are completely correct $m_{ij} = 1\,\text{or}\,0$ and $\alpha = 0$ this equation is identical to the previous error equation (3.4).

The matrix $\mathbf{m}$ is a $(M + 1) \times (N + 1)$ matrix where each entry expresses the probability of correspondence between image points and model points. The individual entries are populated based upon the distance between SOP points p''_{ji} and p'_i given by d_{ji}. As d_{ji} increases the corresponding entry in m_{ji} will decrease towards zero and as d_{ji} decreases m_{ji} increases indicating that points likely match. At the end of the SoftPOSIT algorithm the entries of $\mathbf{m}$ should all be nearly zero or one, indicating that points either correspond or don't. The matrix $\mathbf{m}$ is also repeatedly normalized across its rows and columns to ensure that the cumulative probability of any image point matching any model point is one and the total probability of any model point matching any image point is one. This matrix form is referred to as doubly stochastic and an algorithm from Sinkhorn [37] is used to achieve the form. In the case that a given model point is not present in the image or a point in the image does not have a matching model point the weight in the last row or column of $\mathbf{m}$ will be set to one. The last row and column of $\mathbf{m}$ are the slack row and slack column, respectively, and are the reason why $\mathbf{m}$ has plus one rows and columns. Entries in these locations indicate no correspondence could be determined.

With the error function defined the values of $\mathbf{Q_1}$ and $\mathbf{Q_2}$ which will minimize the error are found in the same fashion previously and are given by

$$\mathbf{Q_1} = \left(\sum_i^N \left(\sum_j^M m_{ji} \right) \mathbf{P_i^M} \mathbf{P_i^{M^T}} \right)^{-1} \left(\sum_i^N \sum_j^M m_{ji} w_i x_j^P \mathbf{P_i^M} \right) \tag{3.11}$$

$$\mathbf{Q_2} = \left(\sum_{i}^{N}\left(\sum_{j}^{M} m_{ji}\right)\mathbf{P_i^M}\mathbf{P_i^{M^T}}\right)^{-1}\left(\sum_{i}^{N}\sum_{j}^{M} m_{ji}w_i y_j^P \mathbf{P_i^M}\right) \tag{3.12}$$

As before the algorithm is started with a $w_{i=0\ldots N} = 1$ and then $\mathbf{m}$ is populated by calculating all of the values of d_{ji}. Since $\mathbf{m}$ must be populated before updating $\mathbf{Q_{1,2}}$, an initial pose must be given to the algorithm which is the pose used to generate $\mathbf{m}$. With $\mathbf{m}$ populated $\mathbf{Q_{1,2}}$ can be updated, which is then used to generate a better guess for $w_{i=0\ldots N}$. This process is repeated until convergence.

At the conclusion of the algorithm the pose parameters can be retrieved from $\mathbf{Q_{1,2}}$.

$$s = (\left\| \left[Q_1^1, Q_1^2, Q_1^3\right] \right\| \left\| \left[Q_2^1, Q_2^2, Q_2^3\right] \right\|)^{1/2}$$

$$\mathbf{R_1} = \left[Q_1^1, Q_1^2, Q_1^3\right]^T / s \quad \mathbf{R_2} = \left[Q_2^1, Q_2^2, Q_2^3\right]^T / s$$

$$\mathbf{R_3} = \mathbf{R_1} \times \mathbf{R_2}$$

$$\mathbf{T} = \left[\frac{Q_1^4}{s}, \frac{Q_2^4}{s}, \frac{f}{s}\right]$$

3.1.2 Limitations and Issues with SoftPOSIT

The need for an initial guessed pose is the major limitation of the SoftPOSIT algorithm. Since the pose update equation is dependent upon the initial pose with which the algorithm is started, the algorithm will only converge to the local minimum which satisfies the error function (3.10). To converge to the true pose the algorithm may need to be started at a variety of different poses around the actual pose. Additionally if the algorithm is started at a pose which is too far away from the correct pose the algorithm will not converge and will terminate early with no solution.

Since the algorithm relies on matching feature points and pays no attention to the visibility of feature points based upon pose, the algorithm will often match points in the model which actually are occluded by the model its self to points in the image. For example, when trying to match our cube model to the images most of the corner points on the back of the cube are not visible because the front of the cube occludes the back; however, the algorithm will often match points which correspond to the back of the model to imaged corners belonging to the front of the cube. These types of matches should not be allowed due to the geometry of the model occluding its self, but there is no mechanism in the algorithm to account for this.

The other major limitation of this algorithm is accurate feature extraction. When trying to detect corners for example a rounded corner may not be detected or the overlap of two objects may lead to spurious corner detections which the algorithm may converge to.

When the algorithm does finally converge to a pose there is no way of knowing if the algorithm generated the correct correspondences or even matched to true features of the object instead of some of the spuriously detected ones. Thus any pose detected by the algorithm must be evaluated to check its 'fitness' before being accepted as the final answer.

3.2 SoftPOSIT With Line Features

3.2.1 SoftPOSIT With Line Features Algorithm

After the initial development of SoftPOSIT an extension of the algorithm was created to allow the algorithm to be run on line features [8]. The underlying SoftPOSIT algorithm is identical to the one previously described. Since the SoftPOSIT algorithm relies on point features to actually preform the pose estimation and correspondence determination the line features and correspondences are converted to point

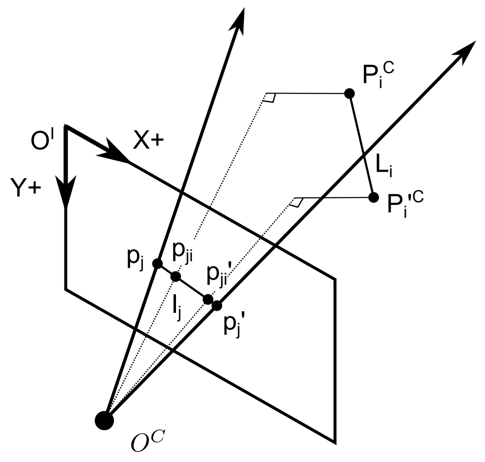


Figure 3.2: Generation of projected lines in SoftPOSITLines

features and point correspondences.

3.2.1.1 Converting line features to point features

For the current image all of the lines which are candidates for matching to the model lines are detected. Using the previous notation the two end points of a model line are given by $L_i = (P_i^M, P_i'^M)$ and the two end points of a detected image line are $l_j = (p_j, p_j')$. N will now represent the number of model lines, meaning there will be $2N$ model points which correspond to the lines and M image lines which will have a total of $2M$ points. The plane in space which contains the actual model line used to generate image line l_j can be defined using the points (C^O, p_j, p_j') as in Figure 3.2. The normal to this plane $\mathbf{n_j}$ is given by

$$\mathbf{n_j} = [p_j, 1] \times [p_j', 1]$$

If the current model pose is correct and model line L_i corresponds to image line l_j then the points

$$\mathbf{S_i^C} = \mathbf{R}_C^M \mathbf{P_i^M} + \mathbf{T}_C^M \quad \mathbf{S_i'^C} = \mathbf{R}_C^M \mathbf{P_i'^M} + \mathbf{T}_C^M$$

will lie on the plane defined by (C^O, p_j, p'_j) and will also satisfy the constraint that $\mathbf{n_j^T S_i^C} = \mathbf{n_j^T S_i'^C} = 0$. Using the SoftPOSIT algorithm its assumed that at first $\mathbf{R}_C^M$ and $\mathbf{T}_C^M$ will not be correct and therefore L_i will not lie in the plane.

Recalling the SoftPOSIT algorithm the model points given the current pose were constrained to lie on the lines of sight of image points. In this instance it will be required that the model lines lie on the planes of sight of the image lines. If model line endpoints given by S_i^C and $S_i'^C$ are the model line endpoints in the cameras frame for the current pose, then the nearest points to these line endpoints which fulfill the planar constraint are the orthogonal projections of points S_i^C and $S_i'^C$ onto the plane of sight. The coordinates of these projected points are given by

$$\mathbf{S_{ji}^C} = \mathbf{RP_i^M} + \mathbf{T}\left[(\mathbf{RP_i^M} + \mathbf{T}) \cdot \mathbf{n_j}\right] * \mathbf{n_j} \tag{3.13}$$

$$\mathbf{S_{ji}'^C} = \mathbf{RP_i'^M} + \mathbf{T}\left[(\mathbf{RP_i'^M} + \mathbf{T}) \cdot \mathbf{n_j}\right] * \mathbf{n_j} \tag{3.14}$$

Notice these points are still in the 3D camera frame, however the image of these points can be generated as

$$p_{ji} = \frac{(S_{ij_x}, S_{ij_y})}{S_{ij_z}} \quad p'_{ji} = \frac{(S'_{ij_x}, S'_{ij_y})}{S'_{ij_z}} \tag{3.15}$$

The collection of point pairs given by 3.15 will be analogous to the constrained SOP points p''_{ji} see equation (3.7). The collection of these points for the current guess of $\mathbf{R}_C^M$ and $\mathbf{T}_C^M$ will be referred to as

$$P_{img}(\mathbf{R}_C^M, \mathbf{T}_C^M) = \left\{p_{ji}, p'_{ji}, 1 \le i \le N, 1 \le j \le M\right\} \tag{3.16}$$

The collection of model points analogous to p'_i see equation (3.8) will be referred to as

$$P_{model} = \left\{ P_i^M, P_i'^M, 1 \leq i \leq N \right\} \tag{3.17}$$

A new $\mathbf{m}$ matrix for expressing the probability that point p_{ji} corresponds to P_i^M and p'_{ji} corresponds to $P_i'^M$ must now be developed. The total dimensionality of $\mathbf{m}$ will be $2MN \times 2N$ but the matrix will only be sparsely populated. First, half of the possible entries are 0 because p_{ji} corresponds to P_i^M and p'_{ji} corresponds to $P_i'^M$ but the opposite is not true i.e. p_{ji} does not correspond to to $P_i'^M$ and p'_{ji} does not corresponds to P_i^M. Since the image points are generated by projecting model lines onto planes formed by image lines, image points should only be matched back to the model lines which generated them. If for example a set of image points p_{j1} and p'_{j1} correspond to L_1 projected onto all of the image line planes, p_{j1} should only be matched to P_1^M and p'_{j1} to $P_1'^M$. Attempting other correspondences would be senseless as the points p_{j1} and p'_{j1} are derived from L_1. Thus $\mathbf{m}$ will take a block diagonal form as in the example Figure 3.2.1.1. In Figure 3.2.1.1 l_1 corresponds to L_3 and l_2 corresponds to L_1. As before the matrix is required to be doubly stochastic which can still be achieved via Sinkhorn's [37] method. When the pose is correct every entry in the matrix will be close to one or zero indicating that the lines/points either correspond or don't.

Again recalling the previous algorithm the values of $\mathbf{m}$ prior to normalization are related to the distance between model points SOP's and their line of sight corrected SOP's. Since this algorithm is matching line features distances will be defined in terms of line differences rather than point distances. Using these distances, any points generated from model line L_i and image line l_j i.e. points p_{ji}, p'_{ji} have distance

	P_1	P_1'	P_2	P_2'	P_3	P_3'
p_{11}	.3	0	0	0	0	0
p'_{11}	0	.3	0	0	0	0
p_{12}	0	0	.1	0	0	0
p'_{12}	0	0	0	.1	0	0
p_{13}	0	0	0	0	.8	0
p'_{13}	0	0	0	0	0	.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
p_{21}	.7	0	0	0	0	0
p'_{21}	0	.7	0	0	0	0
p_{22}	0	0	.2	0	0	0
p'_{22}	0	0	0	.2	0	0
p_{23}	0	0	0	0	.2	0
p'_{23}	0	0	0	0	0	.2

Figure 3.3: Example form of the matrix **m** for SoftPOSIT with line features

measures

$$d_{ji} = \Delta\theta(l_j, l_i) + \rho d(l_j, l_i) \tag{3.18}$$

where

$$\Delta\theta(l_j, l_i) = 1 - \cos \angle l_j l_i$$

l_i is the line obtained by taking the perspective projection of L_i and $d(l_j, l_i)$ is the sum of the distances from the endpoints of l_j to the closest point on l_i. Thus this distance metric takes into account the mis orientation of two matched lines and the distance between the two lines. The reason $d(l_j, l_i)$ is chosen as the sum of the endpoints of the image line to the closest point on the imaged model line is that a partially occluded line will still have a distance of zero indicating that a match is found. This behavior is desirable because the algorithm should be able to match partially occluded image lines to whole model lines. These distance measures are used to populate **m** prior to the normalization by the Sinkhorn algorithm.

