OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

INF

FAKULTÄT FÜR
INFORMATIK

Intelligent Cooperating Systems
Computational Intelligence

2D-Contour Search using a Particle Swarm Optimization inspired Algorithm

Master Thesis

Dominik Weikert

April 25, 2019

Supervisor: Prof. Dr. Sanaz Mostaghim

Advisor: Prof. Dr. Sanaz Mostaghim

Dominik Weikert: *2D-Contour Search using a Particle Swarm Optimization inspired Algorithm*
Otto-von-Guericke University
Intelligent Cooperating Systems
Computational Intelligence
Magdeburg, 2019.

Abstract

This thesis presents Particle Swarm Contour Search: a Particle Swarm Optimization inspired algorithm to find object contours in 2D environments. Currently, most contour-finding technologies are based on image-recognition algorithms, which require a complete overview of the search space in which the contour is to be found. However, for real-world applications this would require a complete aerial or even satellite-based imaging, which may not always be feasible or possible. The proposed algorithm removes this requirement as only the local information of the particles is needed to accurately identify a contour. Particles search for the edge of the object and then travel alongside its contour using their last known information about positions in- and outside of the object. With regard to swarm robotics, this could enable a swarm of robots to collectively identify object boundaries in the environment using internal sensors without any need for additional imaging technologies. The performed experiments show that the algorithm works, with the performance lying within an order of magnitude of an image-processing based algorithm. In summary, the Particle Swarm Contour Search algorithm showed promising results suitable for further research and development.

Contents

List of Figures

List of Tables

List of Acronyms

PSO Particle Swarm Optimization

SPSO (2011) Standard Particle Swarm Optimization [23]

CPSO Charged Particle Swarm Optimization [4]

MNT Moore-Neighborhood-Tracing

1 Introduction

1.1 Motivation

Contour search has been the subject of research for multiple decades, with most work being done based on image-processing technologies, requiring complete overview of the search space to apply mathematical operators to every pixel. In some real-world applications however, such as the tracking of oil spills or wildfires, a complete real-time overview using aerial or satellite imaging is infeasible. Additionally, some hazards such as radiation or chemical spills and vapours might not be detectable via imaging technologies and only via a local sensor measurement.
This sensor measurement could be provided by an autonomous robot, with advances in swarm intelligence and robotics providing an interesting alternative to image-based technologies:
A swarm of agents able to correctly trace the contour of any object in the environment using only internal sensors could potentially provide real-time spread analysis in emergency situations as well as initiate actions to alleviate the situation. Swarm algorithms have already been successfully applied to problems such as collective search [8] [14] and obstacle avoidance [20].

The main difficulty applying a swarm algorithm such as Particle Swarm Optimization to the problem of contour search is that they are mostly suited to numeric optimization and their performance dependent on the existence of a non-zero gradient in a monotonic search space. In contour search, existence of a gradient can not be guaranteed, with a search space consisting of infinite local minima or maxima and non-monotonic function value jumps from one to the other. As such, the gradient will be zero for vast regions of the search space. In such an environment, any numeric optimization based swarm algorithm may quickly devolve into a random search. This thesis aims to adapt existing Particle Swarm Optimization principles to search spaces frequently encountered in contour search.

1.2 Goals

Contour search using a swarm-based algorithm

The main goal of this thesis is to create a generic swarm algorithm capable of finding the contour of a 2D object. The algorithm should be capable of finding and outlining the contour of a given 2D shape.

Evaluation under real-wolrd constraints

As the motivation is to create a swarm algorithm applicable in swarm robotics, the performance under real-world swarm-robotics constraints such as sensor noise must be evaluated to establish a basis for object tracking using a swarm of robots.

Comparison with image processing

Image processing is the current state-of-the art of contour search. As such, the results yielded by the algorithm are to be compared to the output of an image-processing based contour search.

1.3 Outline

In chapter two the basics of Particle Swarm Optimization (PSO) as the inspiration for this thesis is presented. This includes a basic background in PSO as well as a description of the specific PSO algorithm adapted to the contour search problem. Additionally the state of the art in image-processing based contour search techniques will be outlined. The proposed algorithm is then detailed in chapter three. Chapter four encompasses the experiments done to evaluate the algorithm and a discussion of the results. Finally chapter five concludes this thesis with an analysis of the usability of the algorithm and potential for further research.

2 Related Work

In this section, the background of Particle Swarm Optimization as an inspiration for the developed algorithm is described and the basic concepts outlined. Additionally, a basic understanding of the state of the art in image-processing based methods of contour search will be given.

2.1 Particle Swarm Optimization

Particle Swarm Optimization is a nature-inspired paradigm of swarm intelligence. The social behaviour in natural swarms, such as schools of fish or flocks of birds, is mimicked to solve global optimization problems.

In Particle Swarm Optimization, each particle represents one member of the swarm moving through the search space, searching for the minimum. This concept was first introduced by Kennedy and Eberhardt in 1995 [11], after studying the simulations of zoologists Heppner and Grenader about bird flocking [9].
Initially, each agent in the simulation only copied the velocity of its nearest neighbour. While this led to a synchronous movement of the swarm, said swarm also quickly adapted an unchanging direction and speed. In order to combat this completely uniform behaviour, each iteration a random change was done to the chosen velocities of each particle. This was called "craziness" and served to give the simulation more "lifelike" qualities.
As a further improvement of the simulation, and to study birds' apparent ability to find food based on the knowledge of other flock members, a vector serving as the memory of each agent was introduced. In this vector, the the best position and the value of that position were stored. A second vector was used to memorize the global best position any agent in the swarm had found.

