AN EVOLUTIONARY ALGORITHM FOR DEPTH IMAGE BASED CAMERA POSE ESTIMATION IN INDOOR ENVIRONMENTS

by

Ang Lu

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2016

Table of Contents

List of Tables

List of Figures

Abstract

We consider the task of determining the pose of a depth camera based on a single target depth image and a 3D model of the indoor environment that the image was taken in. We identify the quality of a pose estimate with summed differences between depth values in the target depth image and a depth image generated synthetically by using that pose estimate in the 3D model. We then propose an evolutionary algorithm for optimizing pose estimates.

In this thesis, we discuss indoor positioning approaches, introduce our evolutionary algorithm, and then evaluate the performance of that algorithm in three artificial test environments. Finally, we discuss the perspectives for the use of the algorithm in real environments.

List of Abbreviations and Symbols Used

Abbreviations

CMA-ES Covariance Matrix Adaptation Evolutionary Strategy

CSA Cumulative Step Size Adaptation

ES Evolutionary Strategies

GNSS Global Navigation Satellite Systems

ICP Iterative Closest Point

NDT Normal-Distributions Transform

RANSAC Random Sample Consensus

RFID Radio Frequency Identification

SLAM Simultaneous Localisation and Mapping

ToF Time-of-Flight

Symbols

$\sigma \in \mathbb{R}^+$ a step size

$\mathcal{P}$ the population of selected offspring

$\mu \in \mathbb{N}$ the number of parents

$\lambda \in \mathbb{N}$ the number of offspring

$\mathbf{x}$ a solution vector/part of a solution vector

$\mathbf{s}$ a search path

c the cumulation parameter

$\mathbf{z}_k$ a single mutation vector

$n \in \mathbb{N}$ the search space dimension

$\mathbf{m}$ the mean value of the search distribution

S the unit sphere embedded in $\mathbb{R}^4$

$T_{\mathbf{q}}S^3$ the tangent space of the unit sphere in $\mathbb{R}^4$ at the location of unit quaternion $\mathbf{q}$

$\alpha \in \mathbb{R}^+$ a step scale factor

$\tau \in \mathbb{R}^+$ a constant which controls the size of the mutations

$D \in \mathbb{R}^+$ the damping factor

$\mathbf{q}$ a four dimensional unit quaternion

$\mathbf{w} \in \mathbb{R}^4$ a standard normally distributed vector

w_i recombination weights

$\mathcal{N}(\mathbf{0}, \mathbf{I})$ a multivariate normal distribution with expectation and modal value $\mathbf{0}$ and the identity matrix $\mathbf{I}$

Acknowledgements

First of all, I wish to express my gratitude to my supervisor, Dr. Dirk Arnold, Faculty of Computer Science, Dalhousie University, for his valuable guidance and ideas that have helped me complete my research work and this thesis. I would like to thank my grandfather as well as my parents for their enthusiasm and endless support.

Chapter 1

Introduction

" –yes, that's about the right distance –but then I wonder what Latitude or Longitude I've got to? "
— L. Carroll, Chapter I, Alice's Adventures in Wonderland

This chapter introduces an overview of the thesis. First, we discuss our motivation and background. Then, we briefly introduce our method. After that, we summarize the main contributions of this thesis. Finally, we illustrate the structure and outline of the thesis.

1.1 Motivation and Background

Similar to the question that Alice asked herself when she entered the wonderland," – yes, that's about the right distance –but then I wonder what Latitude or Longitude I've got to? ", the problem of localization that we will discuss throughout this thesis is the task of locating objects or people in environments.

With the development of the Global Navigation Satellite Systems (GNSS), estimating the position of objects in an outdoor environment can be performed almost perfectly (7.8 meter horizontal accuracy for civilian usage [10]). At the same time, many indoor localization applications (which we will describe in the following Chapter) are looking for solutions to improve their performance of locating objects or people in indoor environments. Theoretically, most localization systems including GNSS can be used for indoor localization purposes. However, the GNSS is not sufficient for indoor localization tasks for several reasons: 1) GNSS signal will be attenuated and scattered by roofs, walls and other objects; 2) GNSS cannot provide an accurate orientation of objects; 3) positioning accuracy is finer (on a smaller scale) for indoor localization purposes. Therefore, the problem of estimating the position of objects in an indoor environment has become a focus of research during the past decade.

In [23], Mautz summarized many scenarios where the application of indoor localization is needed, such as location based service in indoor environments, context detection and situational awareness, intelligent transportation and police and firefighting work. The task of indoor localization also arises in connection with assembly and maintenance activities, where mechanics or engineers in the interior of the structure face a particular part or parts, and need to look up related information in parts catalogues. Manually looking up parts is tedious and time consuming, and it is desirable to automate the task as much as possible.

Indoor environments have some features [23] that are particularly challenging for localization purposes: 1) the structure of indoor environments (e.g. aircraft, church, museum) can be very different; the presence of people and the change of some movable objects in the environment (e.g. chairs in the classroom) may add more complexities; 2) the demand for precision and accuracy is higher than the demand for precision and accuracy in outdoor environments.

As mentioned in [23], indoor environments facilitate localization in several ways: 1) relatively small searching area; 2) low weather influences (no rain, storm, blizzard etc.); 3) fixed geometric constraints (e.g. walls, ceilings, floors).

1.2 Our Method

We explore opportunities arising from the increasing availability of commercial hardware for depth sensing, variably referred to as depth cameras or RGB-D cameras. A mechanic equipped with an RGB-D camera can take a depth image of the part or parts in question, plus their context in the assembled structure. That image can then be registered against a 3D model of the structure, thus yielding the pose (i.e., location and orientation) of the camera at the time that the image was taken.

We choose to perform registration based on depth images only, rather than additionally relying on RGB data. While discarding potentially useful information, the advantage is simplicity; there is no need to consider illumination conditions, which make colour a challenging issue to deal with. Moreover, as in our case, 3D models do not always have accurate colour information associated with them.

The computational task is closely related to problems in indoor positioning [23], mobile robot navigation [7], and object recognition [25], but it is distinguished from

these in several important respects:

1. We are able to take advantage of the existence of an accurate 3D model of the environment that the depth camera is located in.

2. Positioning must be accomplished based on visual information only. The deployment of hardware for near field communication based approaches is precluded.

3. We strive to compute the camera pose from a single image, as opposed to tracking a path through the environment.

4. We aim to determine the full camera pose (i.e., orientation in addition to location).

While each of these aspects has been considered in related work, we are not aware of approaches for settings where all of them hold simultaneously.

As proposed by Fillingham [8], the quality of a pose estimate is quantified by synthetically generating a depth image of the 3D model with that pose and then computing the sum of squared differences between that depth image and the target depth image. A challenge in developing an evolutionary algorithm for pose estimation is the representation of pose. It is desirable to represent camera orientation using unit quaternions in order to avoid problems with gimbal lock stemming from the use of Euler angle representations, or issues arising from redundancy in representations that use a greater number of parameters. Consequently, the variation operators of the evolutionary algorithm need to be designed to work in unit quaternion space.

1.3 Contributions

First, we present a simple evolutionary algorithm for the optimization of a pose vector. The algorithm is built on the more general approach for optimization on Riemannian manifolds proposed by Colutto et al. [6]. However, it differs in not adapting the full covariance matrix but instead using self-adaptation for controlling a scaling factor between the size of steps in the subspace of locations and those in the subspace representing orientation. Second, we provide a proof of concept for the possibility of inferring camera pose from a single depth image provided that a 3D model of the environment is available.

1.4 Publications

Portions of the work presented in this thesis have resulted in a paper [21], which can be found in the Proceedings of the IEEE Congress on Evolutionary Computation 2016.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Recent techniques and methods related to indoor localization and pose estimation are listed and discussed in Chapter 2. We divide the various techniques into categories and discuss the merits and limitations for each of them. Chapter 3 describes the representation of the camera, the objective function, and the algorithm we designed for the optimization of a pose vector. The evaluation of our algorithm in a unimodal environment, a classroom environment and a church environment is described in Chapter 4. Results for the various experiments are also explored in Chapter 4. In Chapter 5, we summarize the thesis and suggest possible directions for future work.

Chapter 2

Related Work

" I believe in God, Mozart, and Beethoven. "
— R. Wagner, Autobiographic Sketch

In this chapter, different techniques that have been developed for solving indoor localization and related problems are introduced; the advantages and disadvantages of these different techniques are discussed. Radio frequency identification (RFID) based indoor localization techniques are discussed in Section 2.1. A technology that is related to indoor localization and robotic mapping, called Simultaneous Localisation and Mapping (SLAM), is introduced in Section 2.2. In Section 2.3, different methods related to camera pose estimation are discussed. In Section 2.4, we discuss a task which is somewhat related to camera pose estimation in an indoor environment, called object pose estimation. A depth image based indoor localization technique is introduced in Section 2.5. Finally, we introduce the background of evolutionary strategies and discuss two major features in these strategies.

2.1 Radio Frequency Identification

Radio frequency identification (RFID) is a technology that has been designed to uniquely identify an object, animal, or person. RFID technology is widely used in access management, tracking of objects, transportation, identification and security etc. An RFID system consists of a reader and RFID tags. The reader, which has an antenna, can be used to interrogate the nearby RFID tags. An RFID tag is designed to provide data, which is usually used to uniquely identify an object. The data stored in an RFID device can be related to position information. Because of this, RFID has become a competitive technology for indoor localization purposes, even though it is not designed for this.

Ni et al. [26] present an RFID based indoor localization system. In this system,

RFID devices are set up inside a building. While a reader is traveling around the building and interrogating the RFID tags, the system can get the location information of that reader. Typically, the accuracy of an RFID localization system is highly dependent on the density of tags that are set in the environment and the maximal reading ranges of the reader. This system directly provides position information but cannot get orientation information from the sensor.

2.2 Simultaneous Localisation and Mapping

Simultaneous Localisation and Mapping (SLAM) is a category of techniques that has been developed for indoor localization and robotic mapping. It is originally developed by Leonard and Durrant-Whyte in [17]. These techniques help a robot to simultaneously generate a map of environments while it wanders around unknown areas. A SLAM system needs a mobile robot and a range measurement device (e.g. a laser scanner). Many SLAM algorithms have been applied in self-driving cars, robots, underwater vehicles and planetary rovers. Different types of sensors are used in a SLAM algorithm. These sensors can be categorized into laser-based, sonar-based, and vision-based systems. Different algorithms have been invented to manipulate different kinds of input. In [3], Aulinas et al. provide a survey of different types of SLAM techniques.

Zou et al. [40] propose an approach to indoor positioning using depth cameras that does not require a 3D model of the scene. Instead it builds such a model on the fly from the sequence of depth images taken. Depth cameras can provide valuable information for simultaneous localization and mapping, and random sample consensus (RANSAC) is used to establish relative transformations between frames based on point-wise correspondence between feature points. The proposed SLAM approach [40] has been successfully applied in an indoor environment consisting of several rooms connected by hallways. However, the problem considered differs from that faced by us in that we assume the existence of an accurate 3D model, but strive to estimate camera pose from a single depth image rather than tracking a path through the environment.

2.3 Camera Pose Estimation

Since many indoor localization problems focus on extracting the position of viewers, these indoor localization problems are hence similar to the problems of estimating the pose of a camera. One category of techniques that focuses on extracting parameters to describe a camera is called camera pose estimation or camera calibration. To discuss techniques in this category, we start by introducing the camera model that is most used in these techniques. Most cameras described in these related works are pinhole cameras, which can be described by two matrices. The intrinsic matrix contains camera parameters of focal length, aspect ratio and principal point. The extrinsic matrix contains camera position and orientation parameters. Many different methods have been developed to extract both matrices. However, in order to estimate a camera pose, extracting the extrinsic matrix is sufficient.

2.3.1 Reference from Objects

We start by considering what is perhaps the most intuitive technique, estimating the camera pose with the help of a reference object in an environment. In [35], Tsai provides a method to extract these matrices using a monoview coplanar as the reference object. In [38], Zhang introduces a method to extract these two matrices from a one dimensional labeled stick. Although different reference objects may be used in these techniques, one extracting process for each of them is similar. Commonly, an extracting process starts by taking one or more images that contain reference objects from the environment, then detects and extracts the space information that is signed on the object from the images, and then uses this information to estimate camera pose.

2.3.2 Reference from Images

In [22], Luong et al. introduce another similar technique using a sequence of 2D images that are taken from the environment to estimate the camera pose. From a sequence of images (at least three images), the point correspondence between these images provide information that can be used to estimate a camera pose. Many of these methods that can be used to find point correspondence between images are

surveyed by Szeliski in [34]. Generally, one process of estimating camera pose in this category includes the following steps: feature point detection, feature point matching, and camera pose extraction based on matched features. In contrast with previous methods, no reference object with known space information is needed. Instead, a sequence of 2D images is needed to generate 3D point-cloud models which contain both 3D space information of the environment and 2D image feature points. Then the feature points that are extracted from 2D images are used to match the best correspondence in the point clouds to estimate the camera pose.

In [32], Svarm et al. propose a method to solve the pose estimation problem as a registration problem. They use a fast approximate outlier rejection scheme to handle large datasets with large amounts of outliers. This method assumes knowledge about the orientation of the camera relative to the ground plane as many modern cameras and phones have gravitational sensors. In [18], Li et al. describe a method for camera pose estimation on a world wide scale. The method scales to datasets with hundreds of thousands of images and tens of millions of 3D points through the use of two techniques: a co-occurrence based random sample consensus (RANSAC) [9] and bidirectional matching of image features with 3D points. In [29], Sattler et al. present a framework for determining the pose of a query image relative to a point cloud reconstruction of a large scene consisting of more than one million 3D points. This framework actively searches for additional matches, based on both 2D-to-3D and 3D-to-2D searches. This unified formulation of searching allows the framework to exploit the distinct advantages of both strategies.

