Implementing Camera Pose Estimation by Grey Wolf Optimizer

Artificial Intelligence

Ferdowsi University of Mashhad

[transcendentimagination@gmail.com](mailto:transcendentimagination@gmail.com)

Abstract

In this report, we’re going to improve the camera pose estimation from two images taken by two uncalibrated cameras by the Grey Wolf Optimizer algorithm. First we estimate random camera poses by the five point algorithm as the initial population and then we try to find the optimum solution with the Grey Wolf Optimizer. This is an effort to implement the required algorithm and fitness calculations according to the Elashry and Toth [1] that introduces this method.

Keywords

Camera Pose Estimation, Essential Matrix, Fundamental Matrix, Five Point Algorithm, Grey Wolf Optimizer, Swarm Particle Algorithm

Table of contents

[Implementing Camera Pose Estimation by Grey Wolf Optimizer 1](#_Toc171703079)

[Abstract 1](#_Toc171703080)

[Keywords 1](#_Toc171703081)

[Table of contents 2](#_Toc171703082)

[Table of figures 2](#_Toc171703083)

[1- Introduction 3](#_Toc171703084)

[2- Improving Camera Pose Estimation Using Swarm Particle Algorithms 3](#_Toc171703085)

[3- An Efficient Solution to the Five-Point Relative Pose Problem 3](#_Toc171703086)

[4- Grey Wolf Optimizer 3](#_Toc171703087)

[5- Fitness Function 3](#_Toc171703088)

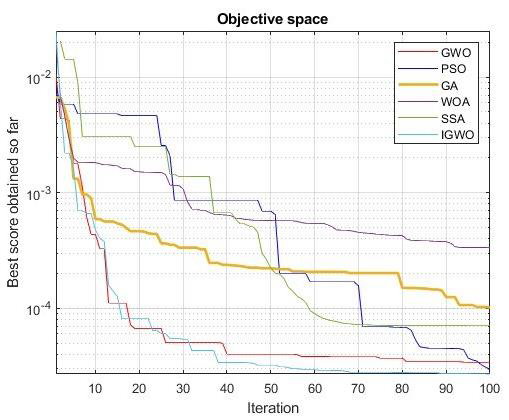
[6- Implementation 4](#_Toc171703089)

[7- Results 4](#_Toc171703090)

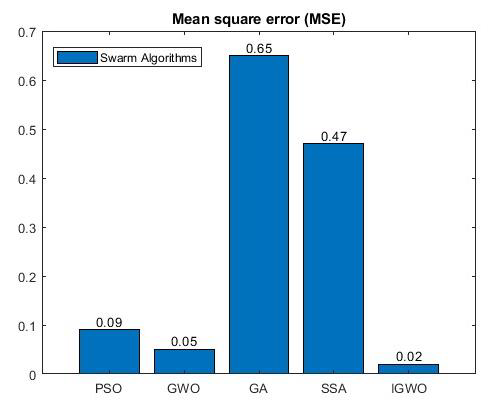
[8- Conclusion 4](#_Toc171703091)

[9- References 4](#_Toc171703092)

Table of figures

[ 7](#_Toc171707267)

[Figure 1: Results obtained from metaheuristic optimization algorithms. 7](#_Toc171707268)

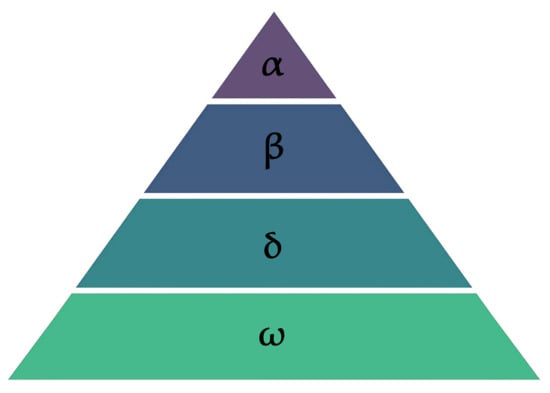
[ 8](#_Toc171707269)

[Figure 2: Histogram of comparing between the algorithms 8](#_Toc171707270)

[A graph with numbers and a number of blue bars

Description automatically generated with medium confidence 9](#_Toc171707271)

[Figure 3: Histogram of computational time for the different algorithms 9](#_Toc171707272)