Using the new weighting matrix, and modified p''_{ji} and p'_i given by equations

(3.16) and (3.17) respectively the originally described SoftPOSIT algorithm can be applied to the line generated points.

The algorithm is started and terminated in the same fashion as before. The algorithm is started with an initial pose guess and assumes $w_i = 1$. The algorithm then generates the points P_{img} and the corresponding weights for the probability of points matching, and updates $\mathbf{Q_1}$ and $\mathbf{Q_2}$ using the weights and current w's. Next, the algorithm updates the values for w's using the current pose guess and repeats the process until convergence.

3.2.2 Limitations and Issues with SoftPOSIT Using Line Features

The major advantage of using line features over point features is that line features are generally more stable and easier to detect. For example a rounded corner probably won't be detected by a corner detector; however, the two lines leading into the rounded corner will still appear. The problem of occlusions generating fake features is still present because two overlapping objects will generally form a line when a line detector is used.

The problem of self-occlusion is also still not addressed in this algorithm, so lines which are not visible in the current object pose can still be matched to image lines. This is especially a problem when the object is symmetric and thus has many lines which are parallel and can align when in certain poses.

This algorithm also returns the local pose which minimizes the error function so again the algorithm must be started using different initial poses to find the global minimum. The poses must also be evaluated for correctness as with regular SoftPOSIT.

Typically, when compared to SoftPOSIT using point features the final poses returned by the algorithm are more accurate and the probability of converging to the correct pose is generally higher.

3.3 Pose Clustering From Stereo Data

3.3.1 Pose Clustering From Stereo Data Algorithm

In Section 2.7 it was shown how it is possible to generate a 3D point cloud reconstruction of a scene given two views of the scene. It will now be assumed that a model point cloud M has been generated where the origin and orientation of the model frame is known and the points coordinates are expressed in reference to this frame. The origin of the model is located at the center of the model point cloud. For every point in the model cloud the line of sight from the camera to to the original point must also be stored, the need for this will be shown later.

If another image is captured of the same object and the coordinates of the 3D point cloud reconstruction are generated with respect to the camera coordinate system, then this point could will be referred to as S the scene point cloud. The goal then is to find some rigid body transform which will relate the points in M to the points in S. This transform will be the pose of the object w.r.t the cameras coordinate system.

It should be noted that in the full implementation of this algorithm the model is generated by taking multiples views of an object from different angles and reconstructing the complete 3D geometry of the object. This task is simple enough to do if the object is placed at a location in a model based coordinate system and the camera is moved to specific known locations in the model frame so that all of the

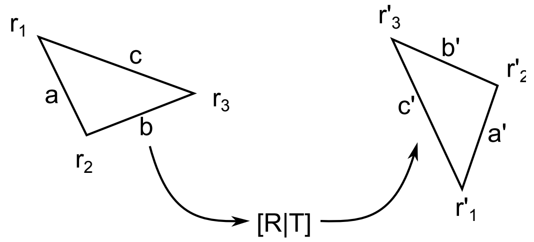


Figure 3.4: Example of two matched triplets

reconstructions from each view can be transformed into the frame of reference of the object. In this implementation we will be using a model constructed from only a single viewpoint; however, this has no bearing on the implementation so the extension to multiple views is as simple as stitching together different viewpoints to make a more complete model.

Assuming that both the model and scene point clouds are generated. If three points in M which correspond to three points in S can be identified, then the transform that moves the coordinates of the the points in M to the corresponding points in S gives the pose of the object. However, full point correspondences are impossible to generate because the only data being used in this algorithm is 3-D point data. Since point correspondences can not be directly generated by matching features, triplet correspondences are generated instead. Where triplet correspondences refers to matching the lengths between three points in the scene to the lengths between 3 points in the model. Figure 3.4 shows two matched triplets. If 3 triplet lengths can be matched then three point correspondences can be generated and the pose can be reconstructed. Since the 3D point cloud reconstruction is not exactly accurate due to noise in the images and re projection errors it is not possible to match triplet lengths exactly. Instead a matching threshold is used such that if two lengths are within some tolerance they are considered to be matched.

Due to the matching tolerance and the geometry of the objects there will be many triplets in the model which can be matched to a single triplet in the scene. If all of the the rotations and translations were computed which move all of the matched model triplets into the scene, then only one of the transforms would be the correct transform and all of the others would be incorrect. If the process of picking a triplet from the scene and matching it to possible matches in the model is repeated then eventually a number of correct guesses would be generated along with many many more incorrect guesses. However, if the poses are stored in a 6D parameter space a cluster of poses corresponding to the actual object pose will develop along with other randomly distributed poses throughout the rest of the space. Dividing the 6D parameter space into a set of hypercubes allows easy detection of when a cluster has been generated in space. Once a cluster of points in the parameter space is detected, the pose which best describes the cluster can be generated. This pose will then correspond to the pose of the object in space.

Using this approach, Hillenbrand developed an algorithm [21] that is summarized as follows:

1. Draw a random point triple from S the sample point cloud
2. Among all matching point triples in M pick one at random
3. Compute the rigid body transform which moves the triple from M to the triple in S.
4. Generate the six parameters which describe the transform and place the pose estimate into the 6D pose space.
5. If the hypercube containing this 6D point has less than $N_{samples}$ members return to 1 otherwise continue.
6. Estimate the best pose using the 6D point cluster generated in space.

Now that the algorithm as a whole has been presented the details of the steps will be presented.

To find pairs of matching triplets an efficient method for matching triplet lengths needs to be developed. To do this a hash table containing triplets from the model which is indexed by the lengths between the points is generated. To ensure that points are always matched in the proper order the line lengths are always generated by going clockwise around the points according to the the point of view of the camera. Failure to do this would result in incorrect point correspondences even though the lines were correctly matched. The three values used to hash three model points r_1, r_2, r_3 with lines of sight l_1, l_2, l_3 are given by the equation

$$\begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} = \left\{ \begin{matrix} \begin{pmatrix} \|r_2 - r_3\| \\ \|r_3 - r_1\| \\ \|r_1 - r_2\| \end{pmatrix} & \text{if } \left[(r_2 - r_1) \times (r_3 - r_1)\right]^T (l_1 + l_2 + l_3) > 0 \\ \begin{pmatrix} \|r_3 - r_2\| \\ \|r_2 - r_1\| \\ \|r_1 - r_3\| \end{pmatrix} & \text{else} \end{matrix} \right\} \tag{3.19}$$

where k_1, k_2, k_3 are the lengths between the points. This hashing method guarantees that the points are hashed in a clockwise order according to the point of view of the camera. In addition to hashing the three points with the order k_1, k_2, k_3 the points are also hashed with the keys k_3, k_1, k_2 and k_2, k_3, k_1. The three points are hashed with all three of these entries because when picking three points from the current scene there is no way of knowing which order they will appear in, only that the lengths between them are generated in a clockwise manner. Using the method presented to generate clockwise lengths a point triple can be selected in S and the appropriate lengths generated. Using those three lengths and the hash table, all of the model triples which could possibly match the scene triple can be quickly found.

The method for finding the rigid body transform which relates the three points is based on quaternions and is explained in [24]. This method is used because it finds the best fit $\mathbf{R}$ and $\mathbf{T}$ in a least squares sense that relates points r_1, r_2, r_3 in the model to points r'_1, r'_2, r'_3 in the scene. This method is also specifically designed to work with three pairs of corresponding points which is the number of correspondences which this algorithm generates.

A method of converting pose parameters to 6D points is now presented. A rotation matrix $\mathbf{R}$ can be expressed as an axis of rotation and an angle of rotation.

$$R = \exp(\hat{\mathbf{w}}\theta)$$

where $\hat{\mathbf{w}}$ is the unit vector about which the rotation takes place and θ is the amount in radians by which points are rotated. The vector $\hat{\mathbf{w}}\theta$ is called the canonical form of a rotation. $\mathbf{R}$ will now be considered the canonical form of a rotation matrix $\mathbf{R}$. Combining the vectors $[\mathbf{R}, \mathbf{T}]$ into one large vector gives a 6D vector which completely describes the rigid body transform/pose. This 6D vector could be chosen to preform pose clustering, but there is one major problem. If parameter clustering is to be preformed, a consistent parameter space must be used so that clusters are not formed due to the topology of the parameter space alone. Hillenbrand shows in his earlier work [39] that the canonical parameter space is not consistent and therefore is not suitable for parameter clustering. He proposes a transform

$$\rho = \left(\frac{\|\mathbf{R}\| - \sin\|\mathbf{R}\|}{\pi}\right)^{1/3} \frac{\mathbf{R}}{\|\mathbf{R}\|} \tag{3.20}$$

which is a consistent space parameterized by a vector $\rho \in \mathbb{R}^3$, where each element ρ_1, ρ_2, ρ_3 all satisfy $-1 \le \rho_i \le 1$. Thankfully the Euclidean translation space is already consistent so all of the pose estimates can be stored in a consistent 6D parameter space using vector $p = [\rho, \mathbf{T}]$.

Now that poses can be parameterized in a 6D space, the final part of the algorithm is examined. This part of the algorithm determines the best pose which represents all of the pose points in the cluster. The best pose is found by using a mean shift procedure described in [7]. The procedure is started with p^1 equal to the mean of all the poses $p_{i=1 \dots N_{samples}}$ in the bin which was filled, and is repeated until $\left\| p^k - p^{k-1} \right\| < \delta$ indicating the procedure has converged.

$$p^k = \frac{\sum_{i=1}^{N_{samples}} w_i^k p_i}{\sum_{i=1}^{N_{samples}} w_i^k} \tag{3.21}$$

$$w_i^k = u\left(\left\| \rho^{k-1} - \rho_i \right\| / r_{rot} \right) u\left(\left\| \mathbf{T}^{k-1} - \mathbf{T}_i \right\| / r_{trans} \right)$$

where

$$u(x) = \begin{Bmatrix} 1 \text{ if } x < 1 \\ 0 \text{ else} \end{Bmatrix}$$

The radii r_{rot} and r_{trans} define a maximum radius around the current mean which points must lie in in order to contribute to the new mean. These values are dependent upon the bin size used to generate the point cluster and can be varied accordingly. The final result of the clustering algorithm is the pose given by p^k which represents the mean of the major cluster within the bin. This is the final pose output by the algorithm.

3.3.2 Limitations and Issues with Pose Clustering

Out of the three algorithms discussed in this thesis this is the only one which is feature independent. This is one of the most appealing parts of this algorithm because it can be used on any type of object with any texture as long a some sort of stereo depth information can be recovered. The drawback to this feature is that it makes it relatively simple to confuse similar objects. For example in the experiments section we will attempt to find cubes and cuboids where the dimensions are the same except that the cuboid is wider. In this sort of case it is easy to place the cube inside of the cuboid because the geometry of the shapes are relatively similar.

Chapter 4

Experiments and Results

4.1 Experiments

4.1.1 The sample set

For the evaluation of the algorithms a total of 95 different images were captured, and their 3D reconstructions were generated. The images include single cubes, cuboids, and assemblies of cubes and cuboids both with and without other objects in the frame and with and without occlusions. The sample set was captured using a pair of Playstation Eye cameras controlled with OpenCV.

Results will be presented that show the effectiveness of each algorithm in detecting the objects of interest see Figure 4.1, and comparing the effectiveness of detecting assemblies using only a single component as the model as compared to the entire assembly as the model, i.e., finding the assembly using only the cube as the model compared to detecting the pose of the assembly using the entire assembly as the model.

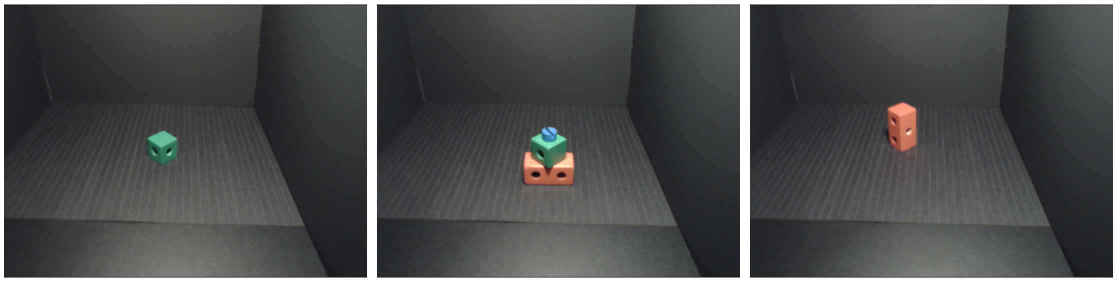

(a) Cube (3cm x 3cm x 3cm) (b) Assembly (c) Cuboid (3cm x 3cm x6 cm)

Figure 4.1: The Objects of Interest

4.1.1.1 Sample set pre-processing

For all of the images background subtraction was used to isolate the actual objects in the scene from the backdrop. The 3D reconstruction and line/corner detection was then performed on the segmented objects only to remove noise sources unassociated with the objects in the scene. No color information was used to distinguish objects from one another, determine object boundaries, or to verify poses.