One interesting application for these techniques is to estimate camera pose from some old movies for which we have lost the information of the camera parameters. However, as described by Lowe in [20], extracting feature points and handling a large number of outliers is time consuming.

2.3.3 Reference from 3D Building Models

Techniques have also been developed to extract camera pose from a 3D model and images. This category includes methods that are closest to our method as both 3D models and images have been used to estimate camera pose.

In [16], Kohoutek et al. provide a method that uses Normal-Distributions Transform (NDT) and Iterative Closest Point (ICP) to estimate the camera pose in indoor environments. The main idea of their method is to find the pose of a Time-of-Flight (ToF) range camera by matching two point clouds. One is acquired from a ToF range camera, another is generated from a 3D environment polygon model. To do so, they designed a coarse-to-fine matching procedure. First, it generates point clouds from both range images (which are taken from a ToF range camera) and a 3D environment polygon model separately. Then a coarse matching process between two 3D models using the NDT algorithm is applied to two point clouds. After that, a fine matching process using the ICP algorithm is applied. The camera pose is estimated when the two matching processes are completed. One major difference between [16] and our method is in the scale and complexity of the 3D model. In [16], the point cloud that is acquired from a ToF range camera may only represent portions of the point cloud that is generated from the 3D environment polygon model. A larger difference between the number of points in two point clouds jeopardize the capability of the NDT algorithm to converge in its best solution. Because of this, the scale and the complexity of the 3D model is limited. In our method, the 3D model of an indoor environment can be larger in scale and much more complicated.

2.4 Object Pose Estimation

A task somewhat related to that of camera pose estimation in an indoor environment is that of pose estimation for objects. That problem forms a cornerstone for many approaches to object recognition [25]. Moreno-Noguer et al. [24] propose an approach for estimating pose from the correspondence between 2D points and a 3D model using a Gaussian mixture model that is progressively refined as new correspondence are hypothesized. Lim et al. [19] address the problems of pose estimation and detection of objects for which accurate 3D models are available. Their approach relies on feature point matching in RGB images. Zia et al. [39] employ 3D geometric object class representations for recognition and evaluate the ability of their approach to accurately estimate pose. Vajramushti et al. [36] present an object retrieval method based on depth information. They employ their approach for retrieval tasks involving over 3,000 3D objects. Finally, Choi and Christensen [5] use an RGB-D camera and

several scanned views of target objects for pose estimation. All of these approaches differ from ours in the quality criteria that they use for alignment, but also in the nature of the underlying search problem. While camera pose estimation in an indoor environment is naturally continuous, object recognition is an inherently discrete task.

2.5 Depth-Image based Indoor Localization

As proposed by Fillingham in [8], the quality of a pose estimate can be quantified by synthetically generating a depth image of the 3D model with that pose and then computing the sum of squared differences between that depth image and the target depth image.

In his method, a depth image which is taken from a target pose is used as input to find the pose of a viewer. Then, a camera is set inside the 3D model to render a depth image. The rendered image is matched against the input image, then the camera parameters are updated iteratively based on matching results to find the most possible pose of the viewer. By varying each of seven parameters of the camera while keeping all the other dimensions constant then plotting fitness landscape, he found that fitness function value decreases monotonically within a range around the global optimum and the fitness landscape is not rugged and has a smooth basin of attraction near the optimum. Because of this, an evolutionary strategy would be a suitable search method for this task.

The algorithm he used in [8] is a two-tier matching strategy to find the target camera pose. In the upper tier, a pose around the global optimum is supposed to be reached as the starting pose for the lower tier. In the lower tier, the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) algorithm is applied to find the local optimum. His strategy was tested in a 3D model of a classroom which was comprised of 180,518 triangles. Four different target poses were generated for this classroom environment. These target camera poses have been tested in his experiments and resulted in a success rate above 61%. In his method, a seven dimensional vector is used to represent camera pose, an infinite number of vectors may mapping into the same pose which might cause some issues for an evolutionary strategy. In our method, we strive to use the unit quaternion which is a more natural and less redundant way to represent an orientation. The initial height of a camera pose is chosen to be near

eyelevel, which constrains the diversity of initial poses.

2.6 Evolutionary Strategies

The evolutionary strategies (ES) proposed by Rechenberg in [28] and described by Beyer and Schwefel in [4] are optimisation techniques. The idea of these techniques is derived from the principles of biological evolution where recombination, mutation and natural selection happen to a population from generation to generation to gradually improve its fit with the environment. In evolutionary strategies, in the process of each generation, new offspring are generated from their parents with different recombinations and mutations, their fitness is evaluated, and the better offspring are selected to become the parents for the next generation. The better offspring are always preserved in each generation. Within a certain number of iterations, the process gradually improves their fit with the environment. The process stops once it reaches the termination criteria. A question arises consequently: Since mutation is the key to generate better offspring, can we control the mutation process to increase the probability of generating better offspring? This question leads to a need for mutation strength control which is referred to as step size control. We will discuss several topics related to evolutionary strategies in the following section.

2.6.1 Objective Function

In optimization algorithms including evolutionary strategies, an objective function is used to measure the fitness of a given solution and hence formulate the optimization problem. To design an objective function, there are two fundamental requirements: a) the objective function defines the fitness of a candidate solution in a specific problem; b) the objective function be computed quickly. For a specific problem, there could be one or more objective functions that meet these requirements. Intuitively, to select the best objective function (if there is one) from these workable objective functions is crucial in designing a better evolutionary strategy. To do so, we need to evaluate these workable objective functions and select the best one. However, evaluating the performance of these objective functions can be complicated as the performance of an objective function is influenced by many factors during a searching process, such as initialization, initial parameter values, searching environment, and aimed solution.

Even if we can find the best objective function (if there is one), we may notice that the mechanisms being used to measure the fitness of the objective functions are other objective functions, and need other functions to evaluate them. Because of this, many objective functions being used are empirical.

Although it is difficult to find the best objective function (if there is one) in a certain environment, striving for a better evaluating mechanism is possible and worthwhile. In [14], Hillis introduce a co-evolving system that is used to generate minimal sorting networks. Instead of using one objective function, the system uses two objective functions to evaluate the fitness of both the sorting networks and test cases separately. This leads to a co-evolution between the sorting networks and test cases. The benefits of this evaluation mechanism are twofold: first, it helps prevent the system from becoming stuck in local optima; second, the testing becomes more efficient [14]. In this thesis, we did experiments with different objective functions and found these different objective functions influence the searching success rate significantly.

2.6.2 Step size Control

Step size control is key to the design of evolution strategies. The step size controls the mutation strength within each generation. A smaller step size increases the likelihood of generating improvements in each generation (and a larger step size leads to the opposite scenario). In situations where larger step sizes lead to larger expected improvements, a step size control mechanism should focus on increasing the step size (and decreasing it in the opposite situation). The main goal of the step size control is to keep the step sizes close to their optimal values. These optimal step size values can significantly change over time or depending on the position in search space. In the following, the 1/5th success rule, self-adaptation method, and cumulative step size adaptation method (CSA) are introduced.

The 1/5th Success Rule

The 1/5th success rule for step size control is discovered very early in the research of evolution strategies by Rechenberg in [28]. This method has been mostly superseded by more sophisticated methods. However, the conceptual insight in it is still valuable.

As described by Hansen et al. in [12], in the 1/5th success rule, the step size increases if the probability of making improvement in generating offspring is larger than 20%; and it decreases if the probability of making improvement in generating offspring is smaller than 20%. Schumer et al. also found a similar rule independently before in [30].

Self-Adaptation

In self-adaptation, new control parameter settings are generated by recombination and mutation. Hansen et. al. exemplified the concept in [13]. The mutation step from generation g to $g+1$ reads for each offspring $k = 1, ..., \lambda$

$$\sigma_k = \sigma^{(g)} \exp(\xi_k)$$

where $\sigma \in \mathbb{R}^+$ denotes step size, $\xi_k \in \mathbb{R}$, for $k = 1, ..., \lambda$ independent realizations of a random number with zero mean. Typically, ξ_k is normally distributed with standard deviation $1/\sqrt{2n}$. After λ mutation steps are generated, the best offspring are selected and the σ is updated as,

$$\sigma^{(g+1)} = \frac{1}{\mu} \sum_{\sigma_k \in \mathcal{P}} \sigma_k$$

where $\mathcal{P}$ is the population of selected offspring, $\mu \in \mathbb{N}$ denotes the number of parents.

However, as described by Hansen in [13], the mutation and recombination on σ introduces a moderate bias such that σ tends to increase under random selection.

$(\mu/\mu, \lambda)$-ES and Cumulative Step Size Adaptation (CSA)

Our algorithm in this thesis is a $(\mu/\mu, \lambda)$-ES with search path. Figure 2.1 gives pseudocode for a $(\mu/\mu, \lambda)$-ES with search path that is based on an algorithm described by Hansen et al. in [12]. Our only difference is in Line 9, where we use the simplified update of the mutation strength mentioned in [13] that is based on the squared length of the overall path. $\mu \in \mathbb{N}$ denotes the number of parents and $\lambda \in \mathbb{N}$ ($\lambda > \mu$) the number of offspring generated in each iteration (Line 1). A single parental centroid $\mathbf{x} \in \mathbb{R}^n$ is initialized as a solution vector, $\sigma \in \mathbb{R}^+$ is step size. (Line 2). Mutation happens to this parental centroid and offspring are generated (Line 6), the

new offspring are selected to form the population $\mathcal{P}$ (Line 7). Recombination happens at the end of the loop and the new centroid is generated (Line 10).

As mentioned in [12], algorithms with a single parental centroid are simpler to formalize, easier to analyze and even perform better in many circumstances as they allow for maximum genetic repair such as parameters recombination.

1. **given** $n \in \mathbb{N}, \mu \in \mathbb{N}, \lambda \in \mathbb{N}, c \in (0, 1], D \in \mathbb{R}^+$
2. **initialize** $\mathbf{x} \in \mathbb{R}^n, \sigma \in \mathbb{R}^+, \mathbf{s} = \mathbf{0}$
3. **while** not meet terminating criteria
4. **for** $k \in \{1, ..., \lambda\}$
5. $\mathbf{z}_k = \mathcal{N}(\mathbf{0}, \mathbf{I})$
6. $\mathbf{x}_k = \mathbf{x} + \sigma \mathbf{z}_k$
7. $\mathcal{P} \leftarrow \mathsf{select\_}\mu\mathsf{\_best}(\{(\mathbf{x}_k, \mathbf{z}_k, f(\mathbf{x}_k))\})$
8. $\mathbf{s} \leftarrow (1 - c)\mathbf{s} + \sqrt{c(2 - c)} \frac{\sqrt{\mu}}{\mu} \sum_{k \in \mathcal{P}} \mathbf{z}_k$
9. $\sigma \leftarrow \sigma \exp\left(\frac{\|\mathbf{s}\|^2 - n}{2Dn} \right)$
10. $\mathbf{x} = \frac{1}{\mu} \sum_{k \in \mathcal{P}} \mathbf{x}_k$

Figure 2.1: Pseudocode of a $(\mu/\mu, \lambda)$-ES with Search Path

Our algorithm uses a CSA method to control the step size, which is also described in Figure 2.1. In self-adaptation, step sizes are connected with individuals and selected based on the fitness of each individual. However, step sizes that serve individuals well may not maximize the progress of the entire population. In a CSA method, a search path carries information of the interrelation between individual steps. With this information, we can improve the step size adaptation and search procedure. The search path is updated by cumulating the actual evolution path in a weighted manner (Line 8), where $\mathbf{s}$ is the search path, $c \in (0, 1]$ is the cumulation parameter, $\mathbf{z}_k$ is the

single (local) mutation step in each offspring (Line 5). The factors $\sqrt{c(2-c)}$ and $\sqrt{\mu}$ in Line 8 guaranty unbiasedness of $\mathbf{s}$ under neutral selection [12]. Then the step size update is accomplished by comparing the length of search path $\mathbf{s}$ with the expected length n (Line 9), where $n \in \mathbb{N}$ is the search space dimension, $D \in \mathbb{R}^+$ is a damping parameter. The length of the search path $\mathbf{s}$ determines the global step-size change in Line 9. The benefits of using a search path are: a) it implements a low-pass filter for selected mutation vectors ($\mathbf{z}$-steps), high frequency information (most likely noisy) is removed. b) it records information of the direction of $\mathbf{z}$-steps, because of this even if the steps have the same length, the length of $\mathbf{s}$ can be different.

2.6.3 Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)

As a CMA-ES is applied by Fillingham in [8], a brief introduction of this evolutionary strategy follows. Hansen described a CMA-ES in [11], the offspring in CMA-ES are sampled from the a multivariate normal distribution

$$\mathcal{N}\big(\mathbf{m}, \sigma^2 C\big)$$

(where $\mathbf{m}$ is the mean value of the search distribution, σ is the global step length and C is the covariance matrix of the search distribution), which is updated each generation. The mean of the distribution is updated each generation based on the best μ offspring from the previous generation, according to

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{(g)}$$

where w_i is the weight given to the i-th ranked candidate (must sum to 1, a weight of $1/\mu$ for each candidate results in taking the arithmetic mean) and $\mathbf{x}_{i:\lambda}^{(g)}$ is the i-th ranked candidate. By adapting the covariance matrix in such a way that successful directions are favoured, the distribution becomes elongated towards more favourable solutions. This is to hopefully decrease the number of generations needed by providing faster convergence. The covariance matrix is updated according to

$$C_\mu^{(g+1)} = (1 - c_\mu) C_\mu^{(g)} + c_\mu \sum_{i=1}^{\mu} w_i \left(\frac{\mathbf{x}_{i:\lambda}^{(g)} - m^{(g)}}{\sigma^{(g)}} \right) \left(\frac{\mathbf{x}_{i:\lambda}^{(g)} - m^{(g)}}{\sigma^{(g)}} \right)^T$$

where c_μ is the learning rate for updating the covariance matrix (which is set to adapt how quickly previous generations fall out of influence), and σ is the global step length. The global step length is adapted to make the search to converge faster.