The personal best is called the cognitive experience and the experience exchanged across the swarm is referred to as social experience.
In the resulting model, each agent was attracted to both its personal best and the global best of the swarm, which removed the need for both craziness and velocity matching. This model was first referred to as Particle Swarm Optimization (PSO).
The PSO concept has since been adapted and improved many times for numerous applications [15] [17].
Any PSO algorithm maintains a swarm, or population, of particles. Each of these particles can measure the value of the optimization function at its current position and thus represents a potential solution in the search space. Each iteration, the position of these particles is adjusted according to its own experience and that of the swarm - the particles are 'flown' through the search space.
Each particle i assigned a position $\vec{x}_i(t)$ and velocity $\vec{v}_i(t)$ at any given time step t. Additionally, the particle possesses a memory of its own best position, denoted $\vec{y}_i(t)$ and called local best, and the best position ever attained by the swarm, the so-called global best, $\vec{y}_g(t)$. The difference between the position and the global best, $\vec{y}_g(t) - \vec{x}_i(t)$ is also called the social component $\vec{v}_{i_s}(t)$, while the difference in position and local best $\vec{y}_i(t) - \vec{x}_i(t)$ is also called the cognitive component $\vec{v}_{i_c}(t)$.

The position of each particle, and thus the entire swarm, is then adjusted in each time-step t according to the following equations:

$$\vec{v}_i(t+1) = \omega\vec{v}_i(t+1) + c_1\vec{r}_1 \otimes \vec{v}_{i_c}(t) + c_2\vec{r}_2 \otimes \vec{v}_{i_s}(t) \tag{2.1}$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1). \tag{2.2}$$

The parameters r_1 and r_2 represent vectors of uniformly distributed random variables in the range of $[0, 1]$. These are included to implement some randomness in the direction of the velocity, increasing the diversity of the particle population and are combined with the individual velocity components via element-wise multiplication $\otimes$. ω is called inertia weight and is used to control the influence of previous speeds of the particle. The influence of the particles own experience and that of the swarm are scaled by c_1 and c_2, respectively.

Equation 2.1 consists of three basic terms, which will be described in the following:

The previous velocity $\omega\vec{v}_i(t)$ is simply a memory of the previous flight direction. In physical terms it serves as momentum and prevents the particle from drastic direction changes. With higher inertia weights, more bias is given towards the current direction.

The cognitive component $c_1r_1\big(\vec{y}_i(t) - \vec{x}_i(t)\big)$ serves as an attractor towards the previous best position the individual particle has attained - the individual 'wishes' to return to a place it has previously found satisfactory.

The social component $c_1r_1\big(\vec{y}_i(t) - \vec{x}_i(t)\big)$ evaluates the current performance of the particle in relation to the entire swarm. In effect, this draws each particle towards the best position found by the swarm. Over time, this causes the swarm to converge to an optimum.

2.1.1 Charged Particle Swarm Optimization

One of the drawbacks of the basic PSO model is its inability to perform well in dynamic or multi-modal environments. Once the swarm is converged to a perceived optimum, it is very difficult to adapt to changes in the optimized function. For multi-modal problems, and especially contour search, the swarm has difficulty navigating due to the monotony of the search space.

To improve the adaptability of PSO algorithms to changes in the environment, Blackwell and Pentley proposed a new algorithm based on charged swarms called charged PSO (CPSO) [4]. To prevent complete convergence to the global best position, some particles are assigned a charge. All charged particles experience a repulsive force from all other charged particles, while the neutral particles behave as normal. This allows neutral particles to exploit the global optimum, while the charged particles continue to explore around the optimum due to the repulsive forces.

$$\vec{a}_i(t) = \sum_{j \neq i} \frac{Q_i Q_j}{(|\vec{x}_i(t) - \vec{x}_j(t))|^3} \left(\vec{x}_i(t) - \vec{x}_j(t)\right) \tag{2.3}$$

The new acceleration introduced by the charge is shown in Equation 2.3. Every particle i is assigned a charge of magnitude Q_i, with neutral particles being assigned a charge of $Q_i = 0$. The complete velocity update then changes as shown in Equation 2.4

$$\vec{v}_i(t+1) = \omega \vec{v}_i(t) + c_1 \vec{r}_1 \otimes \vec{v}_{i_s} + c_2 \vec{r}_2 \otimes \vec{v}_{i_c} + \vec{a}_i(t) \tag{2.4}$$

In their experiments, Blackwell and Pentley were able to conclude that a mixed swarm of charged and neutral particles - also called an atomic swarm - outperformed a swarm consisting of only charged particles. Both versions were able to consistently outperform the non-charged PSO algorithm in a dynamic environment. Since then, more research has been conducted concerning the application of PSO to dynamic environments [3] [2] [12].

2.1.2 Standard Particle Swarm Optimization

In 2011, an improved PSO algorithm, Standard Particle Swarm Optimization 2011 (SPSO 2011) was proposed by Zambrano-Bigiarini et al. [23]. This algorithm is regularly used as a reference point for comparison with other optimization algorithms, and yields improved performance on multi-modal functions.

$$\vec{p}_i(t) = \vec{x}_i(t) + c_1 \vec{r}_1 \otimes \vec{v}_{i_c}(t) \tag{2.5}$$

$$\vec{l}_i(t) = \vec{x}_i(t) + c_2 \vec{r}_2 \otimes \vec{v}_{i_s}(t) \tag{2.6}$$

$$\vec{g}_i(t) = \frac{\vec{x}_i(t) + \vec{p}_i(t) + \vec{l}_i(t)}{3} \tag{2.7}$$

$$\vec{v}_i(t+1) = \omega \vec{v}_i(t) + \mathcal{H}_i(\vec{g}_i(t), ||\vec{g}_i(t) - \vec{x}_i(t)||) \tag{2.8}$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \tag{2.9}$$

Instead of the velocity calculation given in equation 2.1, the velocity is chosen randomly from a hypersphere constructed from the social and cognitive component using equations 2.5 to 2.9. In their experiments, Zambrano-Bigiarini et al. found that SPSO 2011 achieved good performance on multi-modal and multi-dimensional (up to 50) functions with fast convergence, while avoiding stagnation, where all particles converge towards an optimum.

2.2 Edge and contour detection in image processing

Edges in image processing are usually defined as maxima of the gradient value of an image. These maxima represent large changes in pixel values and are thus indicative of object boundaries. Object contours are therefore continuous edges forming a closed loop, e.g. they fulfill the additional constraint that their first and last points are identical.
Several varying methods exist to identify edges in an image, based on two main methods: Gradient based edge detection, which directly searches for gradient maxima and laplacian based edge detection, which relies on on detecting zero points in the second derivative of the image. Much research has been conducted has been conducted into comparing and improving performance of edge detection algorithms [1] [6] [22] [16], but the underlying technique of using kernels to smooth and obtain the image gradient described in this chapter remain largely the same.