4.1.1.2 Sample set divisions

The image set was divided into three parts and then all of the algorithms were run against the sets. The first set is the collection of all images where a cube as in Figure 4.1(a) is the object of interest. This set includes pictures of individual cubes, cubes as a part of an assembly, and cubes with other objects and cuboids present. The second set consists of all images where an assembly as in Figure 4.1(b) is the object of interest. The assembly is a cube directly attached to a rectangular cuboid. The assembly set consists of images of a single assembly and images of a single assembly with cubes, cuboids, and other objects present. The final set consists of all images where the rectangular cuboid shown in Figure 4.1(c) is the object of interest. This set contains images of the cuboid by itself and images of the cuboid with cubes and other objects present.

4.1.1.3 Generation of baseline true poses

All of the images in the sample set show the object of interest in a 45x30 cm space which is roughly 30 cm from the camera. Baseline truths for the positions of the objects were generated by placing the objects at a known location in the world coordinate system and converting the coordinates to a location in the cameras coordinate frame of reference. The orientation of the objects was found by fitting the model of the object to the image of the object by manually generating correspondences and applying a PNP (Perspective N Point) algorithm. The baseline truths were generated in this split fashion because it is difficult to accurately determine the 3-D location of an object when trying to generate correspondences by hand because small changes in location of the object can lead to nearly unnoticeable changes in the image data, and manually localizing the point where a feature is located in an image is not necessarily an easy task even for a human. Locating the object in a world coordinate system can easily be done by placing the object on a sheet of grid paper, as was done in this implementation, and then converting the coordinates to camera coordinates with a coordinate system transform. The orientation of the object on the other hand is difficult to measure by hand and can be relatively accurately determined by generating correspondences by hand, because PNP algorithms are less sensitive to noise when determining orientation as apposed to position.

4.1.2 SoftPOSIT algorithm implementations

The SoftPOSITPoints algorithm was obtained directly from Daniel DeMenthnon and was run in Matlab 2011a.

For the points algorithm the chosen features were the block corners. To find the corners a corner detector was used [5]. This corner detector detects corners by looking for line intersections and is designed to robustly detect rounded corners where lines do not meet at a sharp point, which is often the case with the objects of interest.

The SoftPOSITLines algorithm was created by modifying DeMenthnon's original code and was also implemented in Matlab 2011a.

The SoftPOSITLines algorithm required line extraction and this was achieved via the Canny edge detector [4] in combination with the Douglas-Peucker line simplification algorithm [11].

4.1.2.1 SoftPOSIT pose verification

For both of the variants of the SoftPOSIT algorithm the final step before accepting the estimated pose is an evaluation of the 'fitness' of the pose. This was done by calculating the mean distance of the model lines, projected to the image plane given the current pose, to the nearest image lines. Once all of the distances were calculated the pose was accepted if 5 model lines were within 4 pixels average distance to any image lines. In addition the pose was required to project into the foreground segmented region of the image. This check was preformed first to quickly eliminate poses which place the object in an area occupied by the background.

4.1.3 Triplet matching algorithm implementation

The point triplet matching algorithm was coded in Visual Studio 2008 using C++ and the OpenCV library.

In this implementation of the triplet matching algorithm two levels of pose binning were performed. In the first level of binning the translation space is divided into 4cm cubes which corresponds to dividing each translation parameter into 25 bins and the consistent rotation sphere is divided into 64 bins with 4 bins of division for each parameter. In the second level the 4cm cube is divided into 7mm cubes by re binning with 6 bins per parameter. The rotation space is re binned using six bins per parameter to give a resolution .04 units for each parameter. This binning method was chosen because it allows a large initial search space to be somewhat refined then re binned to give an accurate final pose. This saves time in the binning process and memory in the implementation because the more bins which a space is divided into the longer it will take for a bin to fill which means that more pose estimates must be stored before a single bin fills. With the hierarchical binning method a space can be coarsely binned until a bin fills which gives a smaller search space that can then be binned for a more accurate pose estimate. With the hierarchical method all poses from the higher levels of binning can be removed from memory once the refined search space is determined. This helps with avoiding memory limitations when searching for a pose over a large space. For the first level of binning a bin was considered full when it had 5000 poses contained in a bin and for the second level a full bin was achieved at 300 poses. The bigger bins require a bigger count to be considered full because incorrectly matched triplets can accumulate in space easily when the bins are larger and this can lead to bins which fill that do not contain the actual pose. Likewise smaller bins can have a smaller fill count because randomly matched triplets will be more dispersed throughout the bins.

4.1.4 The search space

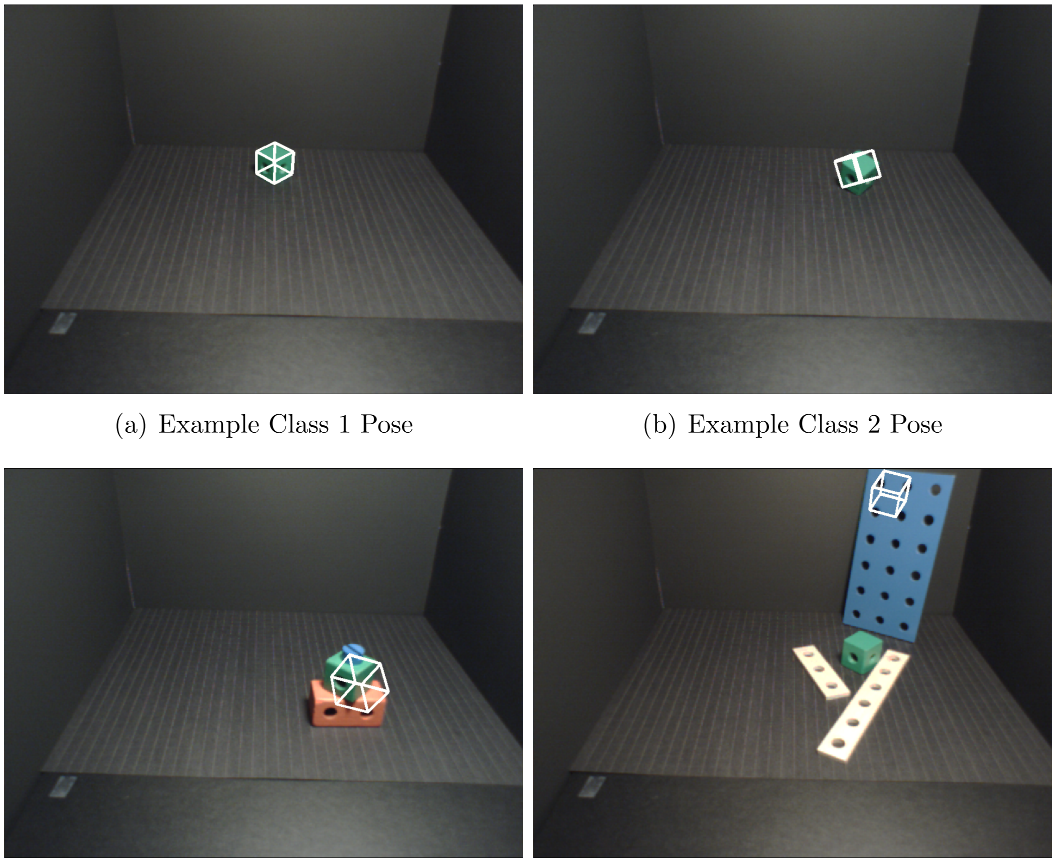
The search space for all of the algorithms was identical. The search space covered -.5m to .5m in the X and Y directions and 0m to 1m in the Z direction, where the X,Y, and Z directions correspond to the camera coordinate axes in Figure 2.2(b) and the full range of rotations in the unit sphere. For the SoftPOSIT algorithms the initial poses were selected at random with uniform probability from within the total range.

4.2 Results

The results of the three algorithms will be discussed in three sections. First the results of the two monocular methods of pose estimation SoftPOSITPoints and SoftPOSITLines will be discussed and analyzed. Then the results of the Binocular triplet matching algorithm will be examined. Finally a global comparison of the three algorithms and their overall performance will be given.

Results from each algorithm will be presented in a fashion that allows direct comparisons of the performance of each algorithm on specific images and in a classed form that enables a way to quickly evaluate the performance of each algorithm in a global sense.

Table 4.1 shows the breakdowns and significance of each class and Figure 4.2 shows example poses from each class. Class 1 corresponds to poses which are correct within a small tolerance, and that can be used in applications where relatively high accuracy is a concern such as accurately gripping an object for assembly. Class 2 provides poses which yield good localization but no useful sense of orientation. This type of accuracy could be useful for simple obstacle avoidance. Class 3 provides poses which accurately express an objects orientation but only approximate its location. These pose results could be used for visual rotational servoing or determining which way a rotating object is pointing. Finally the poses in Class 4 are essentially incorrect and have no real practical use. In all cases the translational error is given in units


(a) Example Class 1 Pose

(b) Example Class 2 Pose

(c) Example Class 3 Pose

(d) Example Class 4 Pose

Figure 4.2: Example poses from each class

Class	Translation Error (cm)		Rotation Error		Significance
	Min	Max	Min	Max	
1	0	1.5	0	2	Essentially correct pose in terms of both position and orientation.
2	0	1.5	2	∞	Pose provides correct localization but no sense of orientation.
3	1.5	3.5	0	2	Pose provides approximate localization and correct orientation.
4	All other cases				Pose provides no useful localization or orientation.

Table 4.1: Summary of the properties and significance of performance classings

of centimeters. The rotation error is given by a uniformly scaled distance in the consistent rotation space (4.1) between the true pose and the algorithms guessed pose.

$$e_r = 10 \left\| \rho_{actual} - \rho_{guess} \right\| \tag{4.1}$$

where

$$\rho = \left(\frac{\|\mathbf{R}\| - \sin \|\mathbf{R}\|}{\pi} \right)^{1/3} \frac{\mathbf{R}}{\|\mathbf{R}\|}$$

For reference, 1 unit of error in the scaled consistent space corresponds to roughly 15° of rotation error along the correct axis of rotation or rotating the correct amount around an axis which is 6° off from the correct axis of rotation. Plots are also provided which show the breakdown of the translation error for each image in terms of error along each unit axis in cm.

The plots displaying the performance of the algorithm across the different images sets are given in Figures 4.3-4.12. Figures 4.3, 4.6, and 4.9 all show the total pose estimation error of each algorithm on each image of the dataset. These figures allow direct comparisons of the performance of each algorithm on individual images.

Figures 4.5, 4.8, and 4.11 all show the breakdown of the translation estimation error of each algorithm on each image of the dataset. These figures allow direct comparisons of the translational accuracy of each algorithm on individual images. Figures 4.4, 4.7, and 4.10 all show total pose estimation error for each algorithm independently sorted by class. These figures allow global comparisons of the performance of each algorithm on the different image sets.

4.2.1 Monocular Methods

Out of the two monocular methods presented, SoftPOSITPoints and SoftPOSITLines, the lines method is definitely the more stable, in terms of converging to a usable pose, and the more accurate of the two. For the case of detecting either the cube or cuboid the lines method consistently had higher detection rates for classes 1,2, and 3 see Figures 4.4,4.10. The only image set where the points algorithm outperformed the lines algorithm was in detecting the assembly, see Figure 4.7. However, in this instance, neither algorithm had any Class 1 detections and the points algorithm only outperformed the lines algorithm by achieving two Class 3 detections.