Chapter 3

Algorithm

"A curious aspect of the theory of evolution is that everybody thinks (s)he understands it."
— J. Monod

In this chapter, we propose an evolutionary algorithm that uses a target depth image as input and a 3D model of an environment as the reference to recover the camera pose of the target depth image by registering it against the 3D model.

Our algorithm consists of: the representation of a camera, the objective function, the coordinate systems and the pose optimization method. We will discuss each of them in the following sections.

3.1 Representation of Camera

As this thesis focuses on designing an optical based indoor localization method, selecting a suitable camera model is crucial. Different types of cameras have been developed to meet different requirements (e.g. zenith camera, omnidirectional camera, pinhole camera) [37]. All the cameras described in this thesis are pinhole cameras. A pinhole camera model can be described with two matrices. The intrinsic matrix contains camera parameters of focal length, aspect ratio, and principal point. The extrinsic matrix contains camera position and orientation parameters. In this thesis, we are focusing on extracting camera pose, which includes camera location and orientation information, and hence assuming that the intrinsic matrix is known.

The camera position and orientation information have six degrees of freedom (6-DoF). To represent a camera position in space is straightforward, a three dimensional vector (x, y, z) meets the requirement perfectly. To represent a camera orientation is more complicated, several different expressions are available to represent an orientation, including fixed angle representation, Euler angle representation, and quaternion

representation. Both fixed angle and Euler angle representation use a vector with three dimensions to describe an orientation while quaternion representation needs a vector with four dimensions to do so. However, the first two representations meet the Gimbal lock problem when two axes of rotation are in a parallel configuration which causes a loss of one degree of freedom. In this thesis, we use the unit quaternion to represent the orientation of a camera.

The location information can be directly represented with a three dimensional vector. However, since the unit quaternion is used to represent and update the orientation of the camera, we need to convert the value from the unit quaternion to a 3 by 3 rotation matrix that used to introduce camera orientation, as described by Shoemake in [31] is converted from a unit quaternion (x, y, z, w) as follows,

$$m = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$$

3.2 Objective Function

In this thesis, we consider the problem of registering the target depth image against the 3D model of the environment as an optimization problem in the space of candidate poses. A candidate pose is used to generate a depth image from the 3D model. That depth image is then evaluated against the target depth image, and the resulting score is associated with the candidate pose.

In [33], Szeliski distinguishes between direct (pixel based) and feature-based alignment approaches. Feature based approaches require feature detection, feature selection and establishing correspondence. In this thesis, we employ a direct approach that quantifies the fit of an alignment simply by summing squared differences in depth values between the two images. Compared with feature based approaches, an advantage of a pixel based approach is that it has a less computational cost [20]. The sum of squared differences equation is

$$\mathbf{S} = \sum_{k=1}^{n} \Big(I_1(k) - I_2(k) \Big)^2$$

where n is the number of pixels in the image, I_1 and I_2 are the two depth images, $\mathbf{S}$ is the sum of squared differences value between two depth images. A larger score

indicates a worse candidate. As mentioned in Chapter 2, it is challenging to find the best objective function (if there is one). The further experiment introduced in Chapter 4 indicates weakness of this objective function in some specific situations.

3.3 Coordinate Systems

A further important consideration is the choice of a coordinate system to compute differences between depth values. The depth buffer in OpenGL stores the depth value in window coordinates. In order to make the best use of the numerical accuracy provided by the depth buffer, the projective transformation progressively compresses depth values with increasing distance from the near clipping plane. Differences in depth values in greater distance from the camera are thus deemphasized, resulting in a focus of the registration algorithm on nearby objects.

Figure 3.1 illustrates how different coordinate systems influence the scoring result. The top left is a target depth image, top right is a candidate depth image. Bottom images are obtained by computing pixel-wise squared differences. Bottom left is in window coordinates and bottom right is in camera coordinates. With window coordinates, the difference is more emphasised on pixels that are close to the camera (e.g. the chairs and the desks). With camera coordinates, the difference is more emphasised on pixels that far from camera (e.g. the rear wall and the desk). We have found in preliminary experiments that transforming depth value into camera coordinates results in a significantly improved ability of our algorithm to locate the globally optimal solution to the optimization problem. Presumably, different coordinate systems may formulate different local optima in the search space.

In this thesis, we convert the depth value from window coordinates to camera coordinates by performing the following transformation,

$$v' = \frac{2 \times z_{\text{far}} \times z_{\text{near}}}{\big(z_{\text{far}} + z_{\text{near}}(z_{\text{far}} - z_{\text{near}}) \times (1 - 2 \times v)\big)}$$

where v denotes the depth value in window coordinates and v' denotes the depth value in camera coordinates, z_{near} and z_{far} are near and far clipping plane separately.

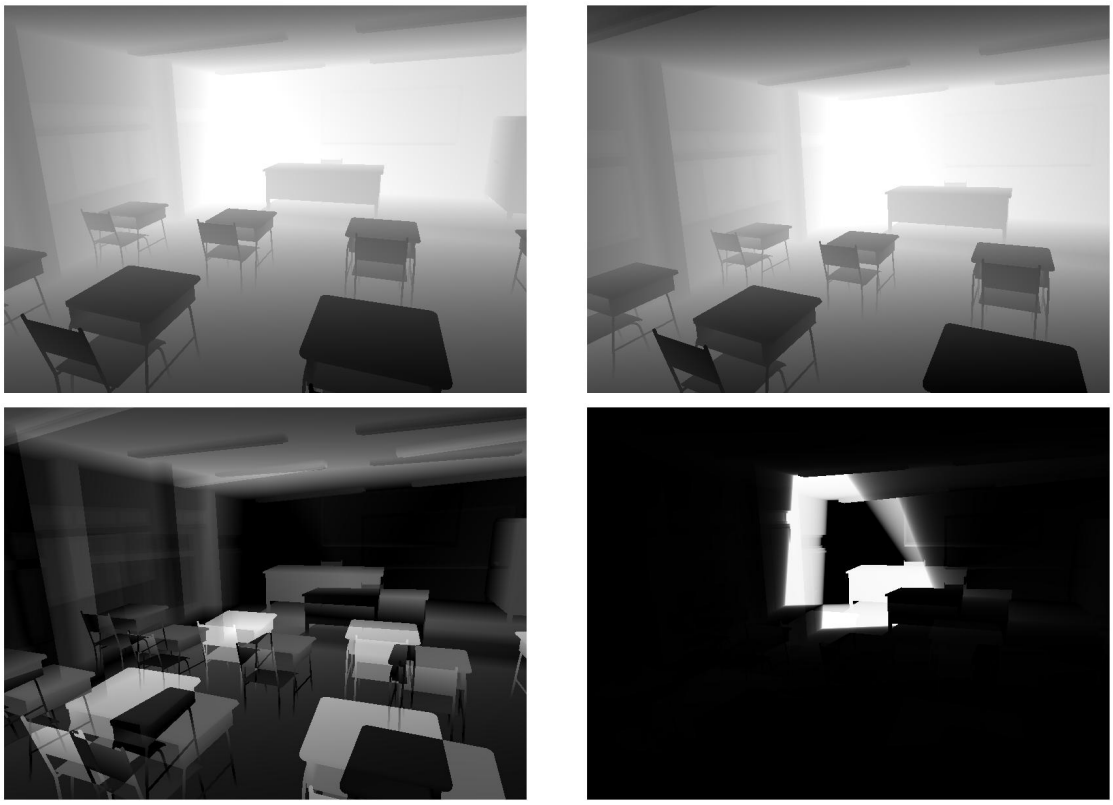

Figure 3.1: A target depth image (top left), a candidate depth image (top right; both with intensities corresponding to depth values in window coordinates) and the images obtained by computing pixel-wise squared differences in window coordinates (bottom left) and in camera coordinates (bottom right; both with intensities normalized).

3.4 Pose Optimization

With the objective of minimizing the distance between the target depth image and a depth image generated using the candidate pose thus defined, the problem of estimating camera pose is an optimization problem over the search space $T_{\mathbf{q}}S^3$ where S is the unit sphere embedded in $\mathbb{R}^4$. A challenge in developing an optimization algorithm is that the search space is not Euclidean but instead a smooth manifold embedded in $\mathbb{R}^7$. When developing an evolutionary algorithm for the problem, care must be taken to design the variation operators in such a way that offspring candidate solutions remain on the manifold without introducing any sort of bias in the search. The algorithm for optimization on Riemannian manifolds proposed by Colutto et al. [6] is immediately applicable. That algorithm is a covariance matrix adaptation evolution strategy (CMA-ES) [13] and as such adapts the full covariance matrix governing its offspring distribution. Mutation vectors are generated in the tangent space at the current population centroid and are mapped onto the search manifold using the Riemannian exponential map. The algorithm uses parallel transport to transform search paths as well the covariance matrix from one iteration to the next.

In the absence of evidence of ill-conditioning of the problem under consideration, we strive for a simpler approach that does not attempt to adapt the full covariance matrix of its mutation distribution. However, as the parametrization of the 3D model is arbitrary, a scale factor that governs the size of steps made in the subspace of locations relative to those in the subspace of orientations needs to be adapted in order to achieve scale invariant performance. The distribution of mutations in our algorithm is thus controlled by two parameters: an overall step size and a scale factor that is used to scale the step size in the subspace of orientations relative to that in the subspace of locations. As Colutto et al. [6] we use cumulative step size adaptation for the control of the overall step size, but we employ self-adaptation for the control of the scale factor. We have conducted experiments with separate paths in the two subspaces but found the self-adaptive approach to be more successful.

Common terminology used in this section is in connection with evolution strategies [12], our algorithm is a $(\mu/\mu, \lambda)$-ES that at each iteration selects μ out of the $\lambda > \mu$ offspring generated. One generation of offspring conduct both recombination and mutation. For each iteration, the algorithm starts with a centroid pose $\langle \mathbf{x}, \mathbf{q} \rangle \in$

$\mathbb{R}^3 \times S^3$ from previous iteration or initialized one if this is first iteration, step size $\sigma \in \mathbb{R}^+$, search path $\langle \mathbf{s}_{\text{loc}}, \mathbf{s}_{\text{rot}} \rangle \in \mathbb{R}^3 \times T_{\mathbf{q}}S^3$, where $T_{\mathbf{q}}S^3$ denotes the tangent space of the unit sphere in $\mathbb{R}^4$ at the location of unit quaternion $\mathbf{q}$, and step scale factor $\alpha \in \mathbb{R}^+$. A single iteration of the algorithm is described in Figure 3.2

Step 1) uses the step size σ along with the step scale factor α to compute the sizes of the mutation steps in the subspaces of location and orientation for each of the offspring to be generated. As we employ self-adaptation, those step sizes undergo multiplication with a lognormally distributed factor. Constant $\tau \in \mathbb{R}^+$, which controls the size of the mutations, is set to 0.3. While the exact value is uncritical, smaller values preclude fast adaptation while larger ones render the process unstable.

In Step 2) the pose vectors characterizing offspring candidate solutions are generated by applying mutation to the pose centroid. Mutations in the orientation subspace are implemented by generating steps in the tangent space at the location of the quaternion associated with the current pose centroid. Those vectors can be generated by sampling standard normally distributed vectors $\mathbf{w} \in \mathbb{R}^4$ and then projecting them onto the tangent space according to

$$\mathbf{z}_{\text{rot}} = \mathbf{w} - (\mathbf{q} \cdot \mathbf{w})\mathbf{q}$$

where $\cdot$ denotes the inner product. The Riemannian exponential map, which for the case of S^3 can be computed as

$$\exp_{\mathbf{q}}(\sigma\mathbf{z}) = \mathbf{q}\cos(\sigma\|\mathbf{z}\|) + \mathbf{z}\frac{\sin(\sigma\|\mathbf{z}\|)}{\|\mathbf{z}\|}$$

is used to map the mutations onto the unit quaternion manifold.

Step 3) evaluates the candidate poses thus generated by using OpenGL to generate the corresponding depth images and computes fitness values by summing squared differences to the target across the images. The offspring are then ranked according to their fitness.

The update of the search path in Step 4) is as proposed by Ostermeier et al. [27]. The orientation component of the search path is in the tangent space of the unit quaternion manifold at $\mathbf{q}$. Cumulation parameter $c \in (0, 1]$ is uncritical and set to 0.25.

Step 5) describes the update of the pose centroid through recombination of the μ best of the λ candidate poses. Recombination of the orientation component is

Input: pose centroid $\langle \mathbf{x}, \mathbf{q} \rangle \in \mathbb{R}^3 \times S^3$
step size $\sigma \in \mathbb{R}^+$
search path $\langle \mathbf{s}_{\text{loc}}, \mathbf{s}_{\text{rot}} \rangle \in \mathbb{R}^3 \times T_{\mathbf{q}} S^3$
step scale factor $\alpha \in \mathbb{R}^+$

1) For $i = 1, ..., \lambda$ generate offspring scale factors

$$\beta^{(i)} = \alpha \exp(\tau z^{(i)})$$

where the $z^{(i)}$ are standard normally distributed, and let $\sigma_{\text{loc}}^{(i)} = \sigma \sqrt{\beta^{(i)}}$ and $\sigma_{\text{rot}}^{(i)} = \sigma / \sqrt{\beta^{(i)}}$.

2) For $i = 1, ..., \lambda$ generate offspring candidate solutions $\langle \mathbf{y}^{(i)}, \mathbf{r}^{(i)} \rangle \in \mathbb{R}^3 \times S^3$ with

$$\mathbf{y}^{(i)} = \mathbf{x} + \sigma_{\text{loc}}^{(i)} \mathbf{z}_{\text{loc}}^{(i)}$$
$$\mathbf{r}^{(i)} = \exp_{\mathbf{q}} \left(\sigma_{\text{rot}}^{(i)} \mathbf{z}_{\text{rot}}^{(i)} \right)$$

where mutation vectors $\mathbf{z}_{\text{loc}}^{(i)}$ are standard normally distributed in $\mathbb{R}^3$, mutation vectors $\mathbf{z}_{\text{rot}}^{(i)}$ are standard normally distributed in $T_{\mathbf{q}} S^3$, and $\exp_{\text{q}}(\cdot)$ denotes the Riemannian exponential map.