[ 10](#_Toc171707273)

[Figure 4: Hierarchy of grey wolf (dominance decreases from top down) 10](#_Toc171707274)

[Figure 1: Camera Pose Fitness Function 11](#_Toc171707275)

# Introduction

First, we introduce the articles used for this work and then we explain the implementations for the algorithms. Finally, we test and compare the results. The problems is follows: We have 2 images captured by the same or different cameras at different positions and rotations (poses). We want to estimate the relative orientation between these two poses from the 2 captured images. In this problem, we know the intrinsic parameters of the cameras, but we don’t have any knowledge about the scene. The general flow is like this:

* Estimate some camera poses with five-point algorithm as the initial solutions of the swarm algorithm. We do it by repeating the five-point algorithm multiple times by choosing random matched points.
* Then, we apply the Grey Wolf Optimizer over the solutions until certain iterations. We choose the alpha wolf as the best solution.

For finding the feature points and matching them, we use the SURF algorithm. For the fitness function we use an innovative way the calculated the distances of the match points on the assumed second camera

# Improving Camera Pose Estimation Using Swarm Particle Algorithms

Camera pose estimation has been a common problem in the field of computer vision. There have been introduced many methods to solve it. They all propose valid solutions and estimate good poses for the cameras, but they may or may not be good enough. The proposed method in that article aims to improve those solutions by evolutionary algorithms (EA) or swarm algorithms (SA) to find the optimum solution in the search space.

It first describes some of the EA and SA algorithms that it wants to apply on the pose estimation including the Genetic Algorithm, Particle Swarm Optimization, Grey Wolf Optimization, Improved Grey Wolf Optimization, and Whal Optimization. Then, it compares the results of these algorithms on the SPIN Lab dataset by best obtained score, mean square error, and computation time. Finally, it concludes that the IGWO performs best in finding the optimum solution and requires most computation time.

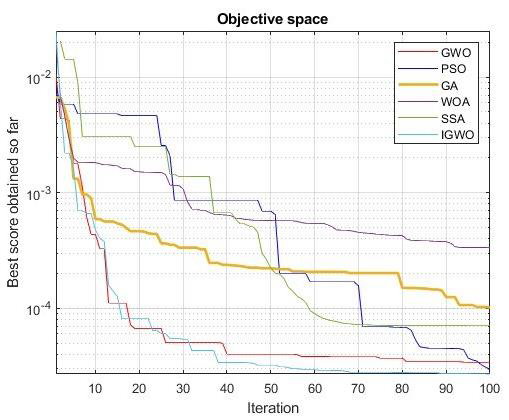


Figure 1: Results obtained from metaheuristic optimization algorithms.

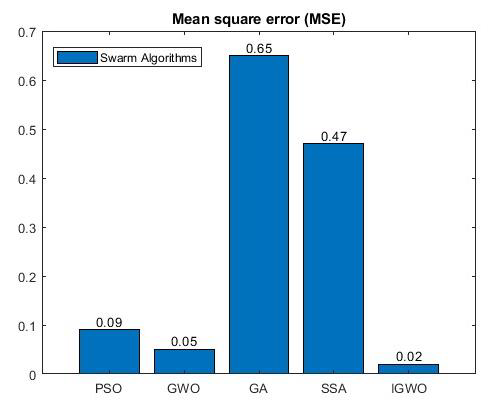


Figure 2: Histogram of comparing between the algorithms

A graph with numbers and a number of blue bars

Description automatically generated with medium confidence

Figure 3: Histogram of computational time for the different algorithms

# An Efficient Solution to the Five-Point Relative Pose Problem

This article tries to find the essential matrix by giving five matched points between two images and knowing the intrinsic parameters of the cameras. It solves a tenth-degree polynomial equation to find the coefficients.

Note that in this report, we use a different five-point algorithm from what the Elashry and Toth [1] used in its experiments.

# Grey Wolf Optimizer

This algorithm Mirjalili, Mirjalili [2] aims to simulate the hunting behavior of the grey wolfs who move in a pack usually consists of 5-12 members. It considers each wolf as candidate solution and the prey as the best solution trying to find. It first categorizes the wolfs into a hierarchy of ranks that determines how the wolfs will move in the search space. There are leader wolfs from the top to the bottom named alpha, beta, and delta wolfs. There is only one of each rank. They are considered as the best three solutions found so far. These leaders are assumed to know the most probable position for the prey. They are responsible for pursuing the solution and leading the pack. The other wolfs are called omega and the move according to the movement of the leaders and some randomness. They follow this pattern until they encircle and hunt the prey.