A major limitation with the monocular pose estimation methods, is correctly determining the Z coordinate of the object, or in other words the object's distance from the camera. Figures 4.5,4.8, and 4.11 all show the breakdown of the translation error into its individual X,Y, and Z components. For both the points and lines algorithms a disproportionately large part of the error is in the Z component. In fact many instances can be found where the X and Y error components are within a few millimeters while the Z component is only within a few centimeters. This error disparity is due to poor localization of image features and pixel discretization. Due to the properties of the perspective transform mis locating a feature in the image by

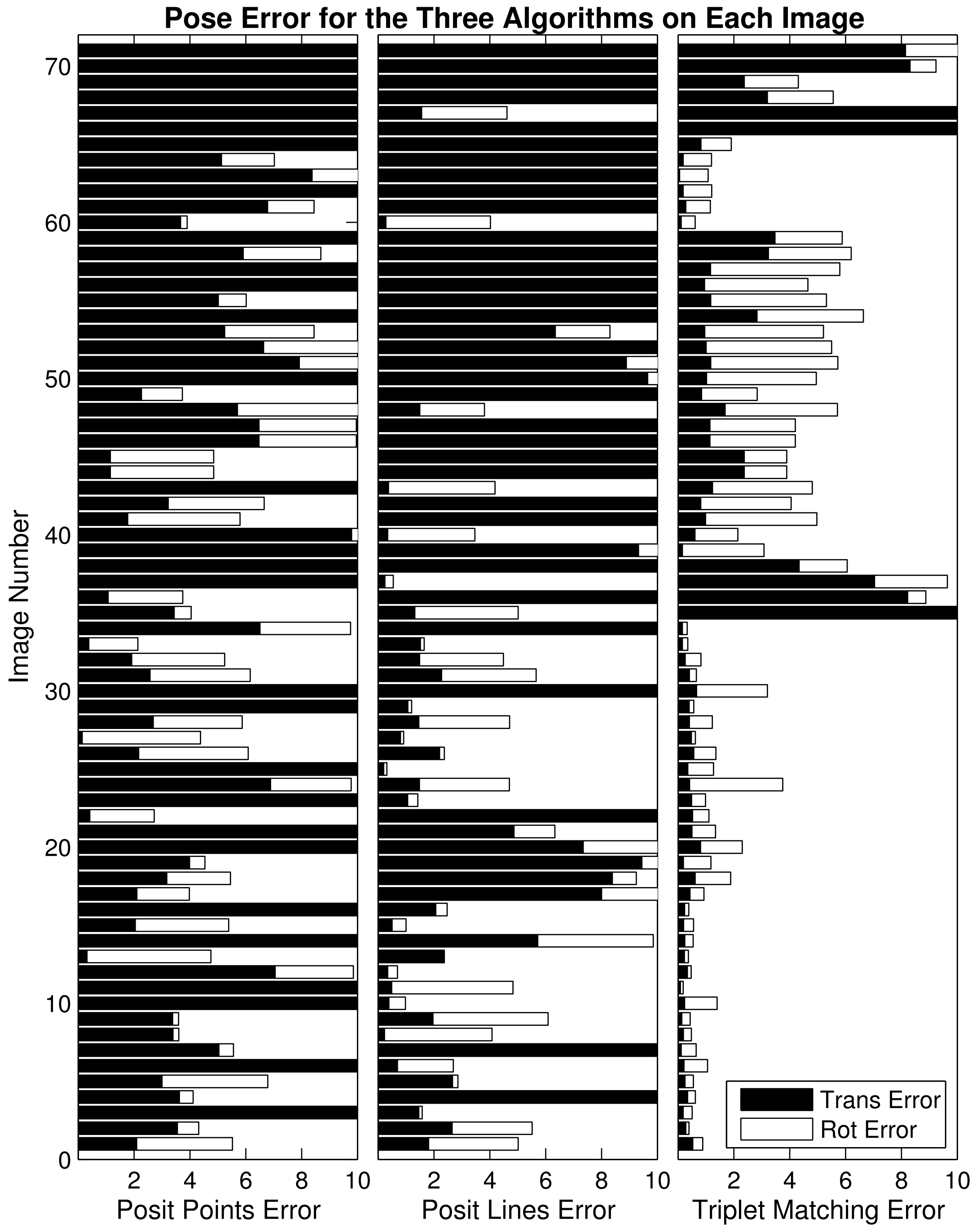


Figure 4.3: Total pose error of the three algorithms for each image in the cube image set. The translation error is given in cm while the rotation errors are lengths in the scaled consistent space (4.1).

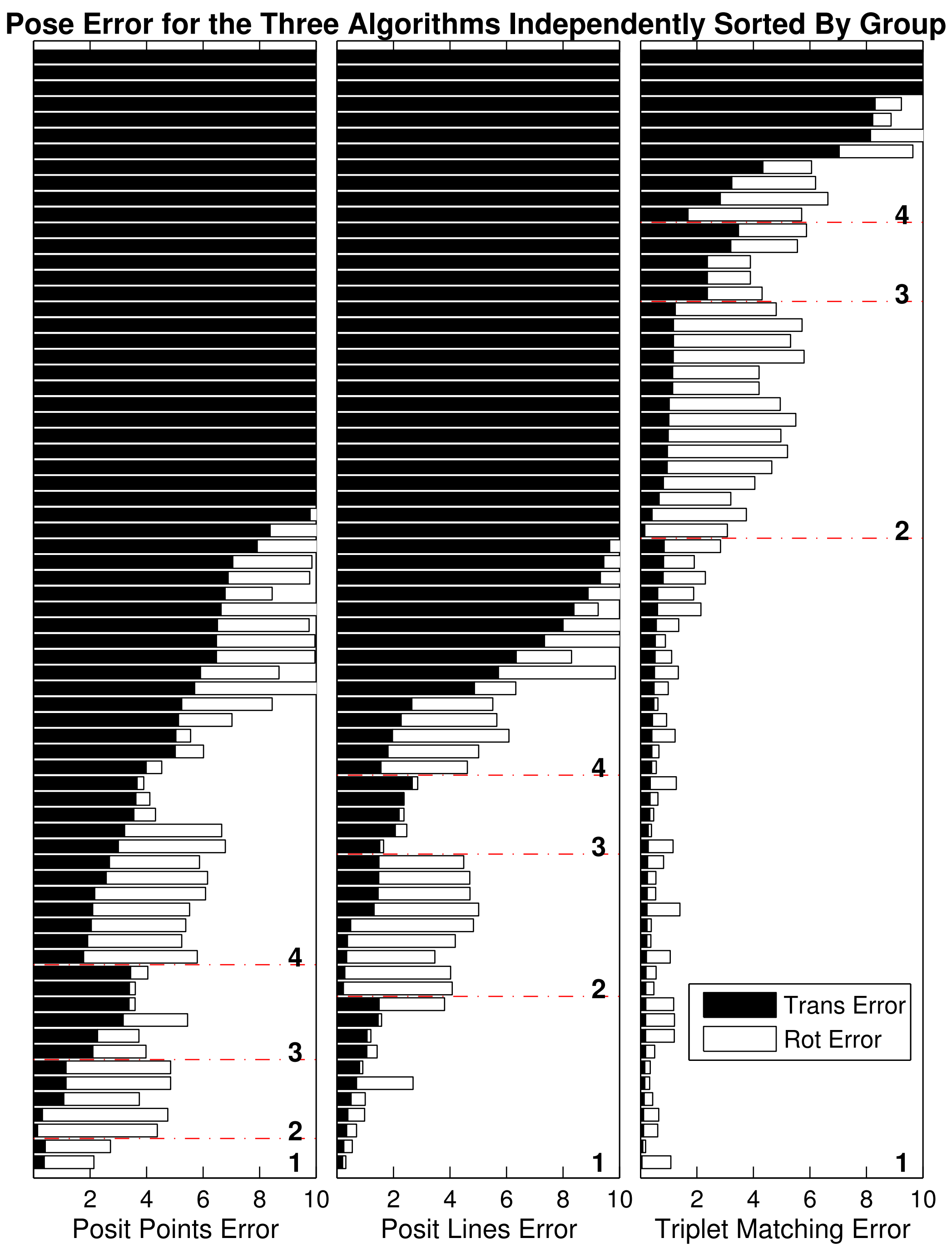


Figure 4.4: Pose error of the three algorithms on the cube image set. For each algorithm, results are sorted by total error and classified. The dotted lines indicate class boundaries and the numbers indicate the class labels. Table 4.1 shows the requirements of each class.

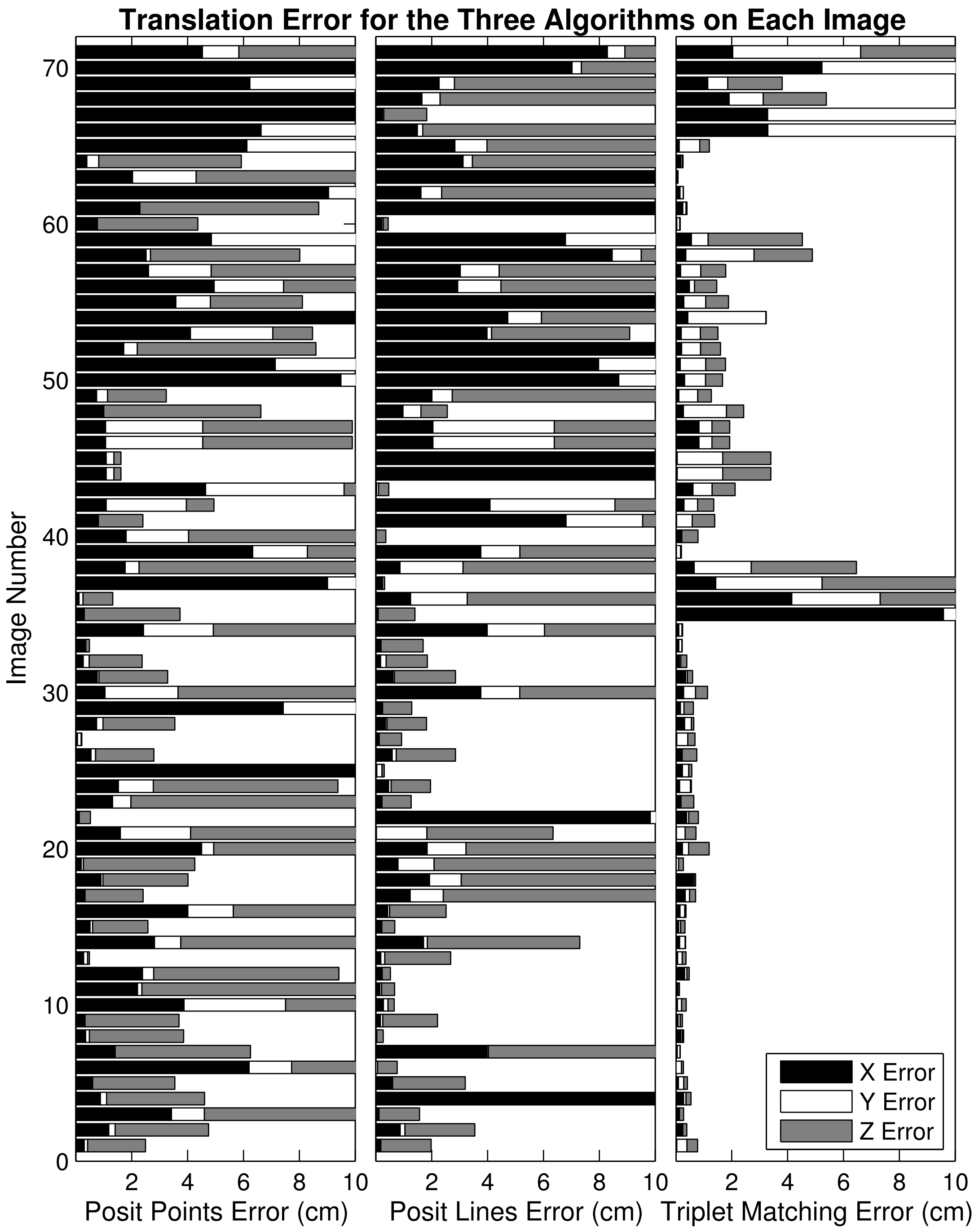


Figure 4.5: Breakdown of the translational error for the three algorithms for each image in the cube set. Errors are given in cm