3) For $i = 1, ..., \lambda$ compute $f(\langle \mathbf{y}^{(i)}, \mathbf{r}^{(i)} \rangle)$ as the sum of squared differences between the target depth image and the depth image generated with camera parameters $\langle \mathbf{y}^{(i)}, \mathbf{r}^{(i)} \rangle$. Let $(k; \lambda)$ denote the index of the offspring candidate solution with the kth smallest objective function value.

4) Update the search path according to

$$\mathbf{s}_{\text{loc}} \leftarrow (1 - c)\mathbf{s}_{\text{loc}} + \sqrt{c(2 - c)/\mu} \sum_{k=1}^{\mu} \mathbf{z}_{\text{loc}}^{(k;\lambda)}$$
$$\mathbf{s}_{\text{rot}} \leftarrow (1 - c)\mathbf{s}_{\text{rot}} + \sqrt{c(2 - c)/\mu} \sum_{k=1}^{\mu} \mathbf{z}_{\text{rot}}^{(k;\lambda)}$$

5) Update the pose centroid according to

$$\mathbf{x} \leftarrow \frac{1}{\mu} \sum_{k=1}^{\mu} \mathbf{y}^{(k;\lambda)}$$
$$\mathbf{q} \leftarrow \exp_{\mathbf{q}} \left(\frac{1}{\mu} \sum_{k=1}^{\mu} \sigma_{\text{rot}}^{(k;\lambda)} \mathrm{z}_{\text{rot}}^{(k;\lambda)} \right)$$

6) Use parallel transport to transform $\mathbf{s}_{\text{rot}}$ from the location of the old pose centroid to the new one.

7) Update the step size and scale factor according to

$$\sigma \leftarrow \sigma \exp\left(\frac{\|\mathbf{s}_{\text{loc}}\|^2 + \|\mathbf{s}_{\text{rot}}\|^2 - 6}{12D}\right)$$

$$\alpha \leftarrow \left(\prod_{k=1}^{\mu} \beta^{(k;\lambda)}\right)^{1/\mu}$$

8) If $2\sigma > \sqrt{\alpha}$, then let $\alpha \leftarrow 2\sigma\sqrt{\alpha}$ and subsequently $\sigma \leftarrow \sqrt{\alpha}/2$.

Figure 3.2: Single step of the evolutionary algorithm for pose estimation

performed in tangent space and the Riemannian exponential map is used again in order to map onto the unit quaternion manifold.

The use of parallel transport in Step 6) is as proposed by Colutto et al. [6] in order to ensure that the rotational component of the search path remains within the tangent space at the current unit quaternion, which has shifted in Step 5). Huckemann et al. [15] provide a straightforward way of computing the parallel transport.

Step 7) updates the step size according to the prescription of cumulative step size adaptation and the step scale factor by computing the geometric mean of the step scale factors of the successful offspring, thus implementing self-adaptation. The damping factor $D \in \mathbb{R}^+$ is rather uncritical and set to 4.0.

Finally, Step 8) implements a cap on the size of steps in the orientation subspace. Arnold [1] shows that if no such cap is imposed, cumulative step size adaptation on spherical manifolds may result in metastable states associated with large steps, leading to many iterations without useful progress. The cap implemented in this step ensures that σ_{rot} does not grow too large while not affecting the step size σ_{loc} used in the subspace of camera locations.

We use $\mu = 3$ and $\lambda = 10$ in all of our experiments. Further initialization condition and termination criteria are discussed in the following Chapter.

Chapter 4

Evaluation

"Sentence first, verdict afterwards."
— L. Carroll, Chapter XII, Alice's Adventures in Wonderland

To see whether the algorithm performs as it is designed, we conducted two kinds of experiment. First, we tested the algorithm in a unimodal environment to see whether it works in a simple search space or not. Then we tested the algorithm in two synthetic 3D environments, a classroom environment and a church environment. The following sections discuss the details of these experiments.

4.1 Unimodal Environment

In a unimodal environment, only one global optimum exists in the search space which means there is no other possible optimum for the algorithm to reach. By performing experiments in a unimodal environment, we can see whether the algorithm is solid enough to reach the global optimum consistently when there is no distraction from local optima. By tracing the value of variables introduced in the algorithm during the experiment, we can have a better understanding of the algorithm. There are two major concerns in the unimodal experiment. First, can the algorithm reach the global optima with different initializations? Second, does the self-adaptation of step size between location space and orientation space work as it is designed to? We will discuss the objective function that was used in our unimodal environment and the experiment results in the following sections.

4.1.1 Objective Function

The unimodal environment does not contain 3D models of an indoor environment. It is unnecessary to render depth image nor to transform between different coordinate systems. The objective function used in a unimodal experiment is different from

but simpler than the function we introduced previously. The scoring function in this environment measures the sum of position distance and rotation distance from the current pose against the target one to evaluate the similarity between these two poses. The objective function being used in the unimodal environment to calculate the scoring value is

$$S = \|\mathbf{x} - \mathbf{x}_{\text{target}}\| + 2 \arccos |\mathbf{q} \cdot \mathbf{q}_{\text{target}}| \cdot s$$

where $\mathbf{x}$ is a three dimensional location vector describing current pose location, $\mathbf{x}_{\text{target}} = (0, 0, 0)$ is the target location vector, $\mathbf{q}$ is a four dimensional unit quaternion vector describing current pose orientation, $\mathbf{q}_{\text{target}} = (0, 1, 0, 0)$ is the target unit quaternion vector, s is a scalar to orientation distance value; $\mathbf{q} \cdot \mathbf{q}_{\text{target}}$ is the dot product of two quaternions. S is the score evaluating the distance between the current pose and the target one; a smaller scoring value S indicates two poses are closer. Ideally, the value of this objective function should decrease to as close to 0 as numeric issue permits as the search progresses.

4.1.2 Experiment result

In this thesis, a self-adaptation mechanism is designed to build a connection between the location step size and the orientation step size in order to keep the scale difference between location and orientation vectors in the offspring in a range that improves the probability of generating better solutions. To test this mechanism and the algorithm, we applied experiments in two categories: one has the orientation scale factor value as 1 and the other has it as $1,000$. In each of these two categories, we initialized position by uniformly and randomly distributing them within a distance in three types which are 1 unit, 100 units and $10,000$ units. The initial orientations are always generated by uniformly and randomly sampling unit quaternions. The initial step size is set to $\sigma = 1$ and the initial step scale factor to $\alpha = 1$. The search path is set to 0 initially. All the experiments are terminated when the scoring value is smaller than 1e$-$6. We label runs in which such solutions are found successful. Runs are terminated as unsuccessful if after $1,000$ iterations no solution satisfying the criteria for successful termination has been found. For each of these different initialization categories, we conducted 100 individual runs. The following sections discuss the experiments within these two categories.

Orientation scale factor set to 1

Table 4.1 summarizes the experimental results when the orientation scale factor is set to 1. With this orientation scale factor, the tests of all three different initializations for location reached a 100% success rate during experiments. As the initial location range decreases from 10,000 units to 1 unit, the number of iterations that it takes to reach the target pose also decreases from 372.5 to 192.8. It is safe to say, a closer starting pose takes fewer iterations to reach the target pose than one with a larger starting distance. Standard deviation also increases as the starting pose is far from the target one.

	success rate	median	mean	standard deviation
1 unit	100%	194	192.8	10.3
100 units	100%	252	252.1	16.3
10,000 units	100%	326	327.5	26.8

Table 4.1: Runs in the unimodal environment with orientation scale factor set to 1

To have a comprehensive understanding of our algorithm, we tracked the value of the objective function score, step size σ and step scale factor α for each individual run. Then we analyzed and visualized the behaviour of these parameters by plotting the median run of each set of 100 experiments based on the number of iteration each run takes.

The scoring value directly indicates the fitness of a result that is generated in each iteration along one searching process. The first row of Figure 4.1 shows the median runs of each type of location initialization and illustrates how the scoring value changes within three different types of location initializations. Different initializations lead to different starting scores due to the design of the objective function. All these median runs can converge to the global optimum almost linearly. A closer starting pose has a smaller score to begin with and often leads to a shorter searching process. A more distant starting pose has a larger score to begin with and costs more iterations to reach the target pose.

As the step size σ controls the mutation strength in each iteration, a larger σ makes offspring more different from their parents. A smaller σ makes offspring more similar to their parents. The middle row of Figure 4.1 illustrates the median run of

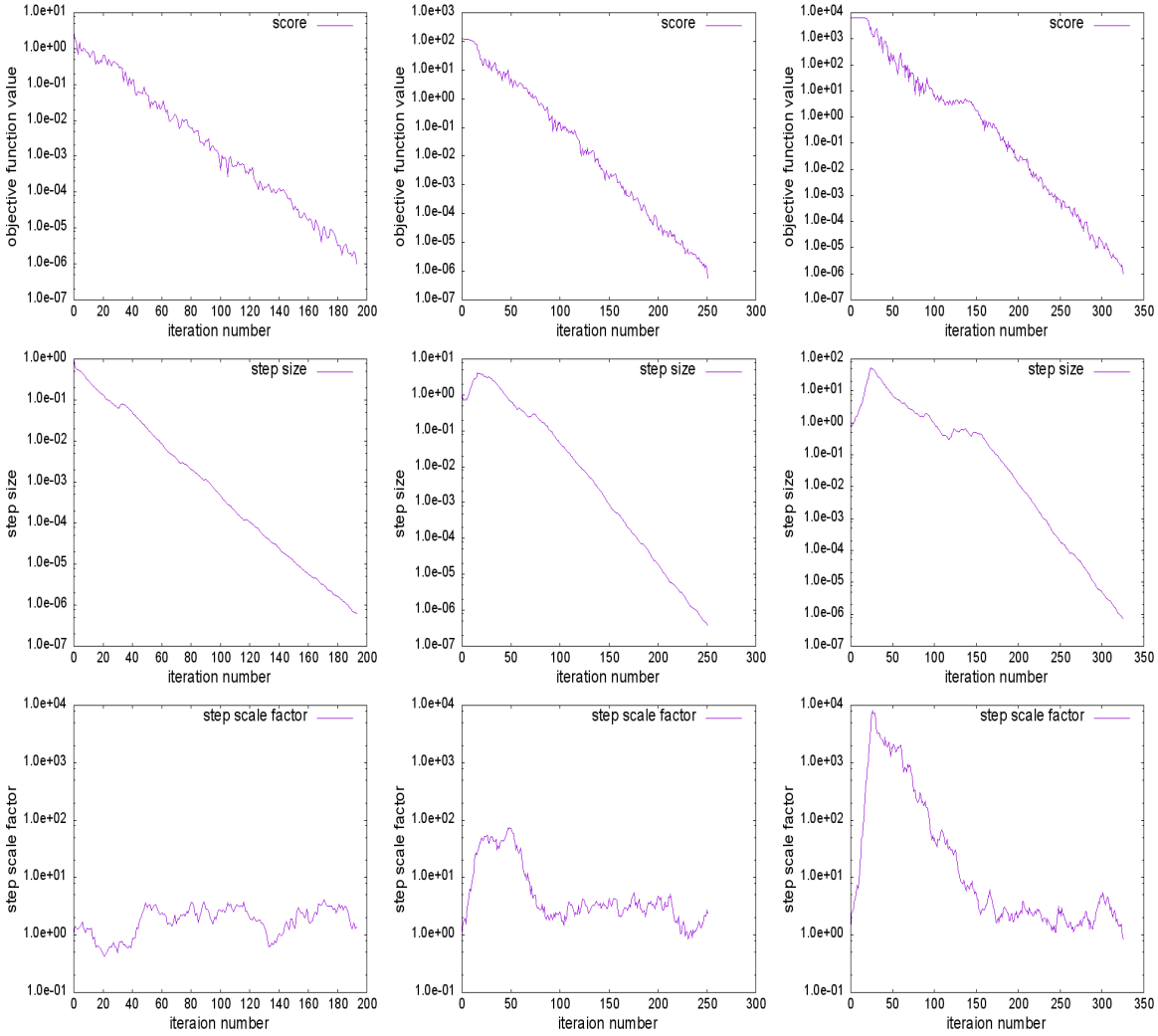

Figure 4.1: Tracing the objective function value (top row), step size (middle row) and step scale factor (bottom row) during median search processes. Orientation scale factor is set to 1. Initial location within different range scale: 1 unit (left column), 100 units (middle column), 10,000 units (right column)

each type of location initialization and shows how the value of step size σ changes during the searching processes. As the searching processes during these three median runs, the step sizes all decrease close to zero. However, there are still some differences: with 1 unit initial location (left), the step size decreases almost constantly; with 100 and 10,000 units initial location (middle and right), the step sizes increase first before decreasing. From certain iterations onward the σ starts decreasing continuously. The sooner the searching process and σ reach that point, the faster the algorithm can finish its searching process. It can be seen from middle row of Figure 4.1 that

different initial locations also influence the maximum value of σ during a searching process. With an initial range for location as 1 unit (left), the starting value of step size is the maximum value during the entire searching process. With a larger initial range for location such as 100 and 10,000 units, the step size increases to 1e1 and 1e2 before it starts decreasing. In some cases the step sizes may have other increases or stagnations (middle and right) during a searching process.

Another variable we traced during the experiment is step scale factor α. In the algorithm, the step scale factor α is designed to automatically connect the position step size and rotation step size. Since the scale of location search space can be significantly larger than that of orientation, the step scale factor α is designed to balance this difference in the scale of two search spaces. As described in Section 3.4, a larger α decreases the step size in orientation space and increases the step size in location space. A smaller α increases the step size in orientation space and decreases the step size in location space.