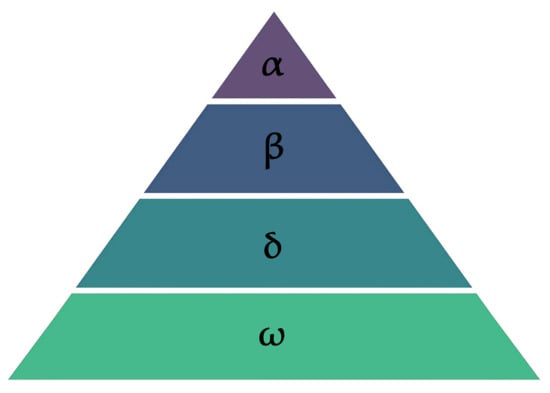


Figure 4: Hierarchy of grey wolf (dominance decreases from top down)

The method consists of different aspects:

## Encircling

The wolves try to close the prey. So, they need to decrease their distance from the prey over time. It controls this closing with A and C vectors. The Xp is the position of the prey and the X(t) is the position of the wolf. On each iteration, the wolf updates its position to get closer to the prey.

The coefficients A and C are determined by the following formulas:

The *a, r1*, and *r2* are coefficient vectors. These components decrease from 2 to 0 linearly to make the wolves close the prey over time and bring convergence. The *r1* and *r2* are randomly chosen between 0 and 1 to allow divergence and the wolf can choose any position around the prey.

## Hunting

The wolves don’t know the position of the prey. We consider the first 3 best positions as the alpha, beta, and delta wolves. So, the omega wolves will update their position according to these leader wolves instead of the prey’s unknown position. This makes the omega wolves follow the leaders and surround and close the prey. So, the formulas are changed like this:

## Attacking

When the coefficient *|A|* is less than 1, is makes the wolf to close the prey. When it’s greater than 1, it makes the wolf to gets farther from the prey for searching and exploration. The *a* coefficient is decreasing from 2 to 0. So, the A is in the range *[-a,a]*. This means the *|A|* coefficient is slowly decreasing and finally causing the wolves to reach the prey.

## Searching

When the |A| > 1, it forces the wolves to diverge and search for the prey. Also, the C coefficient has random values in the range [0, 2], so, sometimes the C > 1 and sometimes C < 1. This allows more randomness during exploration and allows avoiding the local optima better. This means the wolves sometimes close the prey and sometimes search for the prey.

## Algorithm

The algorithm of the Grey Wolf Optimizer is as follows:

*Initialize the grey wolf population Xi (i = 1, 2, ..., n)*

*Initialize a, A, and C*

*Calculate the fitness of each search agent*

*Xα=the best search agent*

*Xβ=the second best search agent*

*Xδ=the third best search agent*

***while*** *(t < Max number of iterations)*

***for*** *each search agent*

*Update the position of the current search agent by equation (3.7)*

***end for***

*Update a, A, and C*

*Calculate the fitness of all search agents*

*Update Xα, Xβ, and Xδ*

*t=t+1*

***end while***

*return Xα*

First, an initial population of the wolves is created. Then we initialize the coefficients a, A, and C. Then we calculate the fitness of all the wolves and chose the best 3 candidates as the alpha, beta, and delta. Then, we start the iterations by first updating the positions of the wolves according to the leader wolves. Then we repeat updating the coefficients, calculating the fitness and choosing the new leaders. We repeat it until some criteria or maximum number of iterations. Then, the alpha is chosen as the best answer.

# Fitness Function

To give a fitness value to each candidate relative pose, we need to define a value that describes how accurate is this pose if it was the pose for the second camera. The matched points found by the SURF algorithm tell us that they are the same points in the 3D space. So, we simulate reconstructing those points in the 3D space and examine how feasible it is. For each matched point in the first camera, we find a ray emitting from the camera origin toward a that pixel on hypothetical plane in front of that camera in the pinhole camera model. This ray will cross the real point the 3D space. We do the same for that matched point on the second image with assumed second camera pose. Then, we determine how close these to rays are by finding the distance between those lines. We repeat this process for all matched points and give the sum of those distances as the fitness value for that candidate relative pose (solution). Because the GWO algorithm tries to find a solution with the minimum fitness, the distance sum must also be minimized.