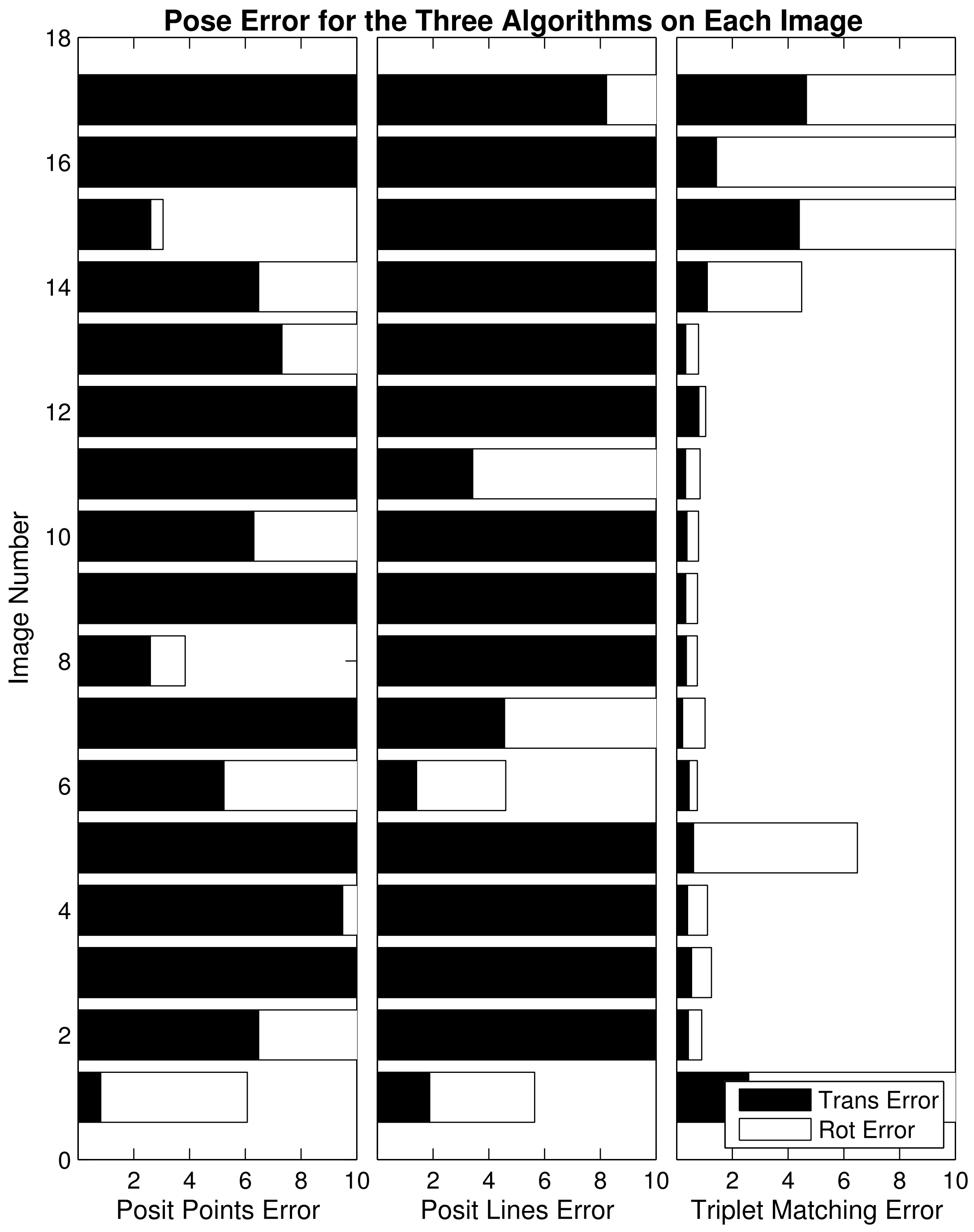


Figure 4.6: Total pose error of the three algorithms for each image in the assembly image set. The translation error is given in cm while the rotation errors are lengths in the scaled consistent space (4.1).

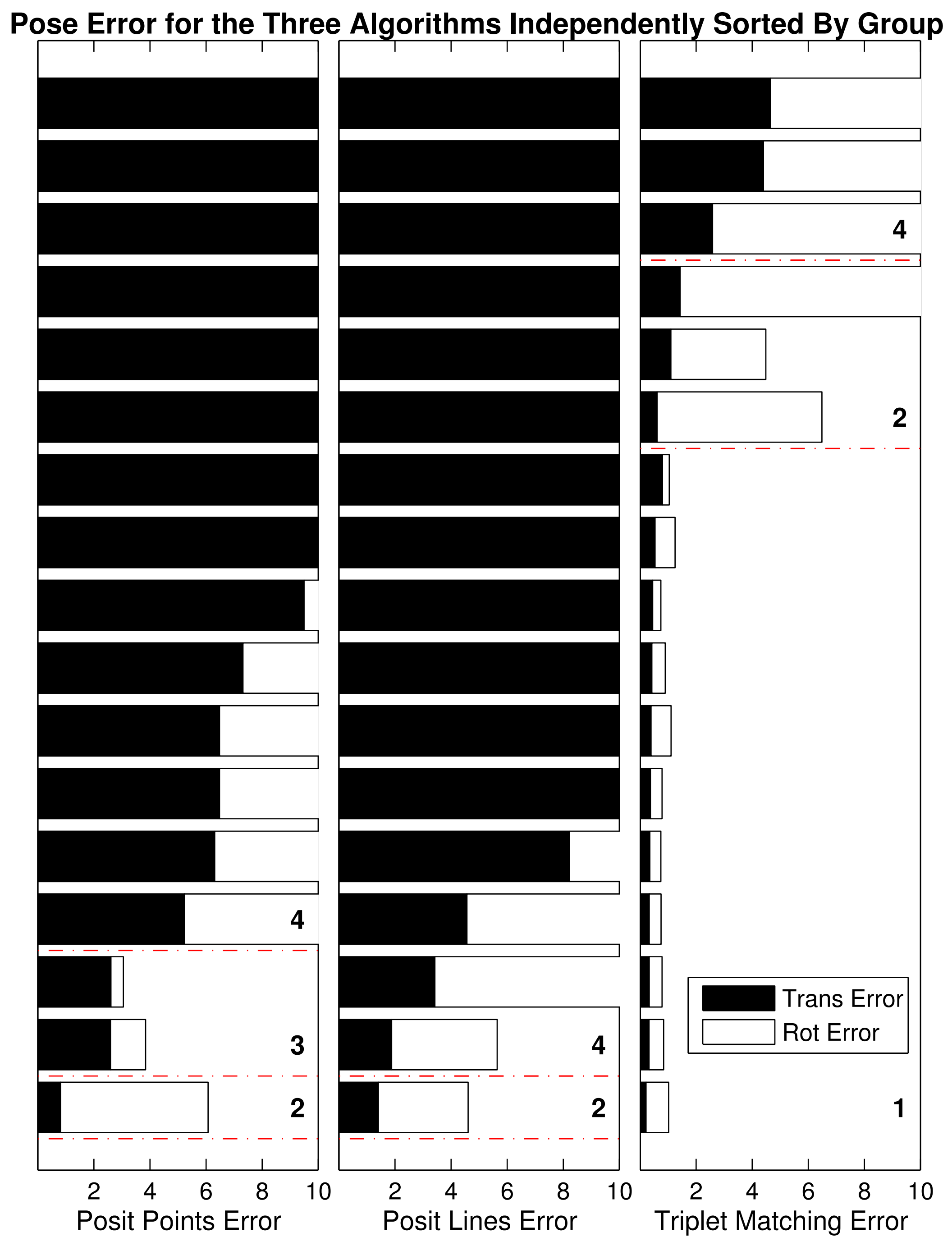


Figure 4.7: Pose error of the three algorithms on the assembly image set. For each algorithm, results are sorted by total error and classified. The dotted lines indicate class boundaries and the numbers indicate the class labels. Table 4.1 shows the requirements of each class.

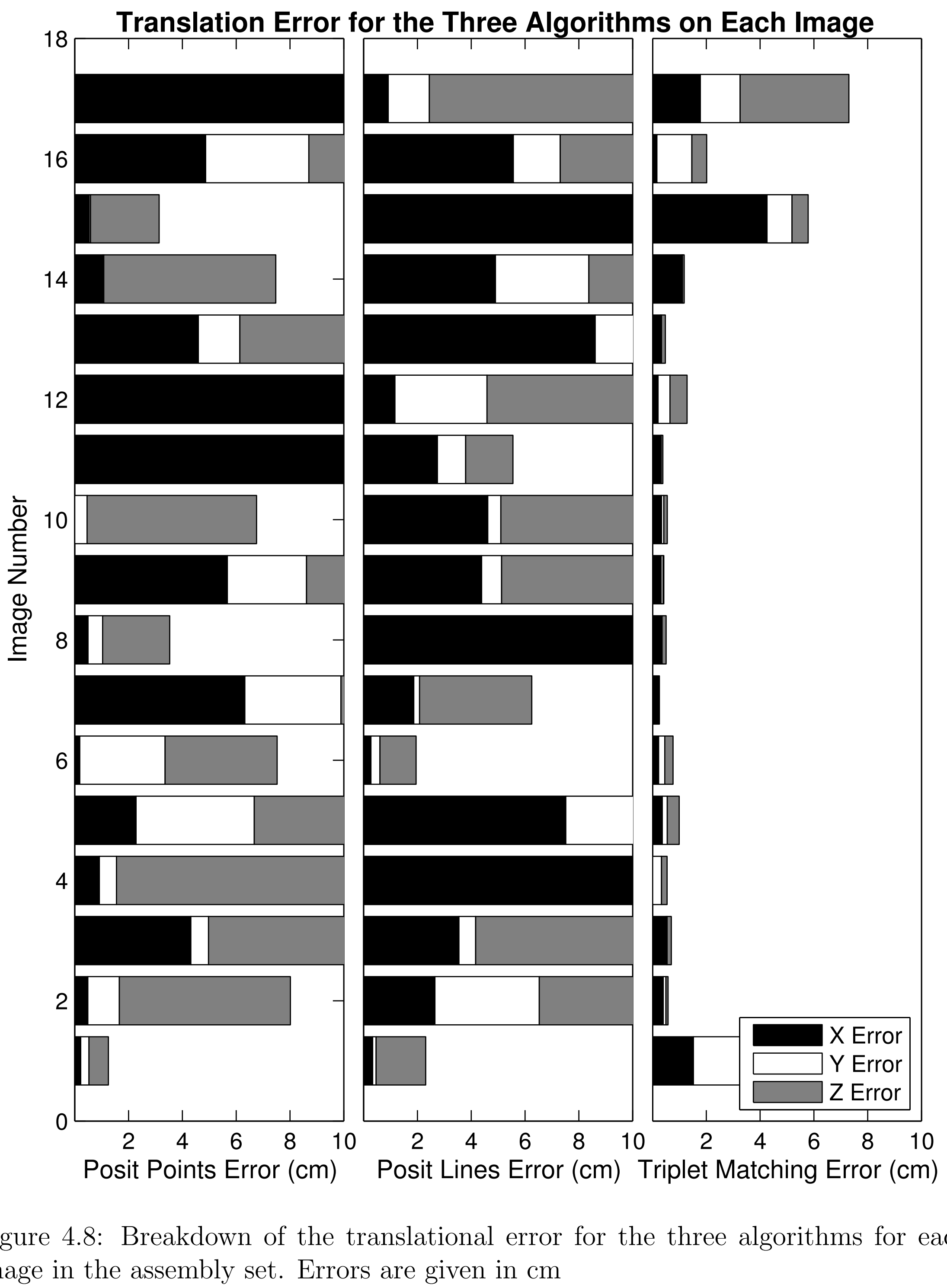


Figure 4.8: Breakdown of the translational error for the three algorithms for each image in the assembly set. Errors are given in cm