2.2.1 Kernel Convolution

Kernel convolution is a technique used in image processing for effects such as blurring and sharpening as well as edge detection. A kernel is a convolution matrix, or mask, that is applied to every pixel of the image. During convolution, the pixel value is added with its neighbours according to the weight provided by the kernel. This sum is then normalized by the total sum of coefficients of the kernel as to prevent brightening or darkening of the image.

The basic principle is shown in pseudo-code in algorithm 1. Examples for simple kernels are given in figure 2.2. If a pixel would be outside of the image border, several methods can be used to get an appropriate pixel value: E.g the nearest pixels could extended to match the size of the kernel, or the image can be wrapped around, with values being taken from the opposite border.

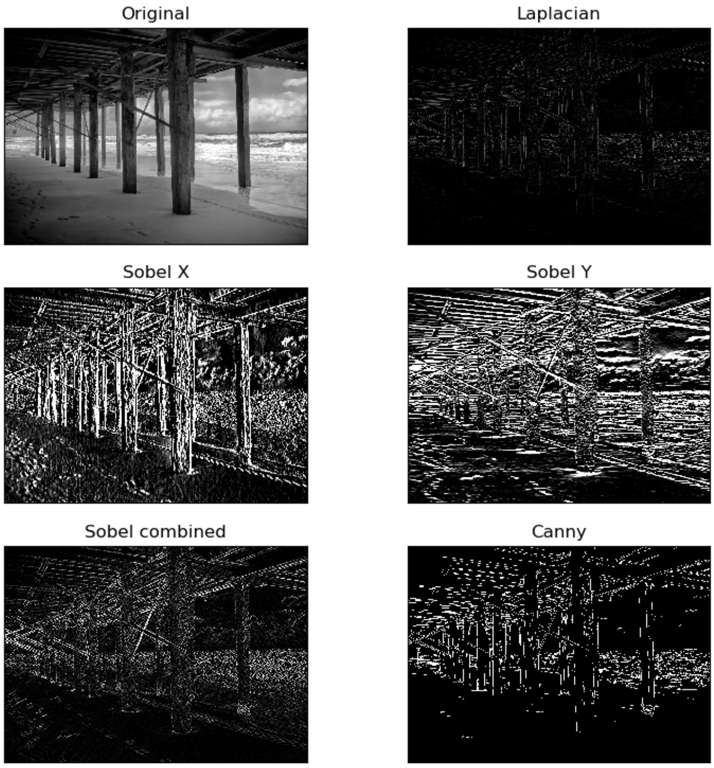


Figure 2.1: Output of edge detection algorithms included in the OpenCV library [5]

$$K_{ID} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

a) Identitiy kernel

$$MBlur = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

b) Box blur kernel

Figure 2.2: Examples of basic kernels: Subfigure a) shows the basic identity kernel, in which only the pixel retains its value and all others will receive the weight of 0 in the summation. Subfigure b) shows a basic box blur kernel which simply averages all pixel values in a $3x3$ grid

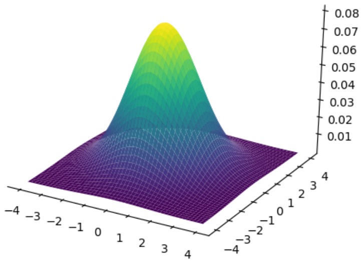
Algorithm 1 Kernel convolution principle

```
Input: image f(x,y), kernel k(x,y), output_image g(x,y)
\\ kernel has dimensions [−w,w] and [−h,h], sum of coefficients S_k

for y = 0 to image.rows-1 do
  for x = 0 to image.columns-1 do
    sum = 0
    for i=-h to h do
      for j= -w to w do
        \ \ sum up pixel values overlapped by kernel
        sum+ = k(i,j) ∗ f(x − j, y − j)
      end for
    end for
    \ \ normalize before output to prevent brightening/darkening
    g(x,y) = sum / S_k
  end for
end for
```

This kernel convolution technique can be used to detect edges by using symmetrical kernels with opposing sides being assigned different signs and the axis of symmetry being assigned 0. When this kernel overlays an edge, the corresponding pixels will have significantly different magnitudes and thus the convolution will result in a high value. Contrary, if the kernel overlays a smooth region without large changes in the pixel values, both sides will cancel each other out and result in a near-zero value. Edge detection kernels are usually used in combination with smoothing kernels to remove noise from the image and prevent false edge detections. The following section describes some frequently used kernels for edge detection and image smoothing.

The sobel operator is an example of such a kernel used for edge detection in images. Fig. 2.4 shows three example kernels. To get the complete gradient two different (rotated) operators are applied for each axis, resulting in two distinct gradient measurements. These can be combined to get the overall magnitude of the gradient by normalizing the vector given by both gradient values $\vec{g} = (g_x, g_y)$.



a) 2-D Gaussian distribution with mean (0,0) and $\sigma = 1.4$

$$G = \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 4 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

b)Gaussian kernel for filtering out noise by spreading pixel values

Figure 2.3: Gaussian distribution and gaussian kernel

The Laplacian operator is another technique used for edge detection. It is based on the Laplacian of the image, $\Delta f(x,y) = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2}$. By reducing the operator to the discrete case in image processing, it can be applied via a single 2-D convolution kernel. No separation for the x and y dimension is necessary because in a discrete 2D case the Laplace operator includes the differences in both dimensions. Commonly used kernels are shown in Fig. 2.5. A disadvantage of this kernel is that it is highly sensitive to noise.

The Gaussian Filter is a widely used filter for smoothing or blurring images and to get rid of noise. A kernel can be constructed by sampling the 2D Gaussian function $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$. This function can be integrated over each kernel pixel to obtain discrete kernel values. A visualization of this function is shown in Fig. 2.3, along with a suitable kernel.