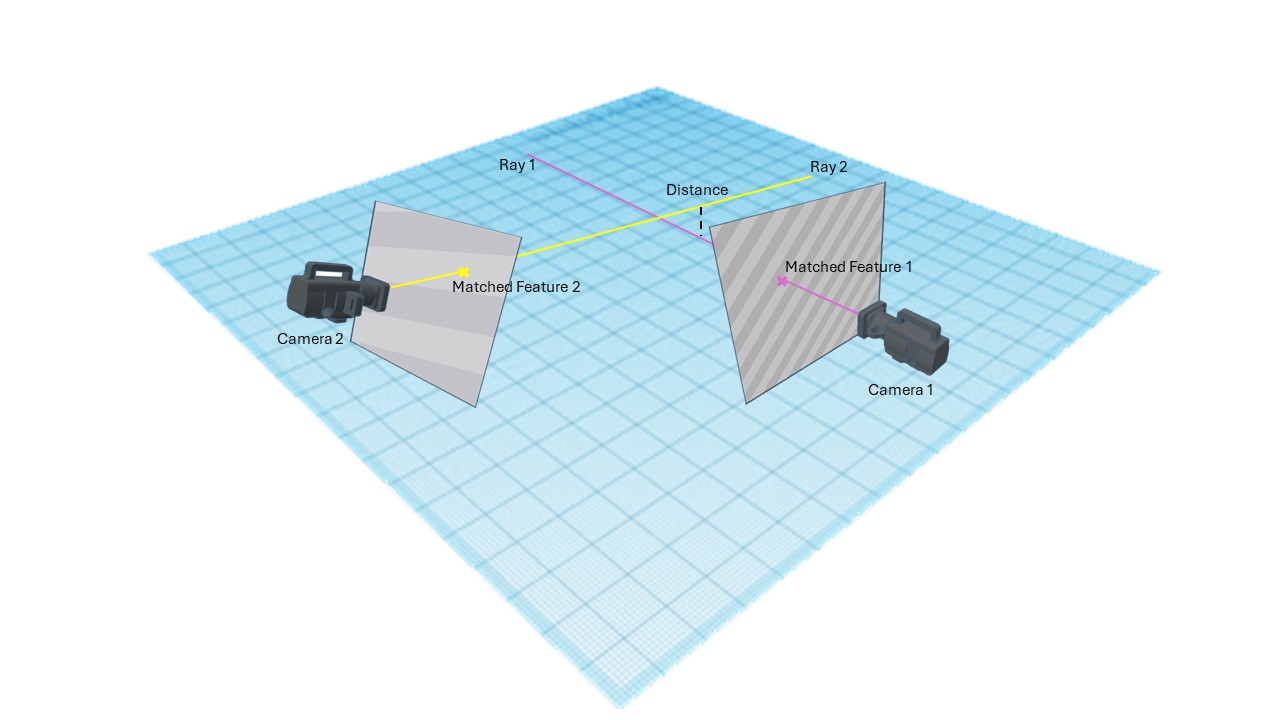


Figure 1: Camera Pose Fitness Function

# Implementation

In this implementation, we define structure of each of the solutions in GWO as the homogeneous rotation-translation matrix that defines the transformation from the first camera pose to the second camera pose.

## Finding Matched Points

We first remove the distortions of the images captured by the cameras to flatten them. Because the images are colorful, we convert them to gray-scale images.

A black text on a white background

Description automatically generated

We extract the features with the SURF algorithm and then we find the matched features. These points serve the points to use for camera pose estimation and fitness calculation in the next steps.