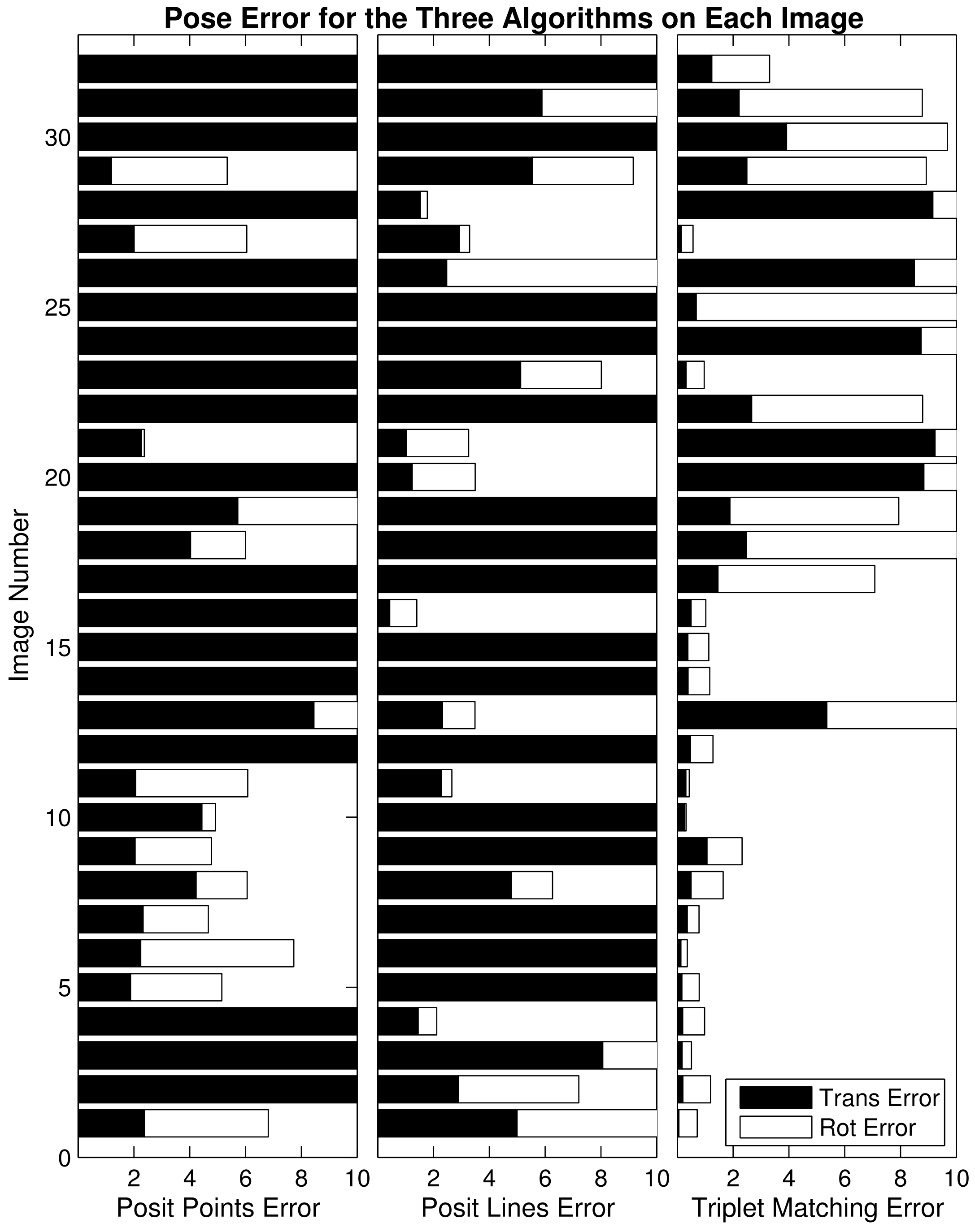


Figure 4.9: Total pose error of the three algorithms for each image in the cuboid image set. The translation error is given in cm while the rotation errors are lengths in the scaled consistent space (4.1).

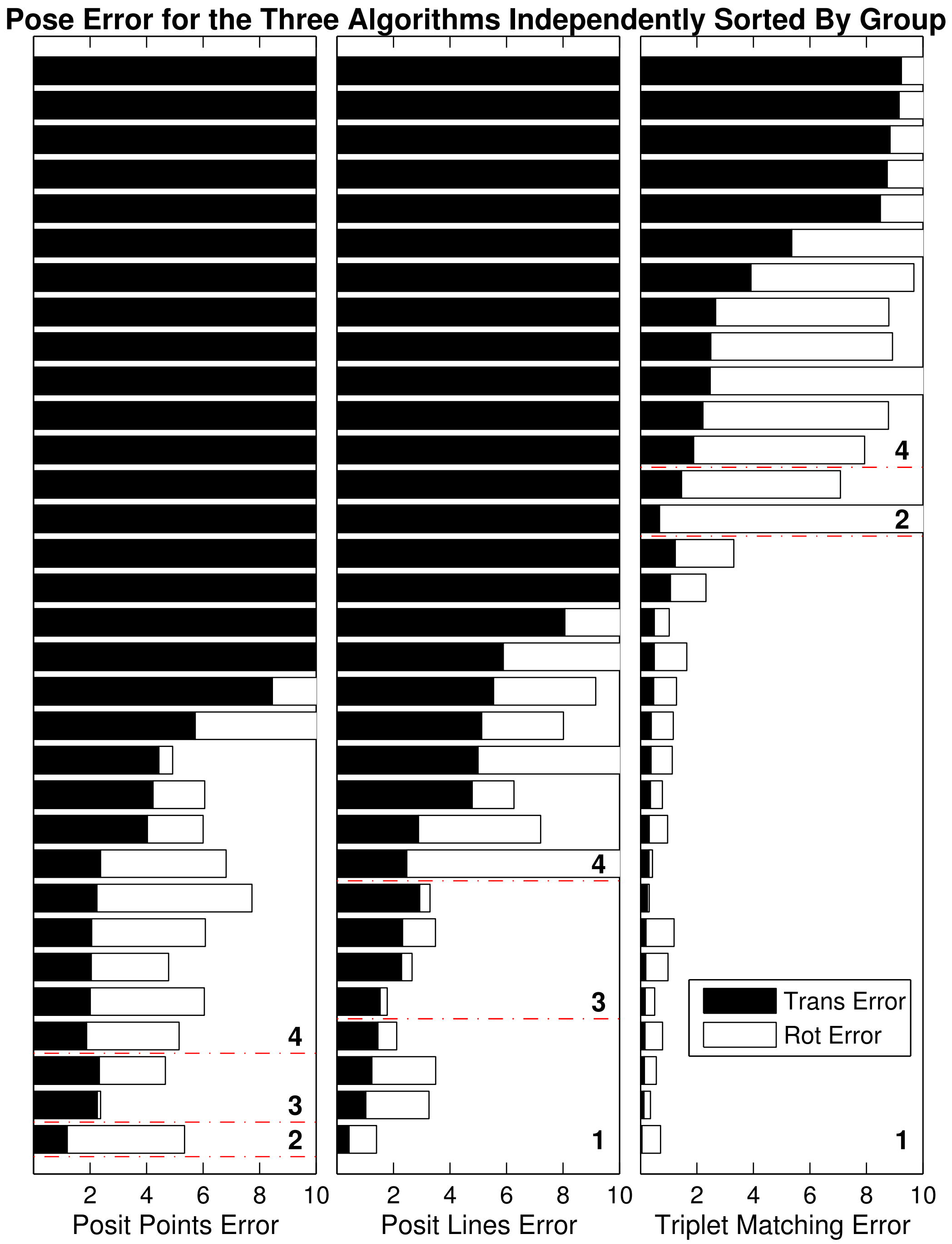


Figure 4.10: Pose error of the three algorithms on the cuboid image set. For each algorithm, results are sorted by total error and classified. The dotted lines indicate class boundaries and the numbers indicate the class labels. Table 4.1 shows the requirements of each class.

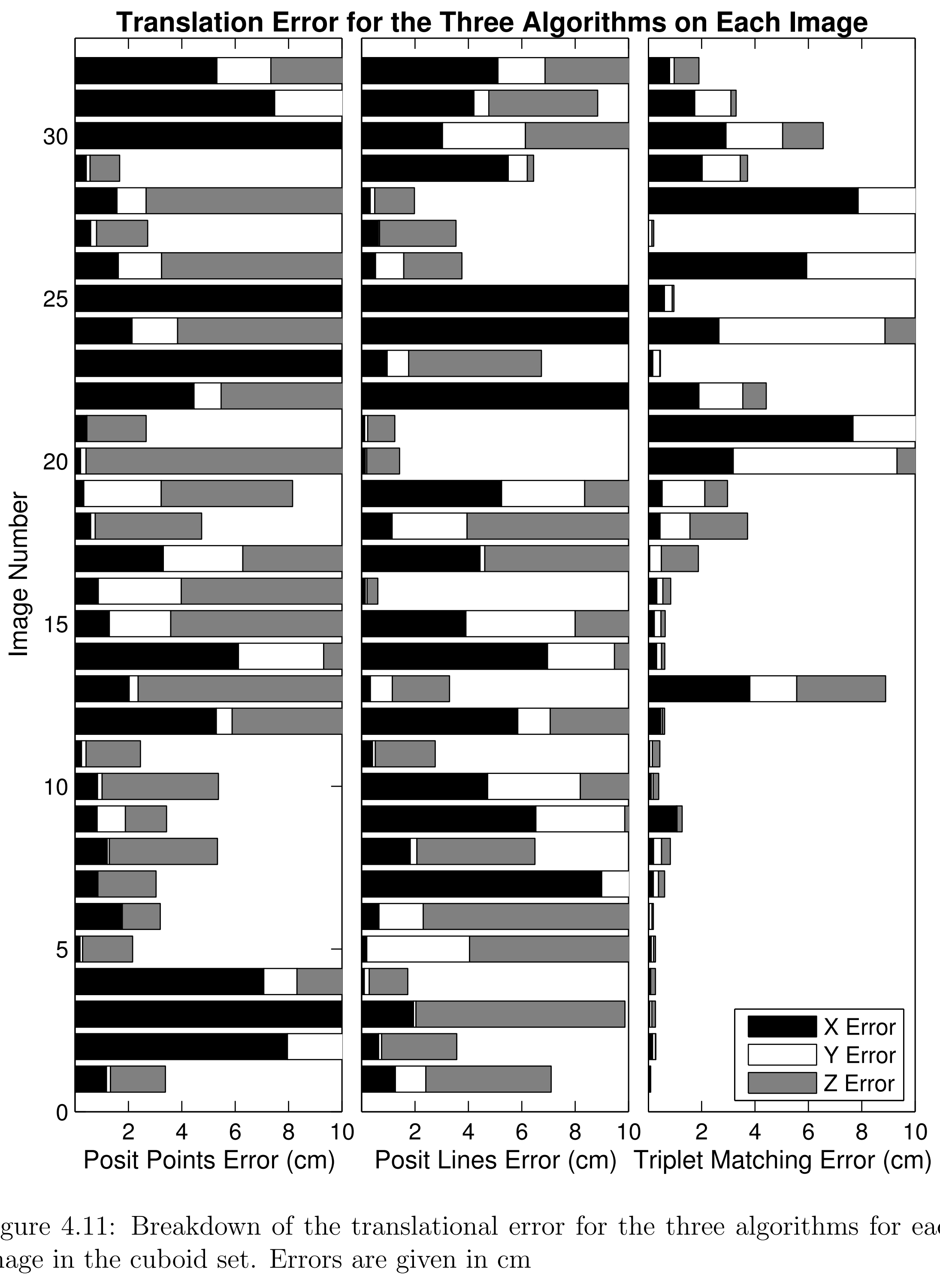


Figure 4.11: Breakdown of the translational error for the three algorithms for each image in the cuboid set. Errors are given in cm

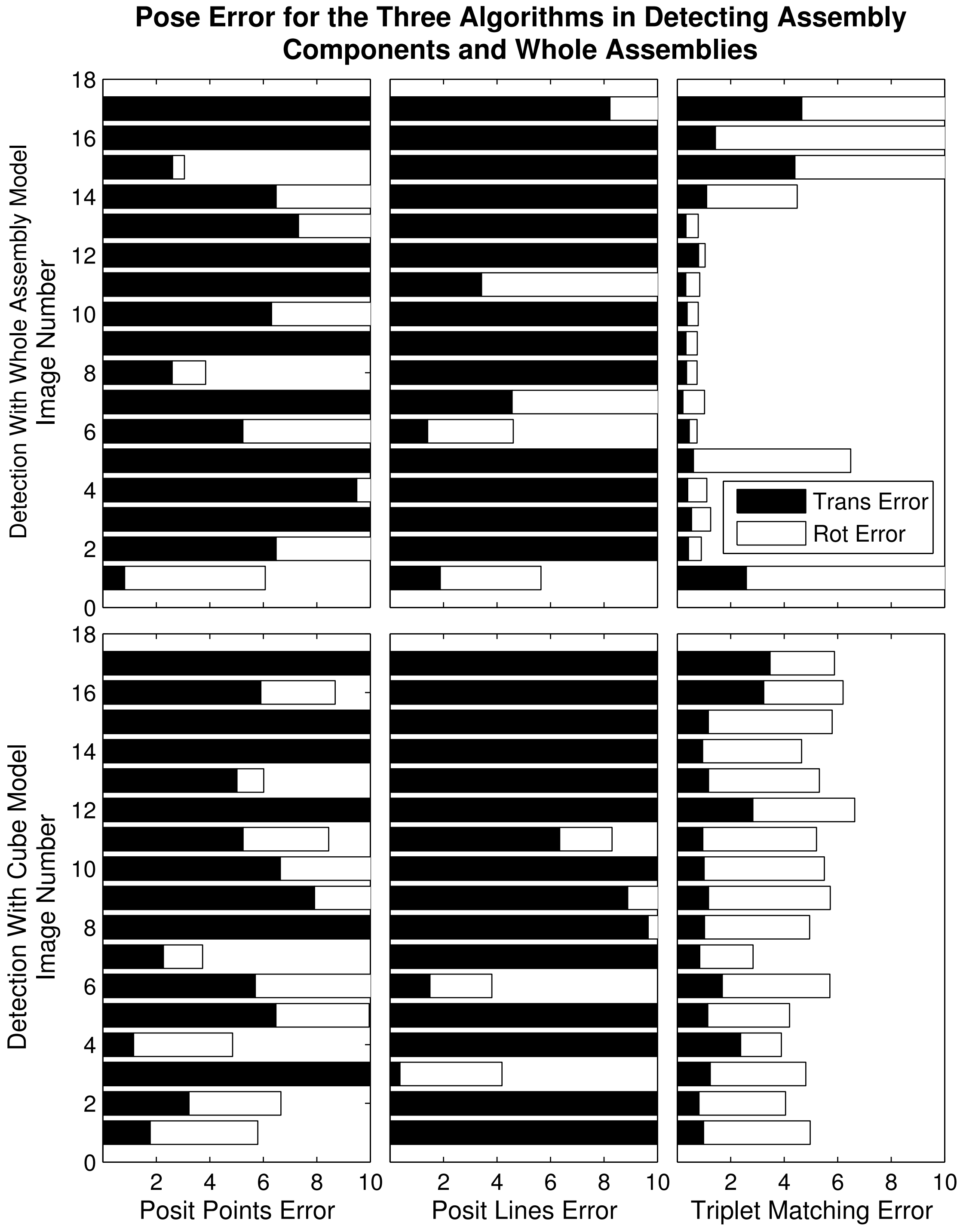


Figure 4.12: Total error for the three pose estimation algorithms on the assembly set. The first row shows the results of trying to find the assembly using the assembly as the model, while the second rows shows the results of finding the assembly using only the cube as the model.