The bottom row of Figure 4.1 illustrates the median run of each type of location intialization and shows how the value of step scale factor α changes during searching processes. With an initialization distance around 1 unit (left), the behaviour of the step scale factor fluctuates in a range between 0.5 to 10. Because the initial location is close to the target pose, the optimum mutation strengthes for location space and orientation space are close.

A better understanding of step scale factor α comes from a larger initialization of location such as 100 or 10,000 units. In these two initializations, the location search space is far larger than the orientation search space. Therefore, using the same step size for both location and orientation search space is highly biased toward the location search space and hence inefficient. We want to control the step size σ in both location and orientation space automatically by introducing the step scale factor α. We can see from the bottom row of Figure 4.1 that the step scale factor α increases dramatically within the first 50 iterations before it starts decreasing. During these iterations, a reasonable compromise is made by increasing the step scale factor α to get a larger step size for location search space and meanwhile keep a rather small step size for orientation space. Comparing step size and step scale factor in Figure 4.1, we can see an increasing step size σ is desirable for the search space of location. And at the same

time, a large step scale factor α keeps a relatively small step size σ in the orientation search space.

Comparing the scores in the top row of Figure 4.1, we notice that the stagnations appear at the same period when the σ increases for the second time (right). One possible explanation for this is that the step size σ becomes too small to make any significant progress during this period; by increasing the step size σ, the algorithm attains the ability to make significant progress again. Fortunately, after stagnations, the algorithm can converge to the target pose linearly.

Orientation scale factor set to $1,000$

We also did the experiments with the orientation scale factor set to $1,000$. Table 4.2 summarizes the experimental results when the orientation scale factor is set to 1,000. With this orientation scale factor, the tests of all three different initializations for location reached 100% success rate during experiments. As the initial location range decreases from 10,000 units to 1 unit, the number of iterations that it costs to reach the target pose also decreases from 392.5 to 268.5. Compared with Table 4.1, the algorithm with the orientation scale factor set as $1,000$ needs more iterations to reach the target pose. In other words, different objective functions influence the efficiency of a searching process in a unimodal environment.

	success rate	median	mean	standard deviation
1 unit	100%	264	268.5	25.4
100 units	100%	311	313.9	28.7
10,000 units	100%	390	392.5	25.1

Table 4.2: Runs in the unimodal environment with orientation scale factor set to 1,000

Similar to the previous experiment, we tracked the value of the objective function score, step size σ and step scale factor α for each individual run. Then we analyzed and visualized the behaviour of these parameters by plotting the median run of each set of 100 experiments based on the number of iteration each run takes.

The top row of Figure 4.2 shows the score of median success runs for each type of location initialization. Different initializations lead to different starting scores. A closer starting pose costs less iterations to reach the target pose than a more distant

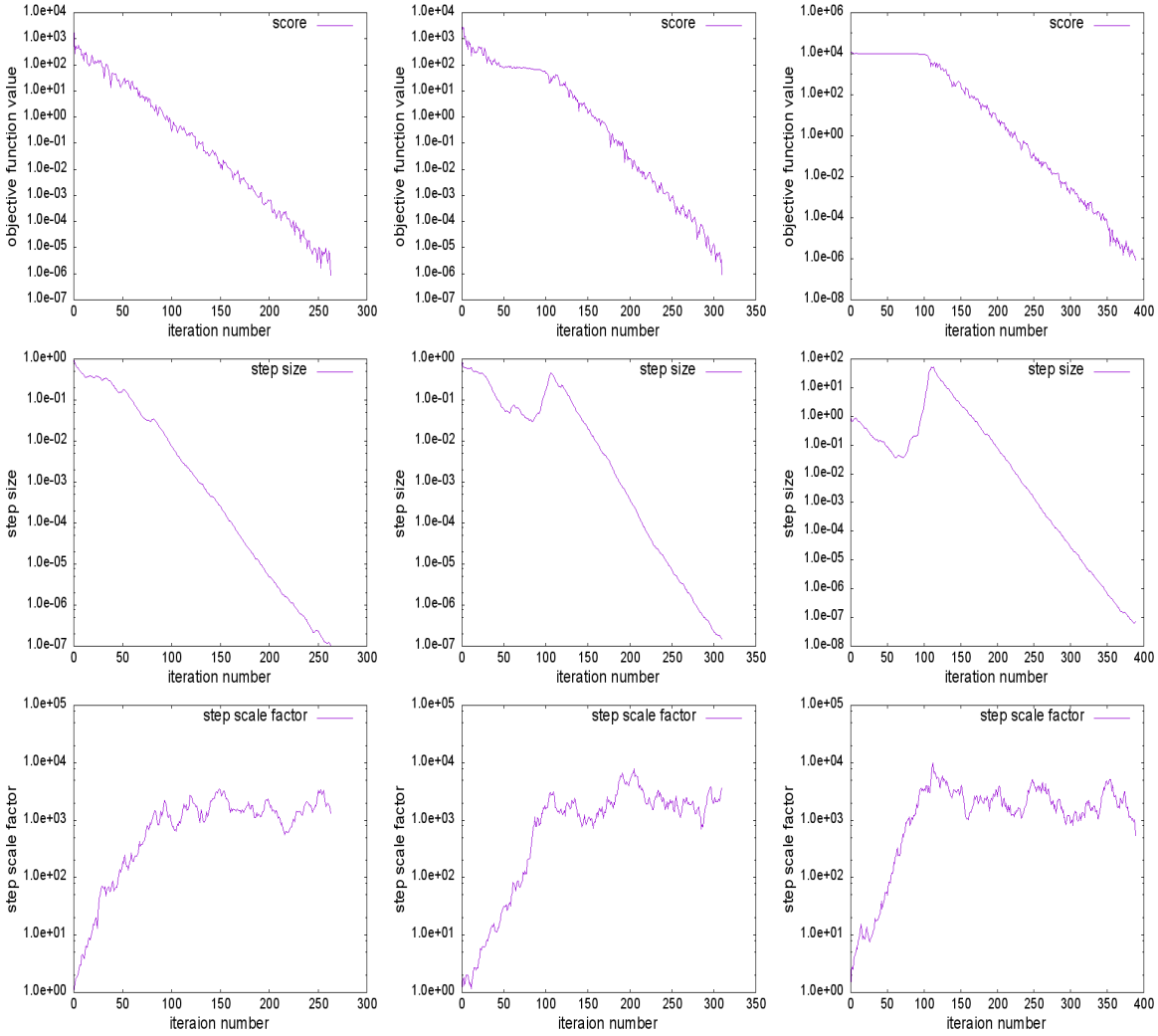

Figure 4.2: Tracing the scoring value (top row), step size (middle row), step scale factor (bottom row) during median search processes. Orientation scale factor is set to 1,000. Initial location within different range scale: 1 unit (left), 100 units (middle), 10,000 units (right)

one. Compared with the top row in Figure 4.1, the stagnations in 100 units (middle) and $10,000$ units (right) are more significant. The stagnation happens in the median run where the initial location range is set to $10,000$; there is no log-linear convergence in the first 100 iterations.

The middle row of Figure 4.2 shows the step size of median success runs for each type of location initialization. Since all the step sizes are set to 1 initially, the change of the step size during the searching process reveals the behaviour of the searching process. For the initial location range set to 1 unit (left), the behaviour of step size is

similar to that in the middle row of Figure 4.2; the step size starts decreasing once the searching process begins. For the initial location as 100 (middle) or 10,000 (right), the behaviour of these step sizes is different from the middle row of Figure 4.2. In the middle row of Figure 4.2, these two step sizes decrease in the first hundred iterations and then increase to their maximum before the linear decrease. One major difference between the behaviour of these two step sizes is their maximum values: the maximum step size in 100 units location initialization (middle) is less than 1, the maximum step size in 10,000 units location initialization (right) is larger than 50. In the middle row of Figure 4.2, these two step sizes increase first then start decreasing linearly.

Figure 4.3 shows the location and orientation distances during a searching process in these median runs. When the orientation scale factor is set as 1 unit (top row), the location distance in all three median runs decreases directly, but in some cases (middle and right) the orientation distance stays in stagnation. This means the algorithm makes some improvement on reaching the target location but not on the target orientation, in the first one hundred iterations. On the contrary, when the orientation scale factor is set to 1,000 units (bottom row), the orientation distance in all three median runs decreases directly, but the location distance stays in stagnation (middle and right) or even increases (left) in the first one hundred iterations. Since the orientation scale factor in the objective function is the only changed variable in these two sets of experiments, it is safe to say that different objective functions influence the performance of a searching process by introducing different weights to their components.

The bottom row in Figure 4.2 shows the step scale factor during a searching process in these median runs. When the orientation scale factor is set to 1,000, the step scale factor increases linearly in the first one hundred iterations. Then the step scale factor fluctuates randomly in a range between 100 to 10,000 units. Compared with the bottom row in Figure 4.1, the step size factor in this case increases dramatically during the first one hundred iterations and stays in a range of value that is larger than the runs in the bottom row in Figure 4.1.

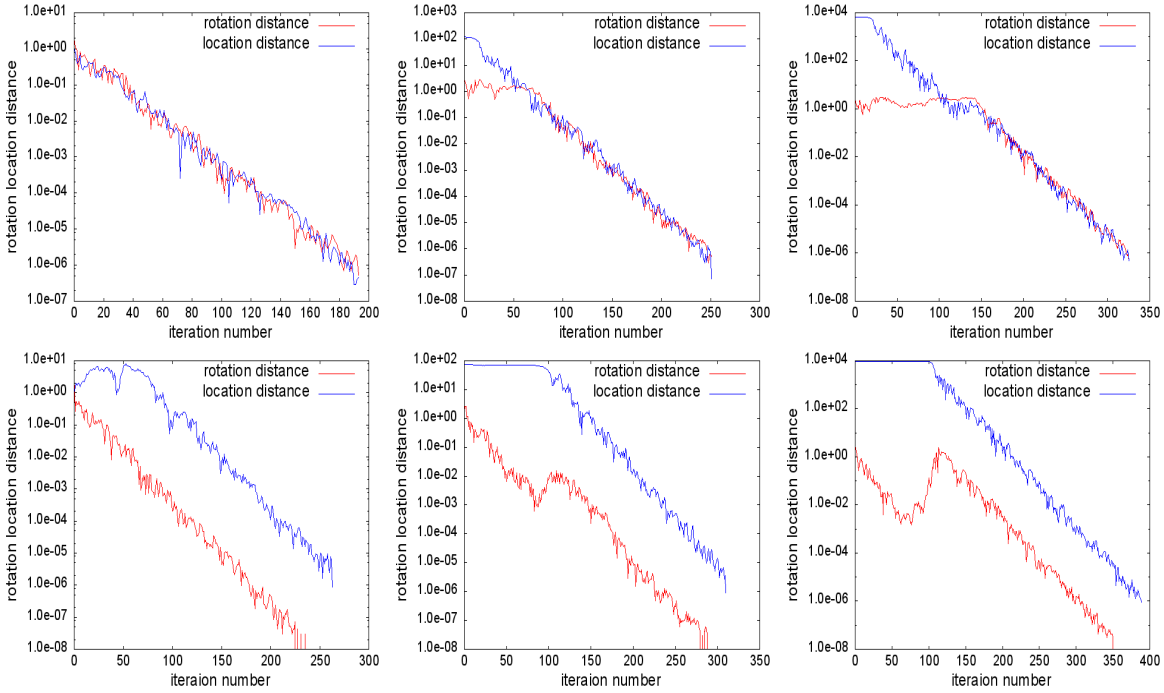

Figure 4.3: Orientation distance and location distance during search processes. Orientation scale factor is set to 1, initial location within different range scale: 1 unit (top left), 100 units (top middle), 10,000 units (top right). Orientation scale factor is set to 1,000, initial location within different range scale: 1 unit (bottom left), 100 units (bottom middle), 10,000 units (bottom right)

Conclusion

From the unimodal experiments with two objective functions, we see the algorithm converges to the target pose linearly with different initializations. During a searching process there are two phrases. In the first phase, the step size and step scale factor adjust to values to generate some meaningful progresses. In the second phase, the step size decreases linearly and the step scale factor fluctuates in a range; the objective function value decreases linearly.

4.2 Experiment in 3D environment

After testing our algorithm in the unimodal environment, we turn to evaluate it in synthetic environment, we conducted experiments in two different indoor environments, one is a classroom environment and the other is the Sagrada Família in Barcelona.

Target depth images are generated from the 3D models using OpenGL as opposed

to using a depth camera in the real environments being modelled. Our target images are thus ideal in that they are noise free, accurate up to the numerical limits of the depth buffer, and taken with the same camera that is used to evaluate candidate pose vectors. The global optimum of the registration problem thus has an objective function value of zero. The initial step size σ and mutation scale factor α are set such that the initial standard deviation of steps in any direction of the location subspace is about 1 meter, and the initial value of σ_{rot} is about 1. The search path is zero initially.

4.2.1 Classroom environment

For the classroom environment, the 3D model of the classroom we used was obtained from `http://www.blendswap.com` created by Andre Schneider under a Creative Commons license. The classroom we used in our project is a rectangular box with furniture (desks, chairs, etc.). The entire model consists of 180,518 triangles. One unit in the 3D model is approximately 1 meter.

We generate initial poses that serve as starting points for the search by uniformly and randomly sampling location from the interior of the classroom, with the constraint that any location has a distance of at least one meter from the walls, floor, and ceiling. The initial orientations are generated by uniformly and randomly sampling unit quaternions.

All runs are terminated when a candidate pose that is within 5 cm in terms of location and within 3° in terms of orientation of the target pose are found. We label runs in which such solutions are found successful. Runs are terminated as unsuccessful if after 500 iterations no solution satisfying the criteria for successful termination has been found.