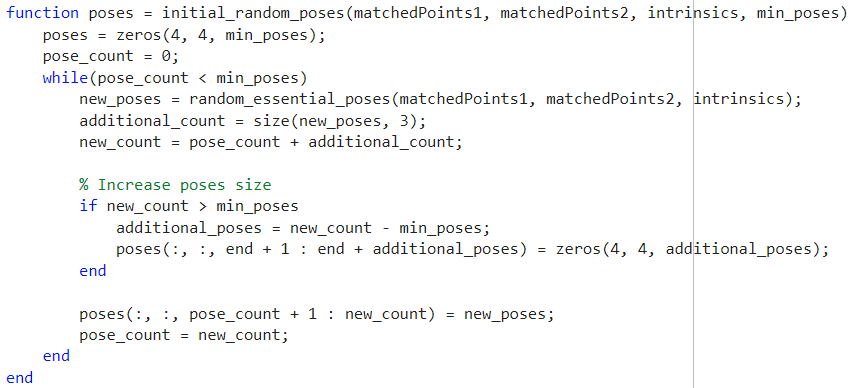
A screenshot of a computer code

Description automatically generated

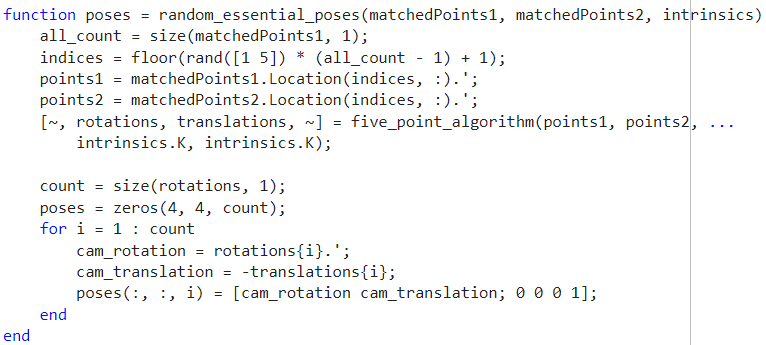
## Initial Camera Pose Estimations

We use the five point algorithm to estimate the essential matrix and extracting the rotations and translations from it according the known intrinsic parameters of the camera. For this purpose, we used the implementation by SergioRAgostinho.

We need to find certain amount of candidate solutions as the initial population. To do so, we repeat the five-point algorithm until enough amount of solutions are estimated.

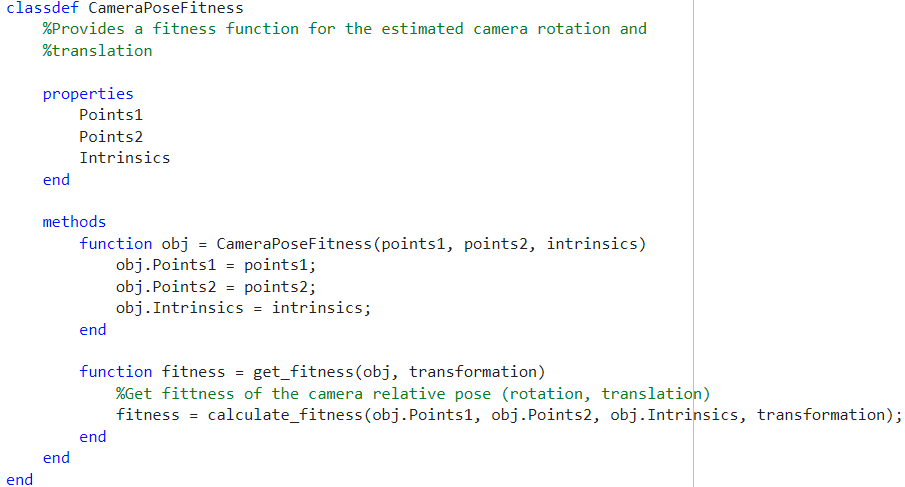


Each time we want to estimate the essential matrix, rotations, and translations, we use 5 random feature points from the found feature points. This allows the solutions to be randomized and valid.