a pixel or two will not have a large bearing on the objects X,Y location in space but it does greatly affect the objects Z location in space. This relationship can be seen in equation (2.3) which shows that an images x,y coordinates are proportional to an objects X,Y world coordinates by a factor given by Z. Thus small changes in x or y correspond to larger changes in Z. If better feature localization were possible many of the Class 3 poses could be improved to Class 1 poses since this would reduce the error in the Z direction.

The lines algorithm does not suffer from this localization issue quite as much as the points algorithm because lines do not require localization to a single point. For this reason the lines algorithm tends to have more Class 1 members than Class 3 members and also consistently has more Class 1 members than the points algorithm.

Another limitation to both of these algorithms is mismatching features. This is especially a problem with highly rotation symmetric objects like the cube and cuboid because these objects can assume many poses where model features will line up with image features and only one of the possible poses is actually correct. These types of poses belong to Class 2 see Figure 4.13 for examples. In both of these instances the poses visually appear to be correct to the naked eye and indeed many of both the line and point features appear to be correctly matched, but in actuality both poses have a rotational error of 4 or more units in the scaled consistent space.

The 'fitness' test for accepting the poses returned by the algorithms is another limiting factor in the correctness of the final pose accepted. If the requirements for the fitness test were increased so that lines have to be closer than 4 pixels apart on average or more than 5 model and image lines must match up then better poses may be obtainable. However, if the acceptance criteria are made too strict then none of the poses returned by the algorithm may ever be accepted. It is also hard to determine a good number for the matching requirement because depending upon the view of the

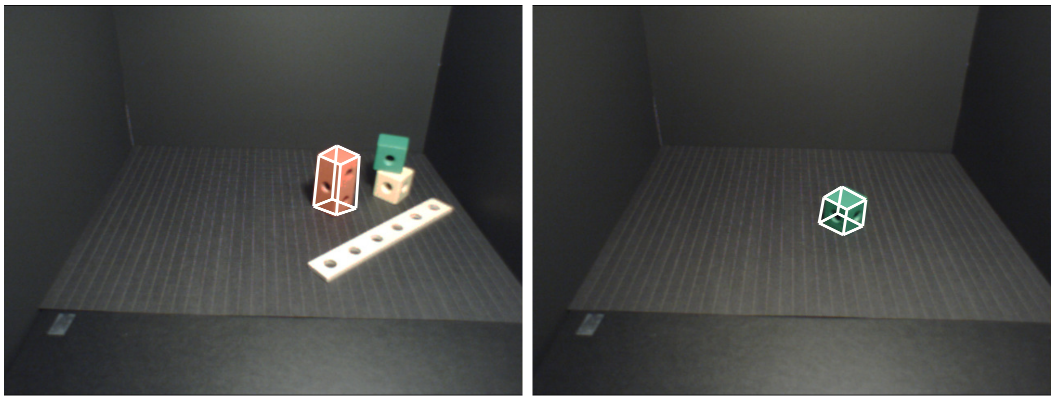

(a) SoftPOSITPoints Class 2 pose example (b) SoftPOSITLines Class 2 pose example

Figure 4.13: Two example results images from Class 2. Both of these poses illustrate instances where poses are perceptually correct and features are matched, however the correspondences are incorrect. The white lines indicate the final pose estimated by the algorithm.

object different numbers of lines are visible in the image.

4.2.2 Binocular Method

The triplet matching algorithm was the one binocular algorithm run on the dataset. For each of the three subsets the triplet matching algorithm was able to return a Class 1 pose on more than 50% of the images. For the assembly case Figures 4.6,4.7,4.8 the algorithm shows the best convergence results of any of the three sets. The improved results are due to the fact that the assembly model is larger and has a more unique shape than the other models thus there are more unique triplets which can be generated and matched by the algorithm than with the other models. For example any three points chosen on a single face of the cube could be matched to three points on any other face of the cube because all of the faces are identical. With the assembly model three points can be chosen which are spaced far enough apart so that they can only be matched back to only one side of the assembly.

Figure 4.12 shows the results of all three algorithms in trying to find the assembly using the entire assembly as the model and using only the cube as the model. For the triplet matching algorithm the pose results are vastly improved when using the entire assembly as the model as compared to using only the cube as the model. In fact when trying to find the assembly using only the cube as the model none of the results were in Class 1, whereas with the assembly model approach 66% of the poses were in Class 1. These results indicate that the performance of the algorithm is improved when the model has a more distinct geometry and can generate more distinct triplets for matching.

One limitation of using the hierarchal binning approach is that if the initial binning returns an incorrect result then the second level of binning is guaranteed to return an incorrect pose. If the binning is done only at a single level then the likelihood of a single bin filling from incorrectly matched triplets is reduced because each bin covers a smaller space and the incorrect matches that accumulate in one big bin would be dispersed throughout smaller bins. As discussed before the single level binning approach can take longer to run and requires more memory.

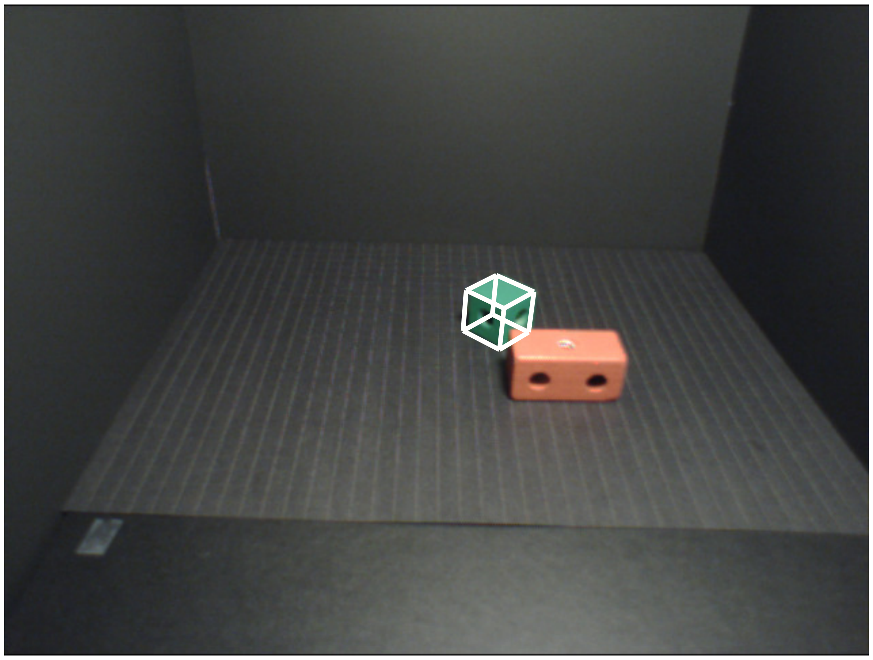
Unlike with the two SoftPOSIT algorithms no pose verification step was performed with this algorithm. Had the same pose verification step been included in this algorithm then the overall performance of the algorithm could likely have been improved. In fact the pose verification would probably work even better because this algorithm does not attempt to match or align features when it generates poses so the line features of the model and image are not predisposed to being aligned unless the pose returned by the algorithm is actually correct.

4.2.3 Overall Effectiveness of Both Methods

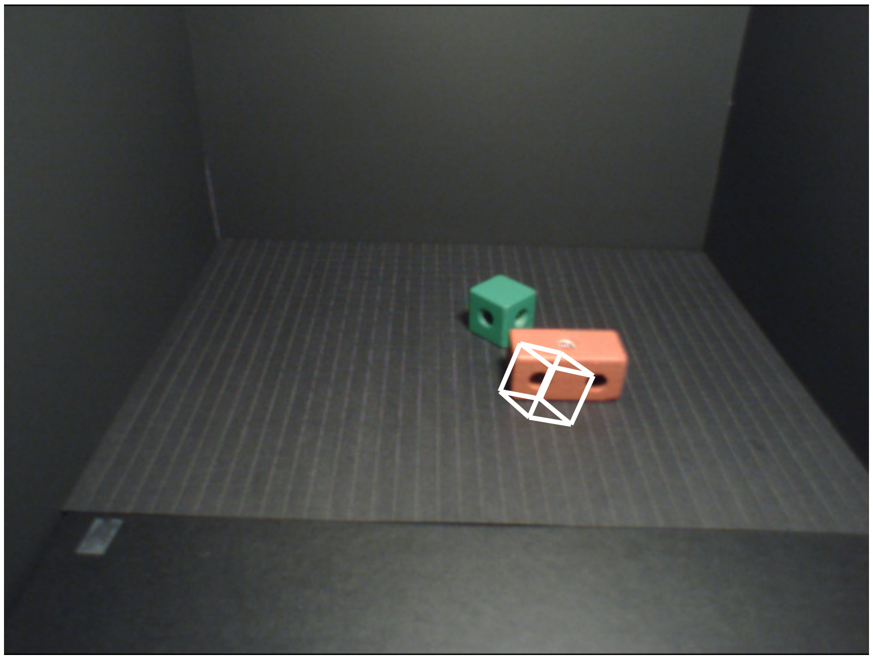
Out of the three algorithms presented the triplet matching algorithm is clearly the most effective both in terms of accuracy in overall pose and likelihood of generating the correct pose. However, there are some interesting instances in which the SoftPOSIT algorithms actually outperform the triplet matching algorithm.

Figure 4.14 shows one instance where the SoftPOSITLines algorithm outperformed the triplet matching algorithm, specifically the SoftPOSITLines algorithm returned a Class 1 pose while the triplet matching algorithm returned a Class 4 pose. In this instance the goal was to detect the green cube, and in this particular image the triplet matching algorithm had a hard time distinguishing between triplets belonging to the faces of the cuboid and the cube. The lines algorithm on the other hand was able to accurately match the lines belonging to the cube to the model lines of the cube. Had the red block been rotated more to expose its end then it is more likely that the lines algorithm would converge to the end of the red cuboid as there would be more lines added to that location.