The Laplacian of Gaussian is a kernel convolution used to mitigate the aforementioned sensitivity of the Laplacian operator to image noise. The Gaussian function is combined with the Laplacian operator to reduce the impact noise has on a purely Laplacian Kernel.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

a) Sobel operator for x-direction

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

b) Sobel operator for y-direction

$$S_x = \begin{pmatrix} -5 & -4 & 0 & 4 & 5 \\ -8 & -10 & 0 & 10 & 8 \\ -10 & -20 & 0 & 20 & 10 \\ -8 & -10 & 0 & 10 & 8 \\ -5 & -4 & 0 & 4 & 5 \end{pmatrix}$$

c) (5×5) Sobel operator for x-direction

Figure 2.4: Kernel operator overview: Kernels used for gradient detection in images, applied for each the x-axis and the y-axis using kernel a) and b), respectively, with a larger (5×5) example for the x-direction in c)

$$L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

a) Laplacian kernel

$$L = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

b) Laplacian including diagonals

Figure 2.5: Laplacian kernels used for second-order gradient detection in images. The high noise sensitivity stems from the high value of the centre pixel, which could vary greatly from its surroundings due to noise and therefore lead to false edge detection. [13]

2.2.2 Canny Edge Detection

In 1986, John Canny proposed a new approach to edge detection called the Canny Detector [7]. The Canny algorithm was developed to satisfy three specific criteria with regards to the detection of edges:

Low error rate, specifically a good detection of only existent edges even in noisy images.

Good localization minimizing the distance of any detected edge to the true edge.

Minimal response to edges: each true edge in the image only results in one detected edge by the algorithm.

To achieve these goals, the algorithm is divided into four steps:

1. Noise filtering
2. Calculating intensity gradient
3. Edge thinning
4. Hysteresis thresholding

The noise filtering step can be achieved via any noise filtering kernel, such as the Gaussian kernel described in the previous section.
To calculate the intensity gradient, the blurred image can be convoluted with an edge-detection kernel, such as the Sobel or Laplacian operator.
For the third step, all pixel values are compared with their neighbourhood in the direction of the edge to suppress all pixels that are not a local maximum, reducing edge width.
Finally, hysteresis thresholding is applied to further filter out weak edges. Hysteresis thresholding is a dual-threshold approach in which all values above the top threshold are kept and all below the bottom threshold are discarded. Values between the two thresholds are kept only if they are connected to a pixel that has already been accepted by the algorithm. This allows connected contours to be found, even though some of the gradient intensities might be below the upper threshold.

2.2.3 Adaptive contour models

Contours differ from edges since they are supposed to be continuous lines that signify an object boundary. Edge detection can be combined with adaptive contour models[10], also called snakes, to detect and follow contours. Active contour models are spline curves parametrized by a variable $p \in (0, 1)$ and two functions $x(p)$ and $y(p))$ representing the coordinates of the points along the curve. The vector $\vec{v}(p) = (\vec{x}(p), \vec{y}(p))$. is used to denote the spline. In case of an object boundary, the curves endpoints need to be identical with $\vec{v}(0) = \vec{v}(1)$. To track object contours, splines are moved throughout the image to minimize a modelled energy function shown in equation 2.10. This function consists of the internal energy of the snake E_{snake}, which is dependant on its curvature and used to smooth out the curve, and the external force derived from the image gradient E_{image}, which minimizes the distance to edges in the image.

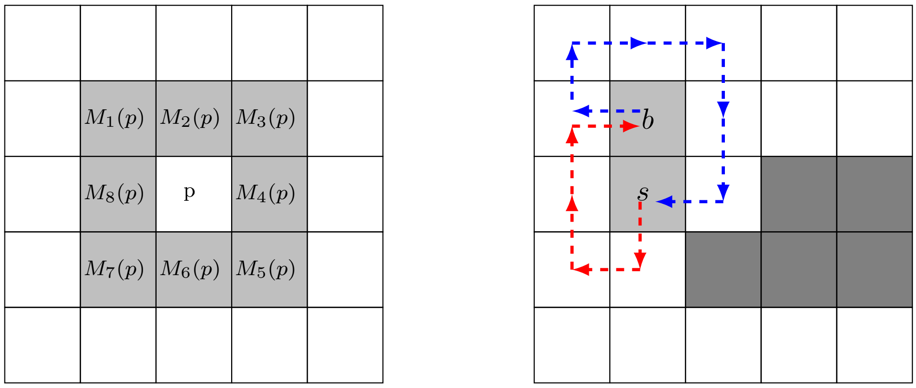
$$E = \int_0^1 E_{snake}(v(p)) + E_{image}(v(p))dp \tag{2.10}$$

This function can be minimized using the Euler-Lagrange equation, resulting in equation 2.11, which can then be solved using a minimization technique such as gradient descent, pushing the snake towards contours in the image.

$$\vec{v}(p'') + \vec{v}(p'''') + \nabla E_{image} = 0 \tag{2.11}$$

2.2.4 Pixel-Following methods

For static contours in binary images, a simpler approach called pixel-following may be applied. With this method, a contour is traced by sequentially searching the direct neighbourhood of a pixel for another black pixel using a relative order, e.g starting clockwise from the pixel to the left of the current pixel. This search is then repeated with the found pixel, forming a chain of pixels until the contour is complete.
An example of this is the Moore-Neighbour-Tracing (MNT), which searches the Moore-Neighbourhood of a pixel p, denoted $M(p)$. The Moore-Neighbourhood of a pixel is shown in fig. 2.6a and a simplified algorithm using this method



(a) An example of the Moore-Neighborhood of a pixel p

(b) Example contour for which base MNT fails: starting from s, b is found(red path) and then s again(blue path)

Figure 2.6: Moore-Neighbourhood and contour example

shown in pseudo-code in algorithm 2.
The algorithm starts by tracing each pixel of the image, e.g column-by-column from the bottom row to the top row, until a black pixel is encountered. This pixel is then set as the start pixel s and the boundary pixel b. Then, the current pixel c is "backtracked", i.e. it is set to the pixel from which b was entered from. From this step, each pixel of the Moore-Neighbourhood $M(b)$ of b is examined until a black pixel is found. This pixel then becomes the new boundary pixel b and the process is repeated until the start pixel is encountered again.