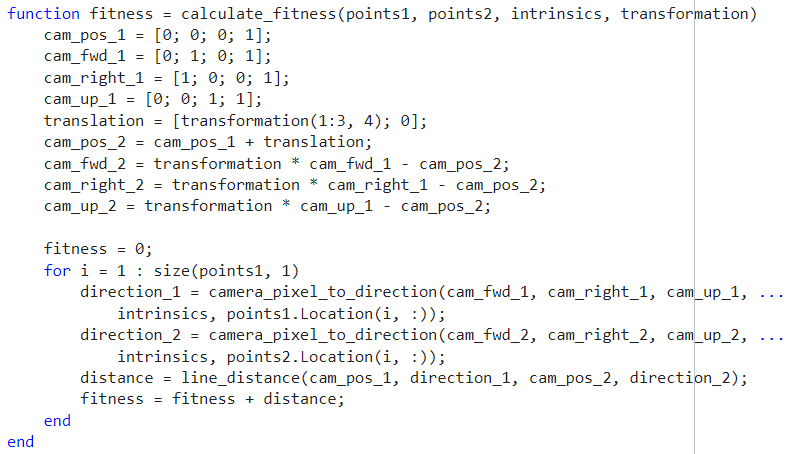


## Fitness Calculation

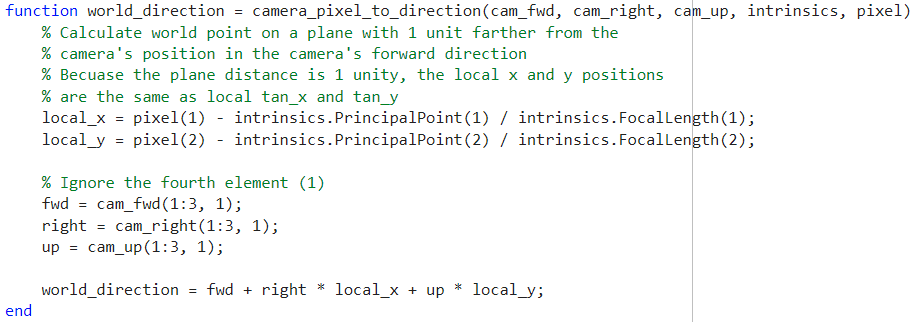
Because the fitness calculation involves using more data than only the input position (camera relative pose) we define a class to hold those data and then we call its calculation method. The data we need, are the knowledge we have about the problem. We need the matched points in both images and the intrinsic parameters of the cameras. The *get\_fitness* method, gets the input solution (transformation from first camera to the second camera) and sends it to the calculation method alongside the other data.



The *calculate\_fitness* method is responsible calculate a fitness for the current solution. It considers the first camera at the origin of the coordination directing toward the front (y) axis. It then applies the rotation and translation of the input solution on the first camera to get the orientation of the second camera. Here, we calculated the position, forward, right, and up vectors of the each camera to define its orientation and use them in next calculations. Then, for each match point in both images, we calculate the rays emitting from the cameras toward the world position of those pixels. Then we calculate the distance between those rays(lines). The sum of the distances will be the fitness we return.



The method *camera\_pixel\_to\_direction* calculates the ray according to the orientation of the camera and the chosen pixel point.



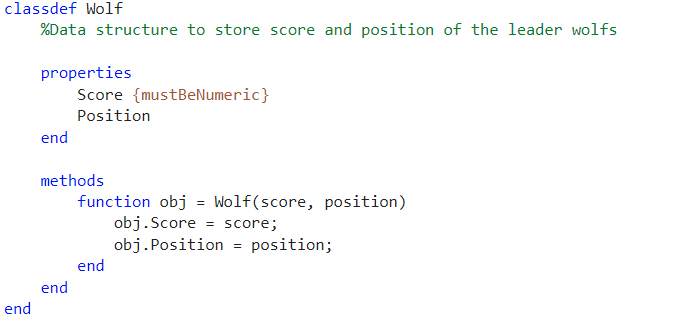
The *line\_distance* calculates the closest distance between the two rays.

A computer code with black text

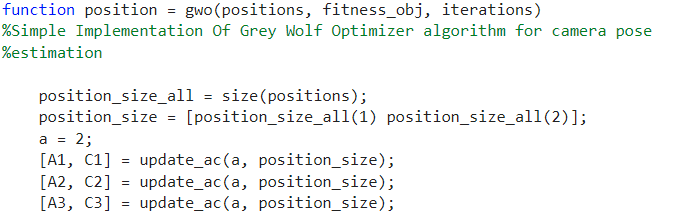
Description automatically generated

## Grey Wolf Optimizer

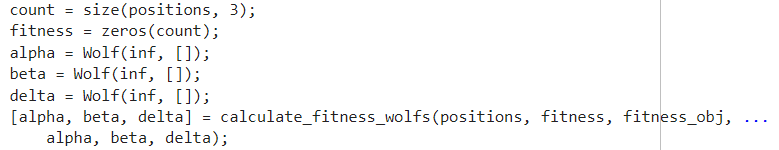
First, we define class structure for leader wolves (alpha, beta, and delta) to hold the score and their current position (which is the transformation matrix).