Classroom experiment result

To test the algorithm in the classroom environment, we selected four target poses in this environment as shown in Figure 4.4. These four target poses vary from each other, trying to capture a wide range of different perspectives.

For each target pose, we have conducted 100 runs independently. The size of depth image that we used during experiments is 160×120. Table 4.3 summarises the

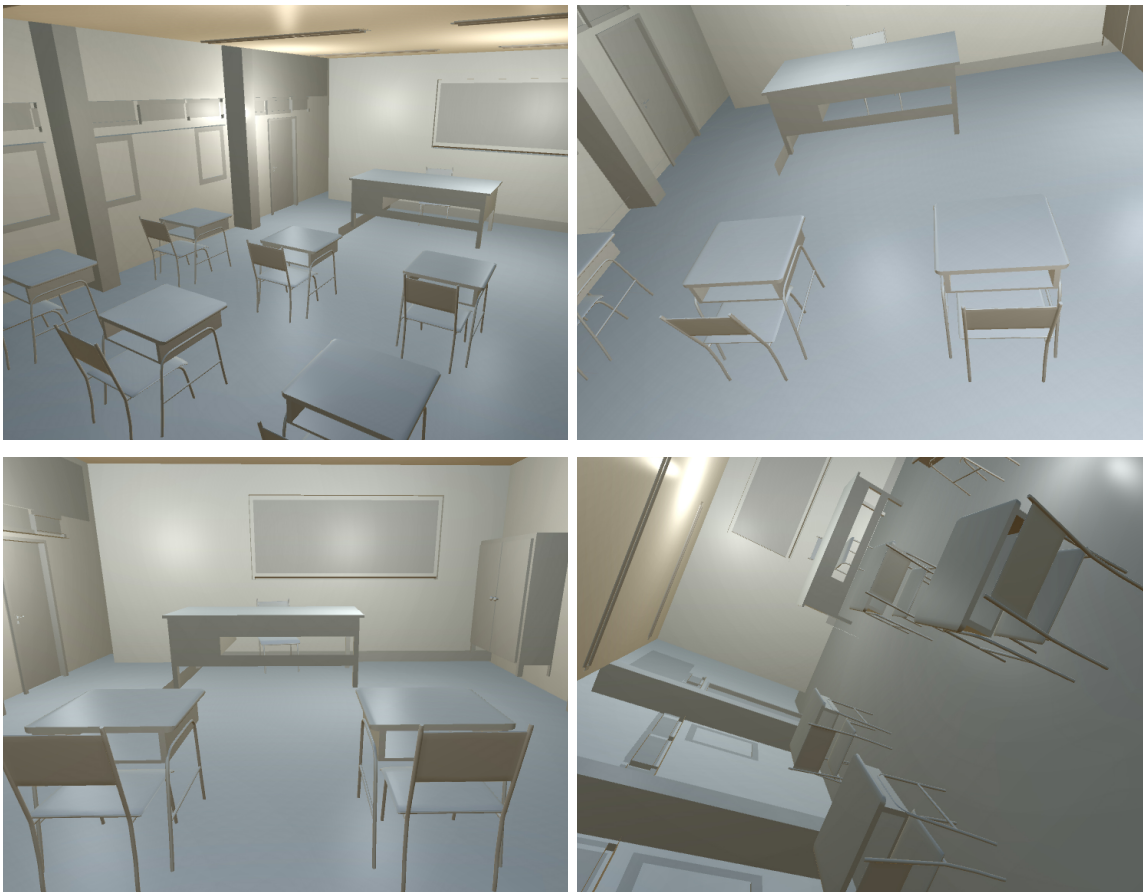

Figure 4.4: Views of the classroom test environment from the four target poses considered. Target poses A through D appear in clockwise order, starting at the top left

experimental results. The column labelled "success rate" indicates what fraction of the runs terminated successfully. The remaining columns give the median, mean, and standard deviation of the number of iterations required in those successful runs.

The success rate of each target is between 30% and 40%. An analysis of the unsuccessful runs shows that nearly 60% of the unsuccessful runs were terminated with an orientation of the camera that differs from the orientation of the target camera by an angle very close to 180°.

Figure 4.5 shows the evolution of the objective function value, the step sizes in the subspaces of locations and orientations employed by the evolutionary algorithm, and the distance from the target pose in both of those subspaces for those successful runs for each of the four targets that required the median number of iterations to terminate. The plots of the objective function value suggest that the algorithm

	success rate	median	mean	standard deviation
target A	31%	309	298.8	93.6
target B	39%	192	217.7	92.0
target C	30%	240	257.1	84.1
target D	35%	214	222.0	73.9

Table 4.3: Runs in the classroom environment

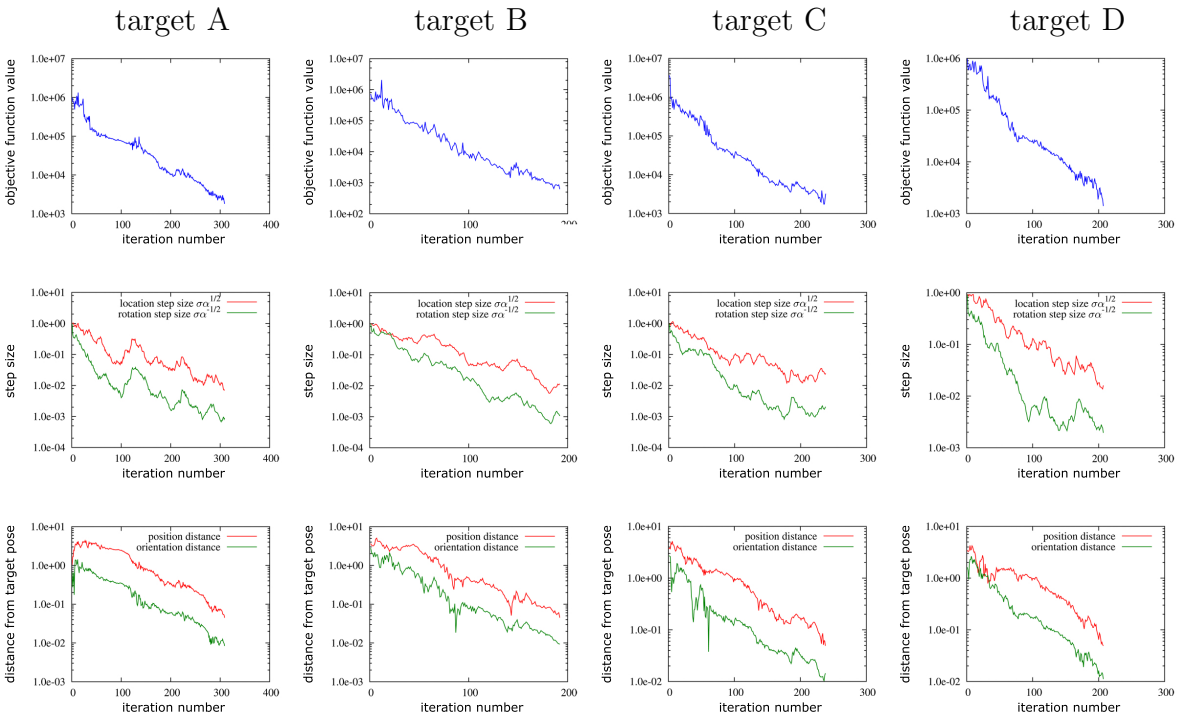

Figure 4.5: Median successful runs in the classroom environment. Shown are traces from those runs for each of the four targets that required the median number of iterations to terminate among all successful runs.

converges linearly. For all of the targets, the step size in the subspace of orientations eventually exceeds that in the subspace of orientations by approximately an order of magnitude, suggesting that self-adaptation of the step scale factor is successful.

Two different searching processes are visualized in Figure 4.6. The yellow cameras represent the initial camera pose that the search starts from. The black cameras represent the target pose, and the cameras that represent the pose centroids generated during the run of the evolutionary algorithm turn from blue to red as the search progresses.

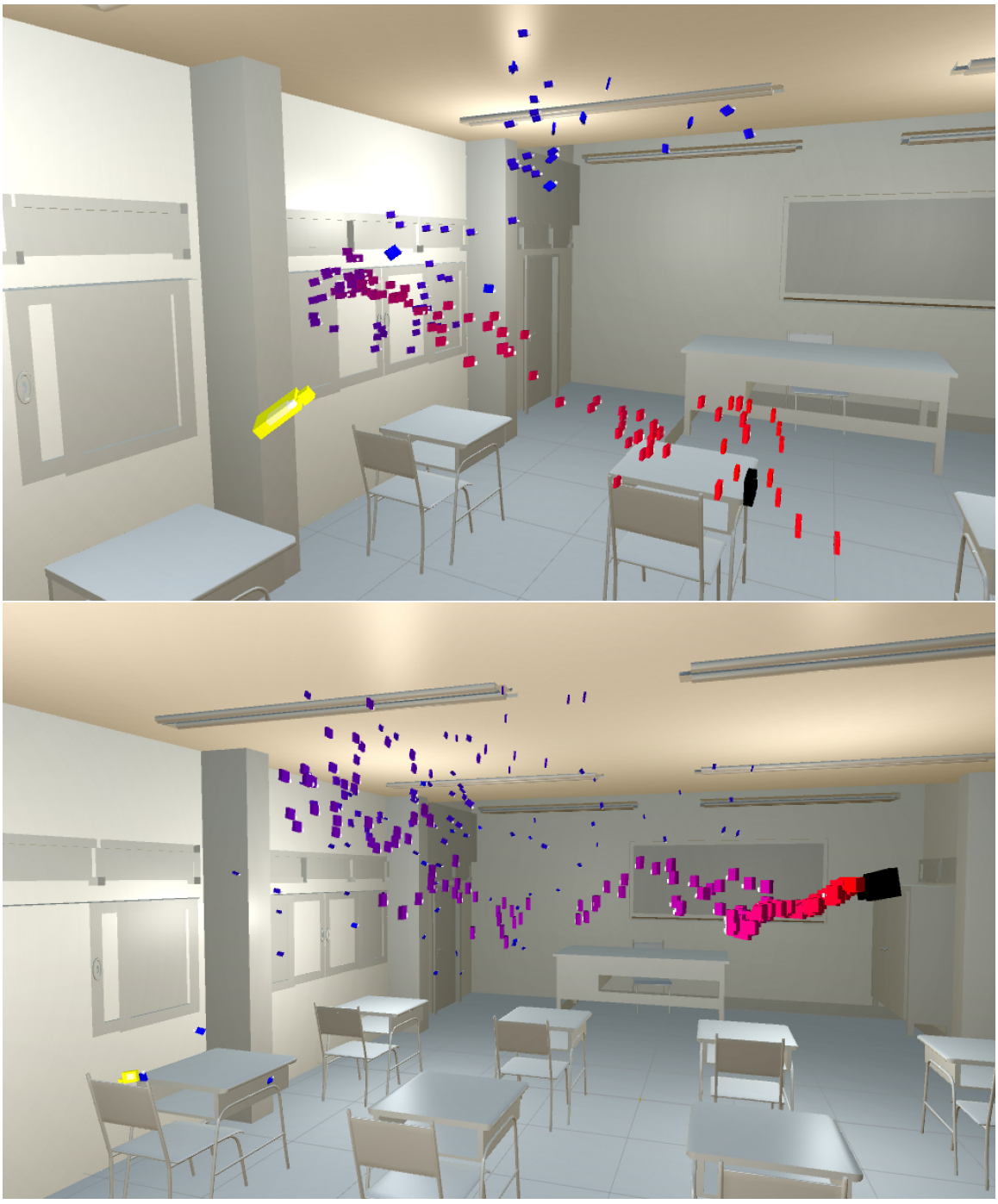

Figure 4.6: Traces from two successful runs for target poses A and D. In both cases, the yellow camera is the initial camera, the black camera is the target camera pose. The colour of camera illustrate the number of iterations. The start colour is blue with iteration increases the colour transit to red.

4.2.2 The Sagrada Família environment

For the church environment, a 3D model of the Sagrada Família in Barcelona, available from Dosch Design (`http://www.doschdesign.com`) is used. The church was originally designed by Spanish Catalan architect Antoni Gaudí. The Sagrada Família environment is of a scale much larger than the classroom, with the central nave vaults reaching a height of about 45 meters. The building's interior is characterized by vaulted ceilings, numerous columns. This 3D model consists of 245,189 triangles. To test the algorithm in the Sagrada Família environment, we selected four target poses as shown in Figure 4.7, trying to capture a wide range of different views.