A disadvantage of the MNT algorithm is the stop condition, which terminates the algorithm as soon as the starting pixel is encountered again, which causes it to fail some patterns. An example of such a pattern is shown in fig. 2.6b To avoid this, the condition can be expanded so that the start pixel has to be entered in the same direction as it originally was.
The MNT algorithm can be expanded to find multiple contours with topological information, such as the algorithm used by the OpenCV framework [21]. An improved pixel-following algorithm was proposed by Seo et al. [19], including an overview and comparison of similar methods.

Algorithm 2 Moore-Neighbour tracing algorithm

```
\\ Input: binary image I
\\ Output: list L of contour pixels
Scan image pixels until current pixel c is black
append c to C, set boundary start s = c and current boundary point b = c
backtrack c
while c ≠ s do
  if c is black then
    append c to L
    set b = c
    backtrack c
  else
    c = next pixel in M(b)
  end if
end while
```

3 Particle Swarm Contour Search

The algorithm created for this thesis, Particle Swarm Contour Search, has the goal of accurately finding the contour of one or more objects in a multi-modal search space, such as e.g. a black surface in a grey-scale image. The idea is to use a Particle Swarm Optimizer adapted to this environment to continuously trace the contour of a found object. To overcome the inherent weaknesses of PSO in this environment, the basic velocity update must be modified to put less importance on the existence of a global optimum.

3.1 Problem Description

The most basic problem for contour search would be the finding of a convex black object within a white background image. As most problems of contour search within the workings of the proposed algorithm can be reduced to a sensor or function value falling within a certain range (e.g substance concentration for chemical/oil spills, heat for fire), and can thus be compared to this basic problem, the image-based contour search will be used as a general problem reference in the following, with the function value $f(x)$ ready by the sensor referred to as the colour value. Additionally, it will be assumed that the size search space is known, as it would usually be in a real-world search application.

All particles move along the pixels of the image and can read the colour value of their current pixel. A threshold colour value is given, at which the particle is considered inside the object. This is analogous to a sensor value such as chemical concentration or heat rising above a pre-set level. The goal for each particle is then to find the object and move along its contour, using its own information and the information gleaned form its swarm. As a secondary goal of this thesis is to establish a basic algorithm to improve upon for swarm robotic applications, the colour of the current position as well as the actual position of the particle itself will be subject to measurement errors.

Increasing problem difficulty can be simulated using more complex and non-continuous shapes with blurred edges, several example landscapes are shown in fig. 3.1.

3.2 Overview

The PSCS algorithm consists of three distinct phases with different velocity update mechanisms. Initially, all particles conduct a global search using a modified CPSO approach. Once a particle detects an object, a new sub-swarm is created from its neighbourhood to explore the contour of that object. If a particle exploring an object detects that it is close to an edge, it enters a contour following phase. A visualization of the overall flow of the algorithm is shown in fig. 3.2.

Algorithm 3 shows a pseudo-code description of the initialization and overall structure of the algorithm. More detailed algorithm descriptions are given for each phase individually in the following sections. The input for the algorithm consists of the evaluated function $f(x, y)$, which represents the measured sensor value at each given position, the population size P, the object detection threshold θ and the minimum contour distance ϵ. The initialization sets all particle positions and velocities, creating a global swarm. The global swarm performs a search for any object in the search space. Once an object is found, a sub-swarm is created form particles in the neighbourhood, which then performs a more localized contour search phase. Once velocity updates for individual particles in these sub-swarms become sufficiently small, a contour tracing phase is initialized.

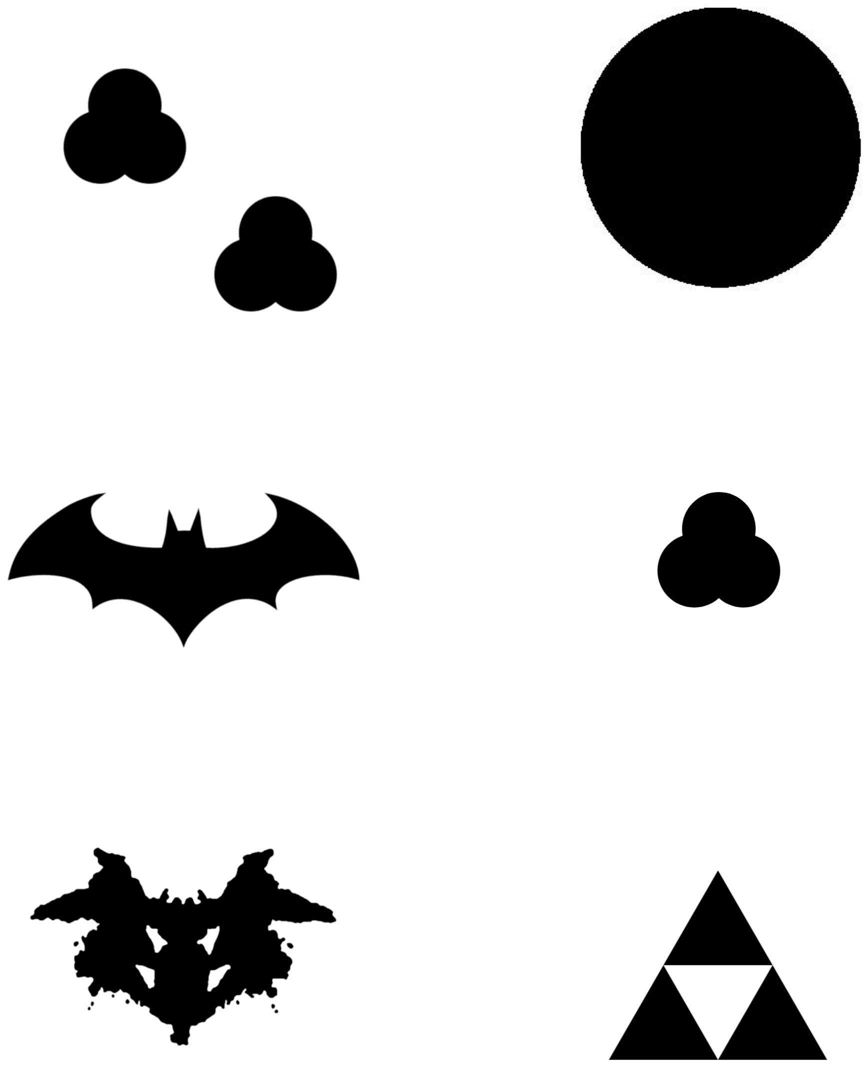


Figure 3.1: Example landscapes for the contour search. Batman image taken from: http://getdrawings.com/batman-symbol-vector, license: CC, last accessed 25.04.2019