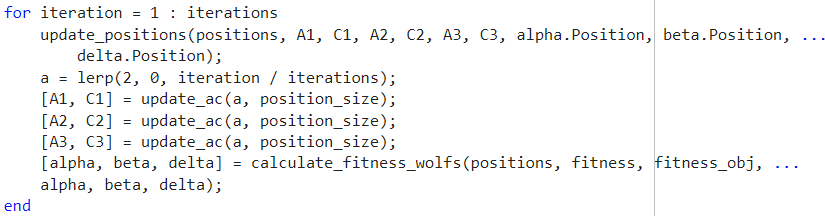
The function *gwo* is responsible for running the Grey Wolf Optimizer algorithm. It first updates the *a, A1, C1, A2, C2, A3, C3* coefficients.



Then, it calculates the fitness of all the wolves and choose the best 3 as the alpha, beta, and delta wolves.



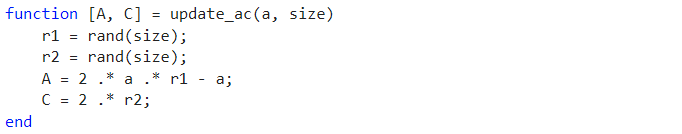
Now, it runs the iterations. In each iteration, it first updates the positions of the wolve according to the alpha, beta, and delta positions. Then, it decreases the *a* coefficient linearly and updates the *A* and *C* coefficient according to the new value of *a*. Finally, it calculates the fitness of the wolves and chooses the best 3 ones again. It repeats these until maximum iteration.



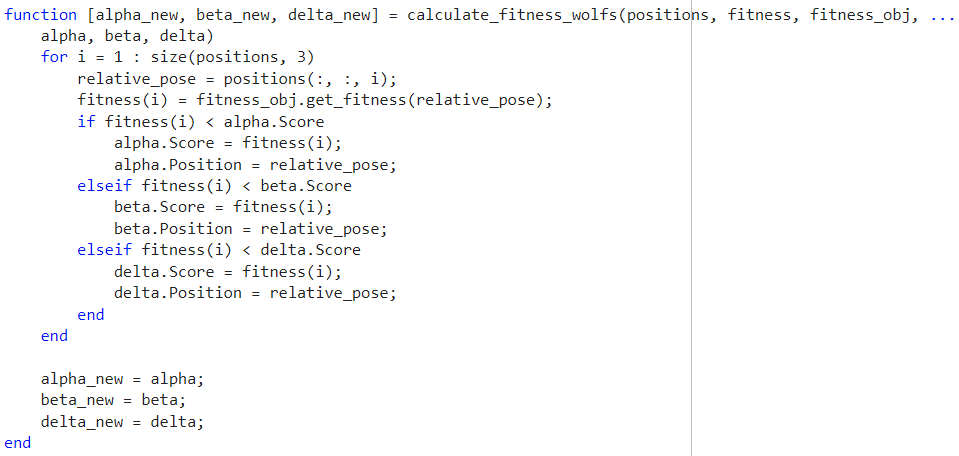
It then returns the alpha as the best solution.



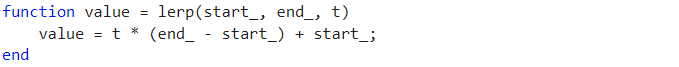
This method is responsible to update the *A* and *C* coefficients according to the *a* and random *r1* and *r2* coefficients.



This method calculates the fitness of the wolves according to the fitness function provided and then chooses the best 3 as the alpha, beta, and delta wolves.



This method linearly interpolates the input value. We used it for decreasing *a* coefficient.



This method is responsible for updating the positions of the wolves according to the alpha, beta, and delta wolves.