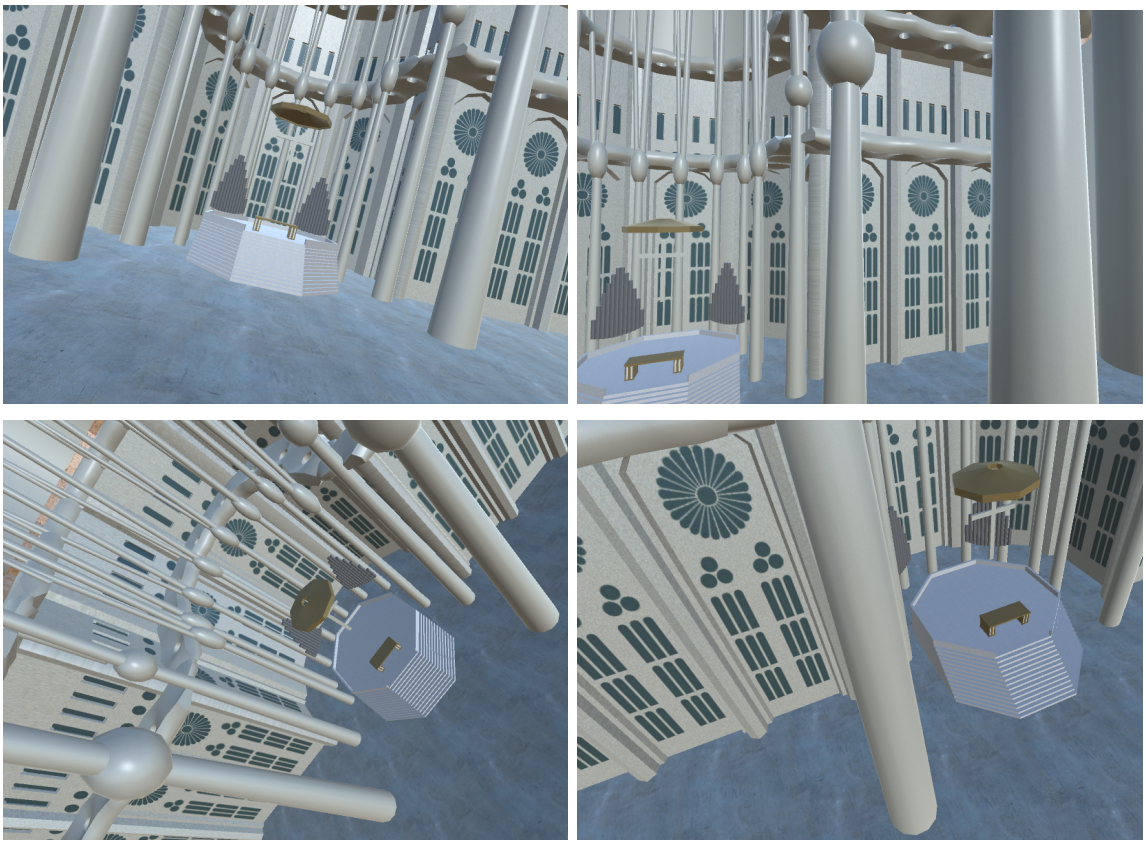

Figure 4.7: Views of the church test environment from the four target poses considered. Target poses A through D appear in clockwise order, starting at the top left

The terminating criteria used in the church environment are different from those used in the classroom environment due to a huge difference in scale. All runs are terminated when a candidate pose that is within 50 cm in terms of location and

within 3° in terms of orientation of the target pose are found. We label runs in which such solutions are found successful. Runs are terminated as unsuccessful if after 500 iterations no solution satisfying the criteria for successful termination has been found.

For the Sagrada Família environment, we generate initial locations uniformly at random within a ball of 10 m radius around the target pose, with the added constraint that starting poses are located in the interior of the building. The initial orientations are generated by uniformly and randomly sampling unit quaternions.

Sagrada Família experiment result

As with the classroom environment, we have conducted 100 independent runs for each of the four target poses in the Sagrada Família environment. The results are summarized in Table 4.4. It can be seen from the table that the success rates in the Sagrada Família environment for target poses A and C are lower than in the classroom one, though by a factor of no more than two. For target poses B and D, less than five out of the 100 runs successfully recovered the target pose.

	success rate	median	mean	standard deviation
target A	27%	296	288.3	101.5
target B	0%	0	0	0
target C	18%	331	326.8	127.9
target D	3%	124	120.7	3.1

Table 4.4: Runs in the Sagrada Família environment

Power function

To improve the searching success rate of the target poses B and D, the first attempt we tried is introducing an objective function that is slightly different from the previous one. In the previous objective function, the component that was used to evaluate the difference between two correspondence pixels in two depth images was a power function of the form:

$$f(x) = x^P$$

where the x is the difference between two corresponding pixels in two depth images, P is constant. In the previous objective function, P is set as 2.0 (to square the difference). In the new objective functions, we set the P values to 0.5 or 0.1. Figure 4.8 shows the depth image of target B (top left) and a depth image generated using similar pose vectors (top right). Shown at the bottom of the figure are the images obtained by computing powered differences between depth values in both $P = 2.0$ and $P = 0.1$. Both with intensities normalized. In the difference image computed with $P = 2.0$, the scoring value is dominated by the pixels around the pillars. With $P = 0.1$, the biggest difference is still in the pixels around the pillars, but the difference is less dominated by these pixels. To experiment, we have conducted 100 independent runs for each of the four target poses with different P values of 2.0, 0.5 and 0.1.

Figure 4.8: Depth image of target B (top left), a candidate depth image (top right) and the images obtained by computing pixel-wise differences with different power value $P = 2.0$ (bottom left) and $P = 0.1$ (bottom right; both with intensities normalized).

The results of the experiments with different P values are summarized in Table 4.5. It can be seen that with a smaller P value such as 0.1 and 0.5, the success rates in target poses B, C and D increase significantly with a loss of success rates in target pose A. Further, the average number of iterations that are needed to reach the target poses is smaller as the P value decreases.

$P = 2.0$				
	success rate	median	mean	standard deviation
target A	27%	296	288.3	101.5
target B	0%	0	0	0
target C	18%	331	326.8	127.9
target D	3%	124	120.7	3.1

$P = 0.5$				
	success rate	median	mean	standard deviation
target A	11%	120	135.5	32.3
target B	29%	210	217.2	86.8
target C	38%	150	159.5	65.0
target D	18%	169.5	168.2	65.0

$P = 0.1$				
	success rate	median	mean	standard deviation
target A	6%	123	119.3	23.6
target B	33%	145	161.6	70.6
target C	32%	132	134.2	39.4
target D	18%	145.5	154.9	41.7

Table 4.5: Runs in the Sagrada Família environment with different P values

Figure 4.9 shows the evolution of the objective function value, the step sizes in the subspace of locations and orientations employed by the evolutionary algorithm, and the distance from the target pose in both of those subspaces for those successful runs for each of the four targets that required the median number of iterations to terminate (in these experiments the P value used in the objective function is set as 0.5, since using the previous objective function cannot provide adequate successful runs). The plots of the objective function value suggest that the algorithm converges

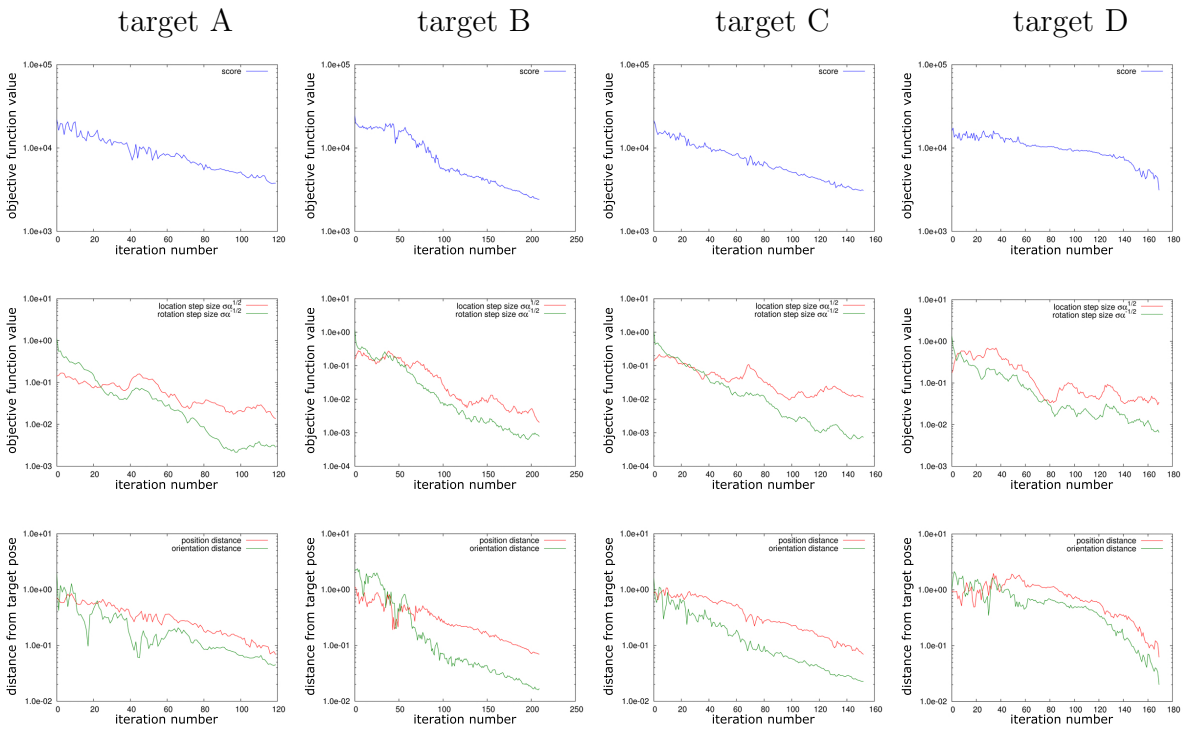

Figure 4.9: Median successful runs in the church environment. Shown are traces from those runs for each of the four targets that required the median number of iterations to terminate among all successful runs.

linearly in targets A, B, and C. For the target D, the algorithm converges slower than in the other three target poses around the first one hundred iterations. Then it converges faster and reaches the target pose.

4.3 Local Optima

A typical problem optimization algorithms face is avoiding sticking in local optima. A local optimum is a solution that is the best solution within a neighbourhood of possible candidates but is not necessarily the best solution among all possible solutions. As evolutionary strategy is an optimization strategy that is designed to search for optima, it selects the best solutions within each generation to get closer to an optimum; however even if it finds an optimum within a certain area of the search space, it lacks the vision to reveal whether this optimum is the global one or not. As an algorithm reduces its searching area by decreasing the step size in order to converge to an optimum more efficiently, it is hard for the strategy to jump out of this neighbourhood and start searching for the global optimum.

During the experiments in two indoor environments, one major problem we are still facing is that the algorithm fails to reach the global optimum (around 70% in each set of experiments). From the unimodal test with different initializations, we see the algorithm is solid enough to reach the optimum after hundreds of iterations as there is only one optimum in the search space. Unfortunately, in the experiments with two indoor environments (classroom and church), some searching processes sometimes result in other poses instead of the target one. Presumably, there may exist some local optima that our algorithm cannot avoid or get rid of and this leads to a failed searching process. In order to have a better understanding of where these failed searches end, we plotted all the results of the 100 runs in an experiment. For this experiment, we generate initial location uniformly at random with a ball of 5 m radius around the target pose. The initial orientations are generated by uniformly and randomly sampling unit quaternions.

Figure 4.10 shows the classified 100 run-results of the target A in the church environment where $P = 2.0$. During the experiment, all the runs that reached the target pose successfully are shown inside the red border. For the unsuccessful runs, many of them converge to certain poses frequently (examples are shown inside different colour borders, for each colour of border encloses one pose). Statistically, those poses which are not the target but the algorithm converges to them frequently are probably the local optima. By analysing the optical information of those unsuccessful images, we can see that these images have similar optical information as that of the target image: either the ground in both images is almost the same or the position of pillars is close or both. This indicates that where these unsuccessful runs converged are probably the local optima. And with a close look at Figure 4.10, we hypothesize that the corresponding poses correspond to local minima of the objective function that are a result of the symmetries present in the geometric model.

4.4 Multimodality

A common approach to deal with multimodality is to detect stagnation in a searching process and to restart with independently sampled initial conditions. In order to increase the success rate with a minimum redundant computational cost, based on our algorithm, we implemented a system for multimodal optimization which can

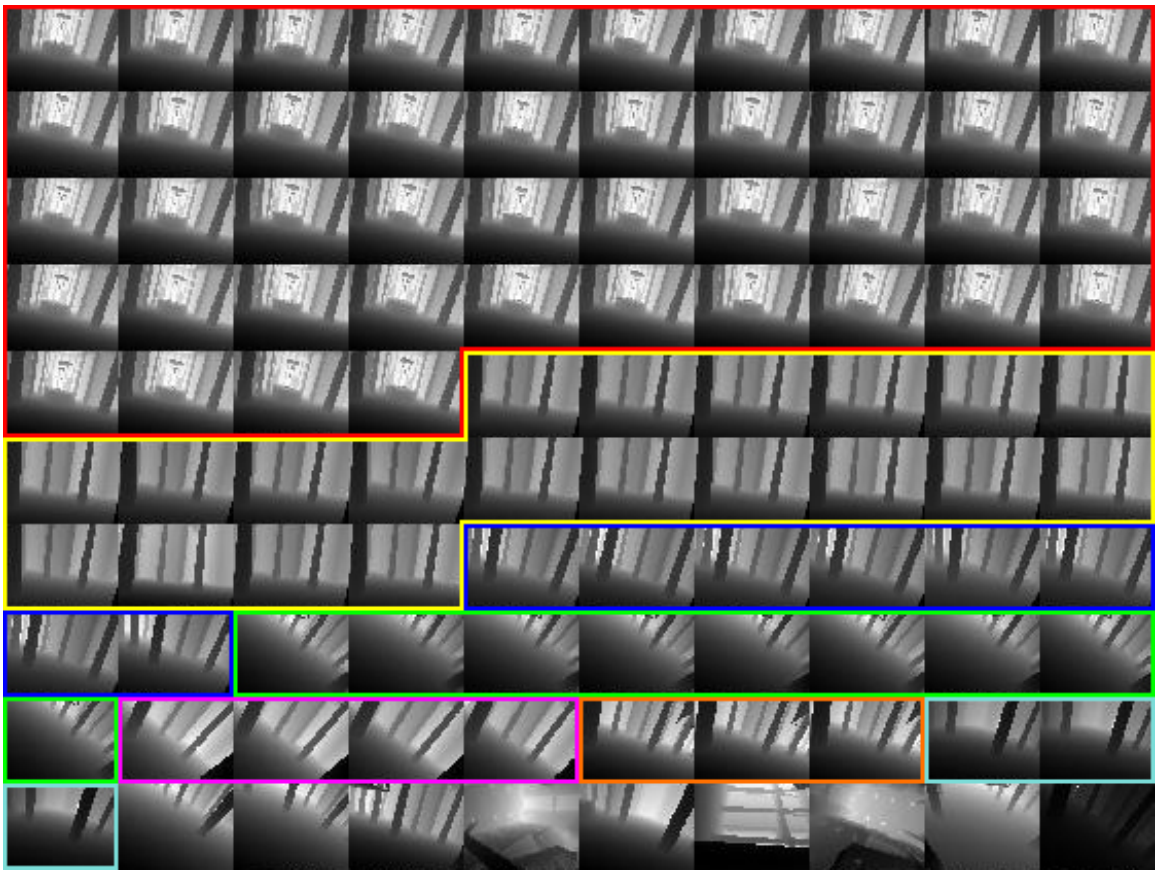

Figure 4.10: Classified results of 100 searching processes of Target A in the Sagrada Família environment. Red border enclose the result of all the successful runs ($P = 2.0$). Each border that with other colour enclose a pose that been reached more than once. The poses that have no border around are the poses that only been reached once.

conduct multiple searching processes simultaneously, then detect and select the most possible candidate to complete the searching process. This system is similar to the two-tier mechanism described by Fillingham in [8]; both methods start with multiple threads and select the best one to complete the searching process. By computing the objective function values in the current set of offspring poses, we usually can detect whether a searching process converges to a global optimum solution or not, within the first 100 iterations. Assuming a likelihood of p converging to the globally optimal solution in a single run, the number of restarts required to locate the globally optimal solution with probability $1 - \epsilon$ is $\lceil \log_{1-p} \epsilon - 1 \rceil$. Assuming $p = 0.3$ (the lowest value encountered across the four target images in the classroom environment), the number of restarts required to locate the globally optimal solution with probability 0.95 is

thus 8; the number of restarts required to locate it with probability 0.99 is 12.

	success rate	median	mean	standard deviation
target A	100%	161	166.9	54.4
target B	100%	132	142.2	54.6
target C	100%	154.5	158.3	41.8
target D	100%	108	115.3	32.9

Table 4.6: Runs in the classroom environment with a system for multimodal optimization. Need to notice that there are an extra 1900 iterations for the runs chosen not to pursue.