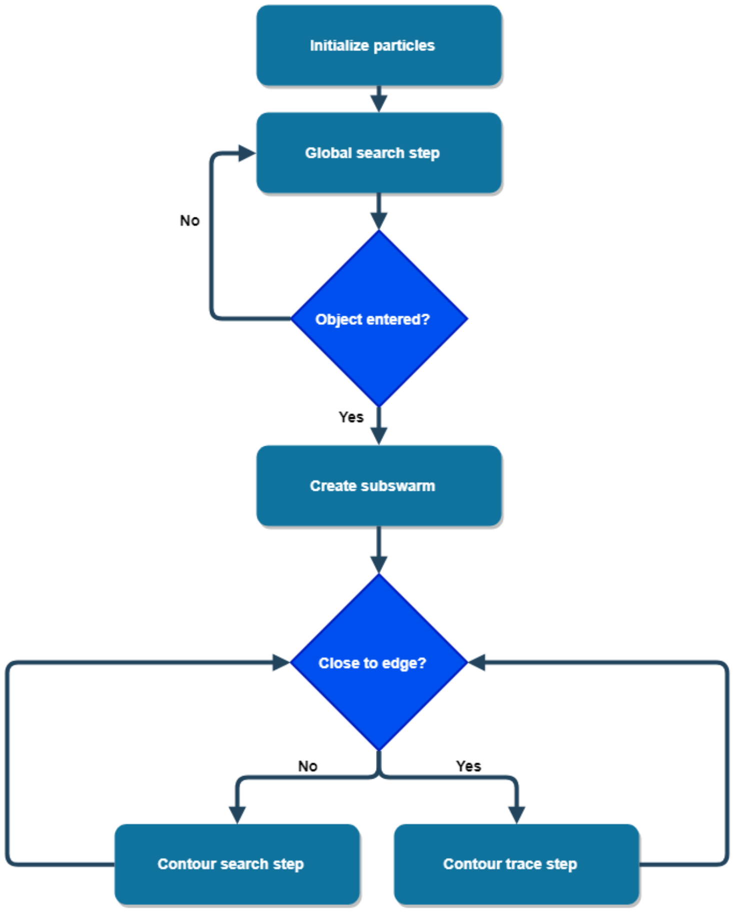


Figure 3.2: Flowchart of the PSCS algorithm phases. The stopping condition, which is checked individually after each phase, is omitted for increased clarity

Algorithm 3 Baseline PSCS algorithm

Input: $f(x, y)$, P, θ, ϵ
Initialize particles in global swarm evenly spaced across the image:
for $i = 0$ to $P - 1$ **do**
  $\vec{x}_i(t) = (\delta_x * i, 0)$
  $\vec{v}_i(t) = (0, V_{START})$
  $S_g.append(p_i)$
end for
while stop condition not met **do**
  for each particle i in S_g **do**
    perform global search update
  end for
  for each sub-swarm S_n **do**
    for each particle i in S_n **do**
      if trace condition met **then**
        perform trace update
      else
        perform search update
      end if
    end for
  end for
end while

Algorithm 4 PSCS global search phase

for each particle i in S_g **do**
  \\ search phase update for the global swarm
  if $c_i(t) < \theta$ **then**
    $\vec{in}_i(t) = \vec{x}_i(t)$
    createSubSwarm(P_S)
  else
    $\vec{out}_i(t) = \vec{x}_i(t)$
    $\vec{v}_i(t + 1) = updateVelocityGobal(\vec{v}_i(t), \vec{x}_i(t))$
    $\vec{x}_i(t + 1) = updatePosition(\vec{v}_i(t), \vec{x}_i(t))$
  end if
end for

3.2.1 Initialization

In this step the particles and their variables are set up. If the size of the search space is known, the particles can be spread evenly along an edge of the search space with a constant velocity V_{START} away from that edge, with $delta_x$ denoting the distance between the particles in the x direction. If the size is unknown, they can simply be initialized with random positions and velocities to ensure non-biased exploration of the search space. With more knowledge of the search space or the object contour, specific patterns could be applied to improve algorithm exploration and diversity.
A threshold value θ for the detection of objects needs to be set. This represents the sensor value for which the particle is considered to be inside the object. This could be chemical concentration, temperature, or radiation count. For the image application, it simply represents the a colour value. The $\vec{in}$ and $\vec{out}$ values of each particle are initialized to its current position and all particles are added to the global swarm, which then begins the object search phase.

3.2.2 Object search phase

This is the update phase for the global swarm S_g including all particles that have not yet been assigned to a sub-swarm. Initially this swarm will consist of all particles and eventually be reduced to 0 as more sub-swarms are created to track object contours. The particles in this swarm use a modified CPSO approach shown in equations 3.1 and 3.2 for their velocity update, with only members in sub-swarms being considered for the calculation of the repulsive force. This leads to a repulsive force acting on the global swarm from all sub-swarms, preventing the global swarm from repeatedly creating sub-swarms to explore the same object. This repulsion is decreased over time to allow all particles to trace the contour if no other object is found.
As with the initialization, this velocity update could be exchanged for any mechanism suited to explore highly multi-modal environments or specific patterns depending on the knowledge of the search space.
If a particle i detects that its measured sensor value c_i is below the given threshold θ, meaning it has entered an object to be explored, it creates a new sub-swarm of size P_S from its immediate neighbourhood. All members of this sub-swarm are assigned the current position x_i as their last known position inside the object, denoted $\overrightarrow{in}$. If the measured value c_i does not fall below θ,

it simply updates its last known position outside any objects, denoted $\overrightarrow{out}$. If any particle would leave the search space, it is simply reflected from the edge with a small random additional change in direction. This reflection behaviour holds true for all phases.
To prevent excessive particle speeds, particle velocity is clamped once it exceeds a maximum value depending on the search space size.