A computer screen shot of a code

Description automatically generated

## Calling the estimation

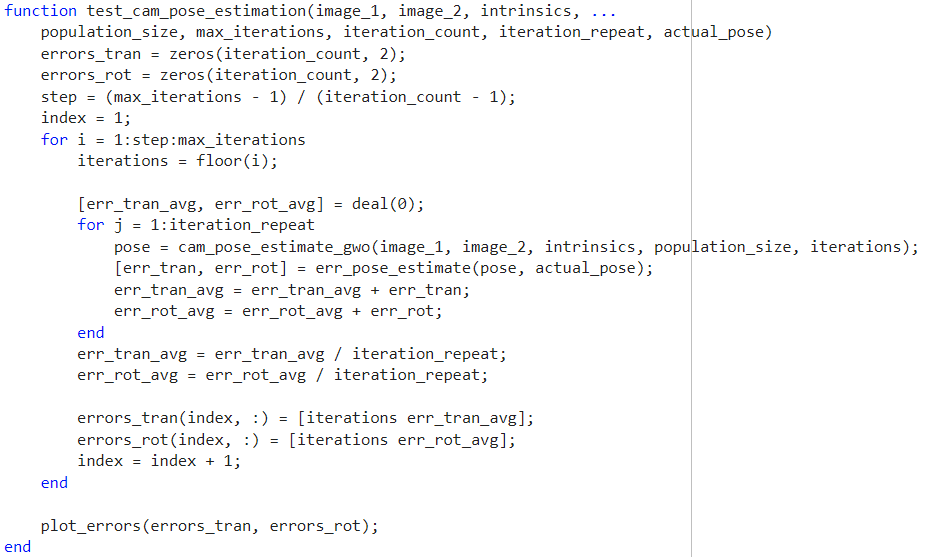
We, send the estimations as the initial population alongside the fitness to the GWO algorithm to find the optimum solution. The position of the returned alpha is the best solution value.

A close-up of a computer code

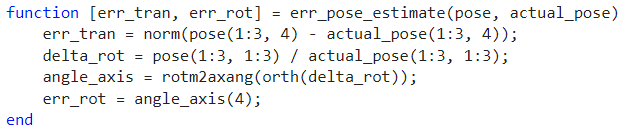
Description automatically generated

## Testing the estimation

We define the *test\_cam\_pose\_estimation* function to estimate the relative pose for different amount of iterations and compare them with the actual relative pose. It then plots them to be able to compare the effect of the iteration count on the algorithm. For each iteration count, we might repeat it certain times and get the average error as the reported error.



The *error\_pose\_estimate* calculates the error between the estimated pose and the actual pose.



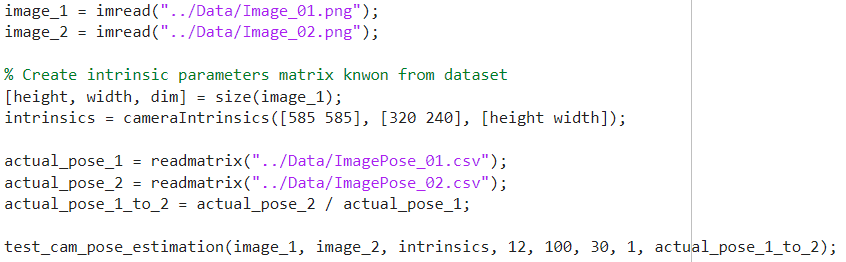
The *plot\_errors* function plot the found errors to display them.

A computer screen shot of error code

Description automatically generated

Note that we use different plots for rotation and translation errors to analyze them separately.

We load the 2 images and their corresponding poses to test the algorithm.



# Dataset

We used 2 images captured by one camera and their corresponding poses from the office images of the (7 scenes data set, n.d.) dataset.

# Results

# Conclusion

We saw that the camera pose estimation for 2 uncalibrated cameras with 2 images can be further improved by applying the Grey Wolf Optimizer on the random initial solutions and finding the optimum solution.

# References

Elashry, A. and C. Toth (2023). "IMPROVING CAMERA POSE ESTIMATION USING SWARM PARTICLE ALGORITHMS." Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. **XLVIII-M-3-2023**: 87-93.

Mirjalili, S., et al. (2014). "Grey wolf optimizer." Advances in engineering software 69: 46-61.

Nister, D. (2004). "An efficient solution to the five-point relative pose problem." IEEE Transactions on Pattern Analysis and Machine Intelligence 26(6): 756-770.

7 scenes data set. (n.d.). Retrieved from https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/

SergioRAgostinho. "Five-Point Algorithm." from https://github.com/SergioRAgostinho/five\_point\_algorithm.