For the case of detecting the cuboid there was three different instances where the lines algorithm was able to outperform the triplet algorithm and even a few cases where the points algorithm was able to outperform the triplet algorithm. Figure 4.15 shows two instances where both variants of the SoftPOSIT algorithm outperform the triplet matching algorithm. In the first image the points algorithm returned a Class 2 pose, the lines algorithm returned a Class 1 pose, and the triplet algorithm returned a Class 4 pose. In this image there were only 4 corners and lines detected on the strip in the image so the algorithms had an easy time in converging to the cuboids features. However the appearance of the strip is very similar to the face of the red cuboid so the triplet algorithm actually converged to the strips surface. In the second



(a) SoftPOSITLines



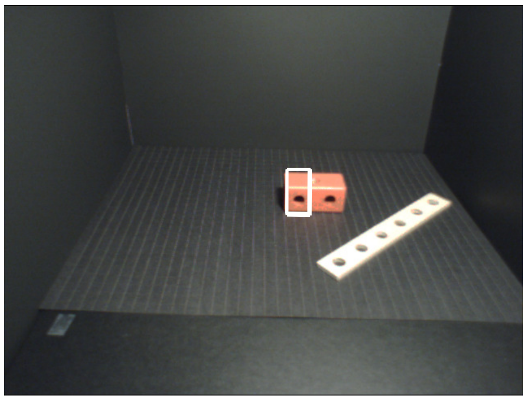
(b) Triplet Matching

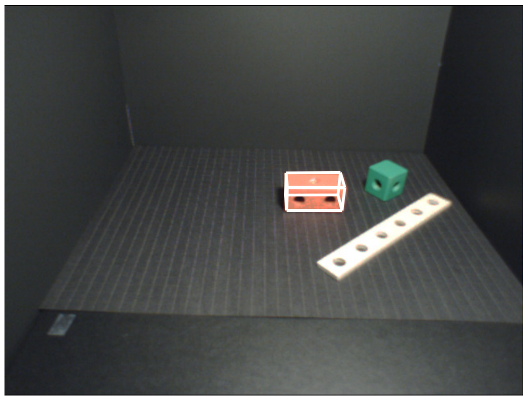
Figure 4.14: Example image where the SoftPOSITLines algorithm outperforms the triplet matching algorithm. The goal is to identify the pose of the green cube. The white wire frames show the poses estimated by the two algorithms. In this instance the triplet matching algorithm incorrectly identified the red cuboid as the green cube.

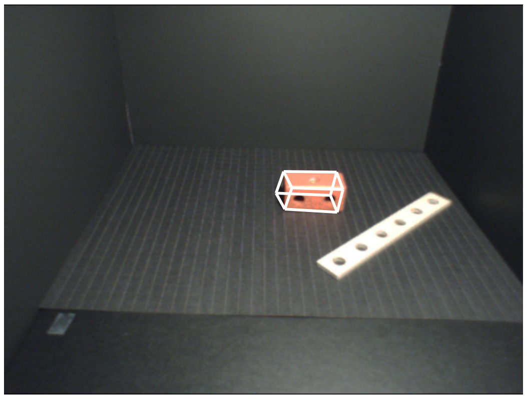
image both of the SoftPOSIT variants returned Class 1 poses and again the triplet algorithm returned a Class 4 pose. The success and failures were all due to the same factors as the other image.

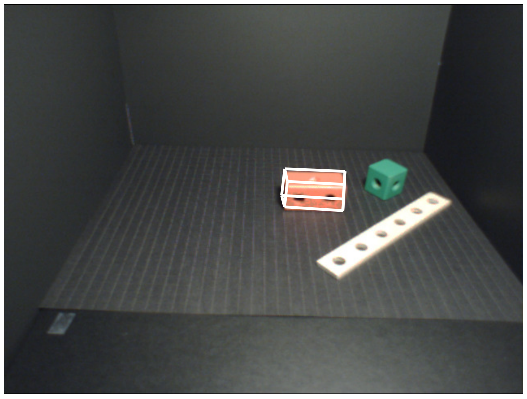
In nearly all of the cases where the SoftPOSIT algorithms were able to beat the triplet matching algorithm, there were some other objects in the scene which the triplet matching algorithm converged to. Thus they did not provide more accurate pose results but simply managed to converge to the correct object rather than other objects in the scene. There were a few instances however where the SoftPOSITLines algorithm gave more accurate pose results than the triplet matching algorithm and both algorithms gave Class 1 results. Image number 25 in Figure 4.3 shows one instance were both the lines algorithm and triplet algorithm returned Class 1 poses and the lines pose is more accurate. In terms of rotational accuracy only the SoftPOSIT-Lines algorithm was able to return poses with comparable and sometimes even better accuracy than the triplet matching algorithm. This is due to the increased stability of line feature extraction and localization as compared to point feature extraction and localization. The SoftPOSITPoints algorithm rarely returned rotations as accurate as the triplet algorithm which is due to the difficulty in detecting and localizing the corners of the objects in an image.

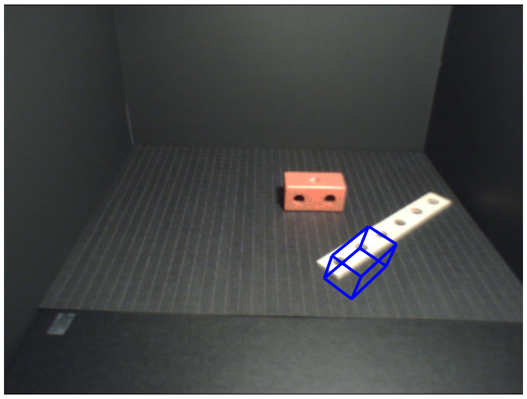
For the images containing only the object of interest the triplet matching algorithm generally returned a Class 1 pose. The two SoftPOSIT variants did not display this sort of consistency. They demonstrate random convergence across the image sets. This behavior is consistent with the way in which the algorithms were run because the poses returned by the algorithm are depend on the initial poses and the initial poses were chosen at random. Thus, the consistency of converging to the correct pose is random.


(a) SoftPOSITPoints


(b) SoftPOSITPoints


(c) SoftPOSITLines Matching


(d) SoftPOSITLines


(e) Triplet Matching

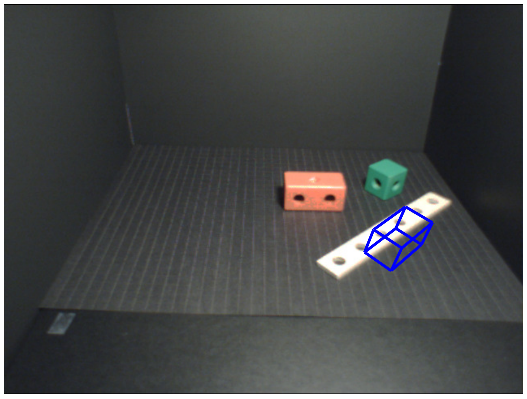

(f) Triplet Matching

Figure 4.15: Two Example images (one per column) where the SoftPOSIT algorithms outperform the triplet matching algorithm. The goal is to identify the pose of the red cuboid. The wire frames show the poses estimated by the algorithms. In both instances the triplet matching algorithm incorrectly identifies the surface of the stick as a surface of the red cuboid.

Chapter 5

Conclusions and Discussion

With the three algorithms in their given state the triplet algorithm is overall the best performer. Out of the two monocular pose estimation methods the SoftPOSITLines variant is the better performer.

If some of the weaknesses of the SoftPOSITLines algorithm discussed prior were addressed, the performance could be improved. Firstly the line extraction could be made more robust. If the lines were more accurately detected and localized then the problem of determining the correct Z component for the pose could be mitigated and the translation results of the lines algorithm would be able to approach those of the triplet matching algorithm. This improvement would shift Class 3 poses back into Class 1. Secondly the lines algorithm could be improved by taking into account model self occlusion. This would prevent lines in the model which are not visible given the current pose of the object from being matched to image lines. This improvement would help reduce the number of Class 2 poses and increase the number of Class 1 poses.

A future variant to the SoftPOSITLines algorithm could be a StereoPOSIT-Lines algorithm. This would be able to improve feature matching even more than simply taking into account self occlusion as the locations of lines in space could be determined. This would enable intelligent matching of model and image lines so that lines on the model closest to the camera given the current pose are only matched to lines in the images which are closest to the camera. The inclusion of stereo data would also enable more intelligent guessing of initial translations since the locations of point clouds in space is already known.

To improve the results of the pose clustering algorithm, Hillenbrand examines the addition of normals at each point in the triplets to add more constraints to the triplet matching algorithm [22]. The addition of the normals reduces the number of incorrect triplet matches which can be generated and increases the likelihood of generating correct poses when matching. The results could also be improved by using models which are generated by using multiple views and thus represent the whole object rather than its appearance from just a single viewpoint. If the SoftPOSITLines algorithm were improved then the rotations returned by the triplet matching algorithm could be refined by the SoftPOSITLines algorithm to help reduce the rotational errors in the triplet matching algorithm.

Although these changes may help with improving the results of these algorithms, the results of this experiment seem to indicate that determining the pose of these objects is generally hard to accurately and consistently do. All of these objects have low feature counts, minimal surface texture, and are highly rotationally symmetric. These aspects make it hard for any of these pose estimators to consistently and accurately work. More research needs to be conducted to see if these sparse feature sets can be combined to make a richer feature set, or if some of these pose estimation algorithms can be combined to create a pose estimator that can use multi modal data such the proposed StereoPOSITLines algorithm or even a combination of the two SoftPOSIT algorithms.

Bibliography

[1] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 2003.

[2] X. Armangu, J. Salvi, and J. Batlle. A comparative review of camera calibrating methods with accuracy evaluation. *Pattern Recognition*, 35:1617–1635, 2000.

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

[4] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8(6):679–698, November 1986.

[5] Xiao Chen He and Nelson H C Yung. Corner detector based on global and local curvature properties. *Optical Engineering*, 47(5), 2008.

[6] Alvaro Collet Romea, Dmitry Berenson, Siddhartha Srinivasa, and David Ferguson. Object recognition and full pose registration from a single image for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA '09)*, May 2009.

[7] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[8] Philip David, Daniel DeMenthon, Ramani Duraiswami, and Hanan Samet. Simultaneous pose and correspondence determination using line features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:424, 2003.

[9] Philip David, Daniel DeMenthon, Ramani Duraiswami, and Hanan Samet. Softposit: Simultaneous pose and correspondence determination. *International Journal of Computer Vision*, 59:259–284, 2004.

[10] Daniel F. DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[11] D H Douglas and T K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[12] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[13] Frederic and Jurie. Tracking objects with a recognition algorithm. *Pattern Recognition Letters*, 19(3-4):331 – 340, 1998.

[14] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12:16–22, 2000.

[15] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377 –388, Apr 1996.

[16] Iryna Gordon and David G Lowe. What and where : 3D object recognition with accurate pose. *Lecture Notes in Computer Science*, 4170(1):67–82, 2004.

[17] T. Grundmann, R. Eidenberger, M. Schneider, M. Fiegert, and G. Wichert. Robust high precision 6D pose determination in complex envi- ronments for robotic manipulation. In *Workshop Best Practice in 3D Perception and Modeling for Mobile Manipulation at the Int. Conf. Robotics and Automation*, pages 1–6, 2010.

[18] R.M. Haralick, D. Lee, K. Ottenburg, and M. Nolle. Analysis and solutions of the three point perspective pose estimation problem. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 592–598, Jun 1991.

[19] C Harris and M Stephens. *A Combined Corner and Edge Detection*, volume 15, pages 147–151. 1988.

[20] Chris Harris. *Tracking with rigid models*, pages 59–73. MIT Press, Cambridge, MA, USA, 1993.

[21] Ulrich Hillenbrand. Pose clustering from stereo data. In *Proceedings of VISAPP International Workshop on Robotic Perception*, pages 23–32, 2008.

[22] Ulrich Hillenbrand and Alexander Fuchs. An experimental study of four variants of pose clustering from dense range data. *Computer Vision and Image Understanding*, 115(10):1427–1448, 2011.

[23] R. Horaud, B. Conio, O. Leboulleux, and B. Lacolle. An analytic solution for the perspective 4-point problem. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1989*, pages 500 –507, Jun 1989.

[24] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.

[25] V. Lepetit and P. Fua. Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.

[26] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[27] David G Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, 1991.

[28] D.G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999.

[29] Eric Marchand and Francois Chaumette. Virtual visual servoing: a framework for real-time augmented reality. *Computer Graphics Forum*, 21(3):289–297, 2002.

[30] Z. Marton, D. Pangercic, N. Blodow, J. Kleinehellefort, and M. Beetz. General 3D modelling of novel objects from a single view. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3700–3705, Oct 2010.

[31] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615 –1630, oct. 2005.

[32] T.Q. Phong, Radu P. Horaud, A. Yassine, and P.D. Tao. Object pose from 2D to 3D point and line correspondences. *International Journal of Computer Vision*, 15(3):225–243, July 1995.

[33] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[34] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. In *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, Oct 2010.

[35] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[36] G. Simon, A.W. Fitzgibbon, and A. Zisserman. Markerless tracking using planar structures in the scene. In *Augmented Reality, 2000. (ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 120 –128, 2000.

[37] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[38] Chien-Ping Lu Suguna Pappu Steven Gold, Anand Rangarajan and Eric Mjolsness. New algorithms for 2D and 3D point matching: pose estimation and correspondence. *Pattern Recognition*, 31(8):1019 – 1031, 1998.

[39] Ulrich and Hillenbrand. Consistent parameter clustering: Definition and analysis. *Pattern Recognition Letters*, 28(9):1112 – 1122, 2007.

[40] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.