$$\vec{v}_i(t+1) = \vec{v}_i(t) + \vec{a}_i(t) \tag{3.1}$$

$$\vec{a}_i(t) = \sum_{j \notin S_g} \frac{Q_j * Q_j}{(|\vec{x}_i(t) - \vec{x}_j(t))|^3} \big(\vec{x}_i(t) - \vec{x}_j(t)\big) \tag{3.2}$$

3.2.3 Contour search phase

Once a sub-swarm S_n is created, it immediately enters this phase, which consists of a modified CPSO algorithm outlined in algorithm 4.
At the beginning of this phase, a data exchange is triggered between the current particle and a random particle of the same sub-swarm with a small probability. This leads to particles travelling through the contour and enables the algorithm to detect holes in the contour.
Following this, the condition for entering the contour trace phase is checked for each particle in the sub-swarm. If it resolves to true, then this phase is skipped and the contour trace phase initiated.
Instead of tacking the local and global bests, each particle updates either its last position inside $\overrightarrow{in}_i$ or outside $\overrightarrow{out}_i$ of the object, depending on its measured value c_i. Accordingly, the velocity update deviates from the CPSO variant as shown in equations 3.3 to 3.6. As either $\overrightarrow{in}$ or $\overrightarrow{out}$ will always be equal to $\vec{x}_i$, one of the velocity components will amount to 0. As such, the usually included scaling values c_1 and c_2 as well as the random variables r_1 and r_2 are omitted for a simple factor of 0.5 for all velocity components including the inertia weight. The reduced randomness helps the algorithm to consistently return to the object contour while the equal weight for the inertial velocity allows particles to overcome narrow corners and highly concave contours.
To prevent an immediate collapse to the position of the first particle, an increased but decaying repulsion factor between members of this sub-swarm is used to retain diversity in the sub-swarm, shown in equation 3.5 with T_n denoting the age of the sub-swarm, beginning with 1.

$$\vec{v}_i^{in}(t) = \overrightarrow{in}_i(t) - \vec{x}_i(t) \tag{3.3}$$

$$\vec{v}_i^{out}(t) = \overrightarrow{out}_i(t) - \vec{x}_i(t) \tag{3.4}$$

$$\vec{a}_i(t) = \sum_{j \in S_n} \frac{Q_j^3}{(|\vec{x}_i(t) - \vec{x}_j(t)|^3 \cdot T_n} \big(\vec{x}_i(t) - \vec{x}_j(t)\big) \tag{3.5}$$

$$\vec{v}_i(t+1) = 0.5 \cdot \vec{v}_i(t) + 0.5 \cdot \vec{v}_i^{in}(t) + 0.5 \cdot \vec{v}_i^{out}(t) + \vec{a}_i(t) \tag{3.6}$$

3.2.4 Contour trace phase

This phase is determined for each particle individually during the contour search phase. After the update of the $\overrightarrow{in}_i(t)$ and $\overrightarrow{out}_i(t)$ values, if their difference is less than a set minimum value ϵ, the contour tracing update is triggered instead of the equations detailed in the previous section as the particle is now close to an edge. For contour tracing, calculating the new velocity receives an additional term representing the direction of the edge of the contour and replacing the repulsion force $\vec{a}_i(t)$:
With sufficient proximity of $\vec{in}_i(t)$ and $\vec{out}_i(t)$, the actual edge of the contour will be perpendicular to the vector $\vec{e}_i(t) = \vec{out}_i(t) - \vec{in}_(t)$. Note that this vector will be either equal to $\overrightarrow{in}_i(t)$, given the particle is currently outside of the contour or equal to $\overrightarrow{out}_i(t)$ if the particle is currently inside the contour. Using this approximation, the particle is moved along one of the normal vectors of $\vec{e}_i(t)$, denoted $\vec{n}_i(t)$. The magnitude s of this velocity can be chosen depending on the application, with smaller values increasing the accuracy of the contour approximation as well as the needed time to fully explore the contour. The full velocity update for this contour tracing is shown in equations 3.7 to 3.11 and a visualization of the vectors involved is shown in fig. 3.3
The final step in this phase is to write the particle positions to an output archive for later evaluation.

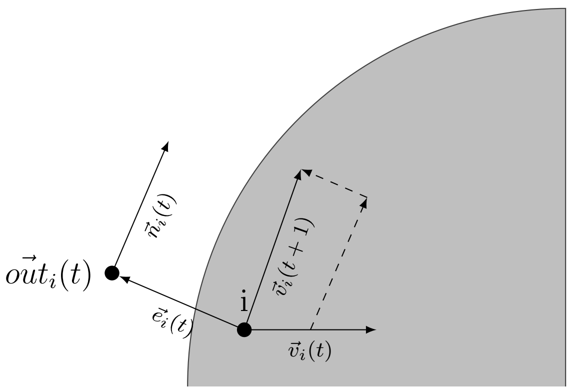


Figure 3.3: Contour trace velocity example: Particle p is currently inside the contour with previous velocity $\vec{v}_i(t)$ and $\vec{e}_i(t) = \overrightarrow{out}_i(t)$. Both of these vectors are added with 50% magnitude to the normal vector $\vec{n}_i(t)$ to obtain $\vec{v}_i(t+1)$.

$$\vec{v}_i^{in}(t) = \overrightarrow{in}_i(t) - \vec{x}_i(t) \tag{3.7}$$

$$\vec{v}_i^{out}(t) = \overrightarrow{out}_i(t) - \vec{x}_i(t) \tag{3.8}$$

$$\vec{e}_i(t) = \overrightarrow{out}_i(t) - \vec{in}_i(t) \tag{3.9}$$

$$\vec{n}_i(t) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \frac{\vec{e}_i(t)}{||\vec{e}_i(t)||} \tag{3.10}$$

$$\vec{v}_i(t+1) = 0.5 \cdot \vec{v}_i(t) + 0.5 \cdot \vec{v}_i^{in}(t) + 0.5 \cdot \vec{v}_i^{out}(t) + s \cdot \vec{n}_i(t) \tag{3.11}$$