The multimodal system supports 20 individual runs at a time in one searching process. The system selects the run which produces the best solution within the first 100 iterations among all runs, and discards the others as assuming they are less likely to reach the global optimum. Then the system keeps the preserved run running until it gets a result. We conducted 100 independent runs for the four target poses in the classroom environment. Table 4.6 summarises the experimental results. The column labelled "success rate" indicates what fraction of the runs terminated successfully. The remaining columns give the median, mean, and standard deviation of the number of iterations required in those successful runs. With this multimodal system, the success rate of all the four target poses reached 100%.

4.5 Computational cost

We did a performance-analysis with our program during the experiments in the church environment to investigate the computational cost of our method. The major configuration of our testing environment is shown in Table 4.7.

Processor name	Intel Core i7
Processor speed	2.6 GHz
Memory	16 GB
Graphic card	NVIDIA GeForce GT 650M
Video RAM	1024 MB

Table 4.7: Testing environment configuration

During a single search iteration (as introduced in Figure 3.2), we found more than 95% of the time is spent on these sections: a) reading a block of pixels from the depth buffer, b) transforming the depth value from window coordinates to camera coordinates, c) calculating the objective function. These three sections are the largest bottlenecks in our implementation; a) is acquired from the glReadPixels function in OpenGL, b) and c) are implemented on the CPU. However, all the three sections are influenced by one factor which is the scale of a rendered depth image; the largest bottleneck in this optic based algorithm is caused by the scale of the optical information it processes. The size of an image and the amount of time spent on these three sections are in positive correlation. Generally, a larger image provides more information but at the same time may jeopardise the searching speed. The ideal size of an image for the experiments would be one that is large enough to provide sufficient information which leads to a reliable searching accuracy and at the same time is as small as possible to keep the searching processes run as fast as possible.

During the experiments, we found different sizes of images may influence the computational cost in these sections. Figure 4.8 shows the computational cost ratio of each section within an iteration, with a size of 40×30 pixels: 91.7% computational cost is spent on reading a block of pixels from the depth buffer, 4.3% of computational cost is spent on transforming the coordinates, and 1.0% of computational cost is spent on calculating the objective function. With a size of 1280×960 pixels: 25.9% computational cost is spent on reading a block of pixels from the depth buffer, 58.7% of computational cost is spent on transforming the coordinates, and 15.1% of computational cost is spent on calculating the objective function. The larger the image used, the more the ratio of computational cost spent on transforming the coordinates and calculating the objective function increases, and the ratio of computational cost spent on reading the pixels from depth buffer decreases.

We tracked the amount of time that is spent on reading a block of pixels from the depth buffer, transforming from window coordinate to camera coordinates and calculating the objective function with different size of images separately. Table 4.9 shows the time cost during reading a block of pixels from different size of depth buffers. Reading a block of pixels from a 1280 by 960 pixels depth buffer costs 4.557 milliseconds while reading from a 40 by 30 pixels depth buffer costs 0.219 milliseconds.

Image size	Reading pixels	Transforming coordinates	Objective function
40×30	91.7%	4.3%	1.0%
80×60	83.8%	11.0%	2.8%
160×120	67.6%	25.3%	4.8%
320×240	55.8%	21.8%	12.6%
640×480	30.0%	55.8%	13.7%
1280×960	25.9%	58.7%	15.1%

Table 4.8: Average ratio of three most computational costly sections during a searching iteration with different size of images

Image size	Time cost (millisecond)	Image size	Time cost (millisecond)
40×30	0.219	80×60	0.251
160×120	0.527	320×240	0.786
640×480	1.026	1280×960	4.557

Table 4.9: Average time cost on reading a block of pixels from the depth buffer with different size of images

Table 4.10 shows time cost during transforming the depth value from window coordinates to camera coordinates. With the largest size, 1280 by 960 pixels, transforming coordinates costs 10.325 milliseconds while the smallest size, 40 by 30 pixels, costs 0.028 milliseconds.

Image size	Time cost (millisecond)	Image size	Time cost (millisecond)
40×30	0.028	80×60	0.067
160×120	0.209	320×240	0.745
640×480	2.681	1280×960	10.325

Table 4.10: Average time cost on transforming the depth value from window coordinate to camera coordinate with different size of images

Table 4.11 shows the time cost during calculating the objective function with different sizes of images. With the largest image size of 1280 by 960 pixels, calculating the objective function costs 3.984 milliseconds while the smallest size of 40 by 30 pixels costs 0.019 milliseconds.

In the experiments, the size of image is 160 by 120 pixels which costs 0.527 milliseconds on average in reading a block of pixels from the depth buffer, 0.209 milliseconds

Image size	Time cost (millisecond)	Image size	Time cost (millisecond)
40×30	0.019	80×60	0.029
160×120	0.075	320×240	0.281
640×480	1.005	1280×960	3.984

Table 4.11: Average time cost on calculating objective function with different size of images

on average on transforming the depth value from window coordinates to camera coordinates, and 0.075 milliseconds on average in calculating the objective function. In the preliminary experiment, we found a smaller image (e.g. 30×40 pixels image and smaller) will lead the success rate of a searching process to drop significantly, presumably because the information stored in the image is not sufficient. On the other hand, the success rate of a searching process will increase as the size of image processed increases. However, in the preliminary experiment, there is no significant increase in the success rate if the image size is larger than 160×120 pixels.

Chapter 5

Conclusion and Future Work

"Change is one thing, progress is another."
— B. Russell, Unpopular Essays

In this thesis, we have presented an evolutionary algorithm for the optimization of pose vectors. The algorithm simplifies the covariance matrix adaptation evolution strategy for optimization on general Riemannian manifolds by Colutto et al. [6] in that it does not adapt the full covariance matrix governing the sampling of offspring candidate solutions, but instead uses self-adaptation to control a factor that determines the size of steps in the subspace of locations relative to that in the subspace of orientations. An advantage of the simplified algorithm is that it is easier to experimentally analyze its performance.

We have then applied that algorithm to determine the pose of a depth camera based on a single target depth image and a 3D model of the indoor environment that the image was taken in. Evaluation of the objective function for a candidate pose requires generating a depth image of the model from that pose and then summing squared differences between the depth values obtained and target depth values. We have found that the algorithm is able to recover the target pose in about one third of the runs when applied in a classroom environment with random initialization, and in a much larger environment of a church if initialized within about 10m of the target pose. The results are encouraging as modern graphics hardware makes it possible to conduct many evaluations of the objective function in a short amount of time. Independent restarts of the algorithm can be used to arbitrarily increase the likelihood of locating the target pose.

The future works here are in multiple directions. A systematic analysis of how the likelihood of recovering the target pose varies with population size parameters μ and λ will allow us to determine whether it is advantageous to perform restarts with the

same values of the population size parameters or the approach of doubling population size parameters between runs proposed by Auger and Hansen [2] is advantageous. The observation of symmetries in the search space suggests that the pose estimation problem may be characterized by rather large basins of attraction, and that the former approach may be preferable.

A second line of inquiry is to try to improve the global search capability of the algorithm, thus making it applicable in larger environments. The insight that unsuccessful runs tend to recover orientations that differ from the target ones by 180° offers opportunities for improved initialization conditions when restarting. Also of interest will be the use of other approaches for dealing with multimodality that strive to balance exploration and exploitation, e.g. through the minimization of cumulative regret. There is also the option of nonlinearly transforming differences between depth values before summing them up, thus modulating the objective function.

And finally, it would be interesting to evaluate our approach in real environments, with target images taken using depth cameras rather than being generated synthetically from 3D models. The noisy nature of those images will pose difficulties in the optimization process not present in our current set-up.

Bibliography

[1] D. V. Arnold. On the use of evolution strategies for optimization on spherical manifold. *Parallel Problem Solving from Nature—PPSN XIII*, pages 882–891, 2014.

[2] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation*, pages 1769–1776, 2005.

[3] J. Aulinas, Y. R. Petillot, J. Salvi, and X. Lladó. The SLAM problem: a survey. In *Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 363–371, 2008.

[4] H.-G. Beyer and H.-P. Schwefel. Evolution strategies–a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

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

[6] S. Colutto, F. Frühauf, M. Fuchs, and O. Scherzer. The CMA-ES on Riemannian manifolds to reconstruct shapes in 3-D voxel images. *IEEE Transactions on Evolutionary Computation*, 14(2):227–245, 2010.

[7] G. N. DeSouza and A. C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transcations on Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002.

[8] J. Fillingham. Exploring the depth-image based fitness landscape for indoor localization. Honours thesis, Faculty of Computer Science, Dalhousie University, 2013.

[9] M. A. Fischler and R. 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.

[10] National Coordination Office for Space-Based Positioning. Global positioning system standard positioning service signal specification. *United States Coast Guard Navigation Center*, 1995.

[11] N. Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[12] N. Hansen, D. V. Arnold, and A. Auger. Evolution strategies. In *Springer Handbook of Computational Intelligence*, pages 871–898. Springer, 2015.

[13] N. Hansen and A. Ostermeier. Completely derandomized adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[14] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1):228–234, 1990.

[15] S. Huckemann, T. Hotz, and A. Munk. Intrinsic MANOVA for Riemannian manifolds with an application to Kendall's space of planar shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):593–603, 2010.

[16] T. K. Kohoutek, R. Mautz, and J. D. Wegner. Fusion of building information and range imaging for autonomous location estimation in indoor environments. *Sensors*, 13(2):2430, 2013.

[17] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[18] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua. Worldwide pose estimation using 3D point clouds. In *Computer Vision–ECCV 2012*, pages 15–29. Springer, 2012.

[19] J. J. Lim, H. Pirsiavash, and A. Torralba. Parsing IKEA objects: Fine pose estimation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2992–2999, 2013.

[20] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[21] A. Lu and D. V. Arnold. An evolutionary algorithm for depth image based camera pose estimation in indoor environments. In *IEEE Congress on Evolutionary Computation*, 2016.

[22] Q. Luong and O. D. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices. *International Journal of Computer Vision*, 22(3):261–289, 1997.

[23] R. Mautz. Indoor positioning technologies. Habilitation thesis, ETH Zürich, 2012.

[24] F. Moreno-Noguer, V. Lepetit, and P. Fua. Pose priors for simultaneously solving alignment and correspondence. In *European Conference on Computer Vision*, pages 405–418. Springer, 2008.

[25] H. Murase and S. K. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[26] L. M. Ni, Y. Liu, Y. C. Lau, and A. P Patil. Landmarc: indoor location sensing using active RFID. *Wireless Networks*, 10(6):701–710, 2004.

[27] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1994.

[28] I. Rechenberg. *Evolutionsstrategie–Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[29] T. Sattler, B. Leibe, and L. Kobbelt. Improving image-based localization by active correspondence search. In *European Conference on Computer Vision*, pages 752–765. Springer, 2012.

[30] M. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.

[31] K. Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 245–254. ACM, 1985.

[32] L. Svarm, O. Enqvist, M. Oskarsson, and F. Kahl. Accurate localization and pose estimation for large 3D models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 532–539, 2014.

[33] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2006.

[34] R. Szeliski. *Computer Vision: Algorithms and Applications*, pages 303–332. Springer Science & Business Media, 2010.

[35] R. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, 1987.

[36] N. Vajramushti, I. A. Kakadiaris, T. Theoharis, and G. Papaioannou. Efficient 3D object retrieval using depth images. In *6th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR)*, pages 189–196, 2004.

[37] Wikipedia. List of camera types — Wikipedia, the free encyclopedia, 2016. [Online; accessed 12-July-2016].

[38] Z. Zhang. Camera calibration with one-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):892–899, 2004.

[39] M. Z. Zia, M. Stark, B. Schiele, and K. Schindler. Revisiting 3D geometric models for accurate object shape and pose. In *IEEE International Conference on Computer Vision (ICCV Workshops)*, pages 569–576, 2011.

[40] Y. Zou, W. Chen, X. Wu, and Z. Liu. Indoor localization and 3D scene reconstruction for mobile robots using the Microsoft Kinect sensor. In *IEEE International Conference on Industrial Informatics*, pages 1182–1187, 2012.