Algorithm 5 PSCS contour search and trace phases

for each subswarm S_n **do**
  \\ search phase update for each sub-swarm
  for each particle i in S_n **do**
    exchangeData()
    if $c_i < \theta$ **then**
      $\vec{in}_i(t) = \vec{x}_i(t)$
    else
      $\vec{out}_i(t) = \vec{x}_i(t)$
    end if
    if $|\vec{out}_(t)i - \vec{in}_i(t)| < \epsilon$ **then**
      \\ contour trace phase
      $\vec{v}_i(t+1) = updateVelocity(\vec{in}_i(t), \vec{out}_i(t))$ \\Equation 3.11
    else
      \\ contour search phase
      $\vec{v}_i(t+1) = updateVelocity(\vec{in}_i(t), \vec{out}_i(t))$ \\Equation 3.6
    end if
    $\vec{x}_i(t+1) = updatePosition(\vec{v}_i(t), \vec{x}_i(t))$
  end for
end for

4 Evaluation

The goals motivating this thesis include the experimental evaluation of the PSCS algorithm and a qualitative comparison to an image-recognition based contour search algorithm. To this end, the algorithm described in the previous chapter was implemented and experiments performed. This chapter describes the implementation and the performed experiments.

4.1 Implementation

The algorithm and experimental evaluation are implemented using the python programming language. The main algorithm accepts all necessary parameters as arguments, allowing independent variation and evaluation of parameter sets. The objective function $f(x, y)$ for the particles is represented as a binary image containing the contour to be approximated.
While python packages providing various PSO implementations exist, most of the algorithm was self-implemented due to the various changes to the mathematical foundations of the PSO velocity updates and the overall control flow as well as the specific input nature for the evaluation requiring the algorithm to run on a binary image. Most of the mathematical operations are done using functions provided by the numpy package.
The importing of the images and reading of specific pixel-values as well as the image-recogniton based contour search algorithm are provided by the OpenCV python library [5]. An automated way to start experiments for each set of parameters was implemented using a simple configuration file. The output of the algorithm is written to a file during each iteration in order to reduce RAM usage.
For the evaluation, a concave hull has to be constructed from the output produced by the algorithm, which is done via the QGIS project [18], which provides a python plugin to calculate concave hulls of point clouds. The difference of the areas of the concave hull and the actual shape is then used as

Category	Parameter	Values
General	Population Size	1,5,20
	Sub-swarm Size	1,5,20
Contour search/trace phase	ϵ	10,25,50
	Step-size s	1,10,25,50
Errors	σ	0,5,10

Table 4.1: Parameter values used for experiments

metric by which the algorithm is evaluated. The data is loaded and analysed via the pandas package, and for visualization the matplotlib package was used.

4.2 Parameters

The performance of the algorithm is dependant on a variety of parameters. To reduce the parameter space to a manageable size, parameters already evaluated by previous work, such as the repulsion factor for the charged PSO repulsion, are used as recommended by the original authors. The parameter settings that were tested are shown in table 4.1. As the parameter space of all combinations is still too large for individual experimentation on each test image, an exhaustive parameter evaluation is only done for a selected image, which combines multiple convex and concave contour sections in addition to sharp angles, providing a sufficient representation of possible object shapes. The best parameter sets are then used on the remaining images and the results evaluated in comparison to the output of the OpenCV contour search.

4.3 Experiments

As mentioned previously, the first experiments were conducted in a single test image with all possible parameter combinations, with the exception being the error value, which is always set to a single value for all error sources. While the output of the algorithm allows analysis of all particle positions for any time-step, only the positions of particles in the contour tracing phase are evaluated,

as these are the only information necessary to evaluate the concave hull of the found object.
Further experiments are done on a subset of the images shown in Fig. 3.1. For these experiments, the input images are heavily blurred to represent a more natural flow of the function value instead of a sharp edge. For each image, the obtained area is then compared to the area received from running the OpenCV contour search algorithm on the same image. Each experiment is repeated 11 times to improve the statistical significance of the results. Each experiment terminates after 20.000 function evaluations have been calculated.

4.4 Error Model

With a goal of the thesis being the evaluation of the algorithm under real-world constraints, simple additive error terms are introduced for velocity, position and sensor measurements.
Error terms are calculated for each component according to Equations 4.1 to 4.3, with the error values $\mathcal{N}$ drawn for each equation individually from a two-dimensional normal distribution. The velocity $\vec{E}_v(t)$ error is applied directly during the velocity update, while the positioning error $\vec{E}_x(t)$ is only applied during during the objective function evaluation at the current particle position, combined with the measurement error $\vec{E}_f(t)$.

$$\vec{E}_v(t) = 0.01 \cdot \mathcal{N}((0,0),\sigma) \otimes \vec{v}_i(t) \tag{4.1}$$

$$\vec{E}_x(t) = \mathcal{N}((0,0),\sigma) + \vec{x}_i(t) \tag{4.2}$$

$$E_f(t) = \mathcal{N}((0,\sigma) + f(\vec{E}_x(t)) \tag{4.3}$$

$$\tag{4.4}$$

The introduction of these errors leads to situations in which the algorithm cannot produce an output from which a concave hull can be constructed. For all following experiments, these failings are tracked and displayed along with the results.

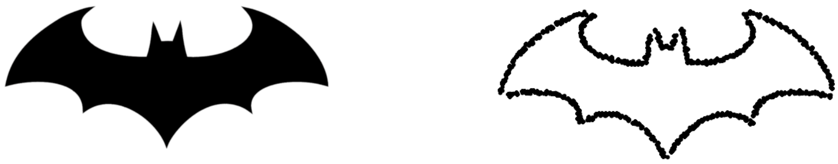
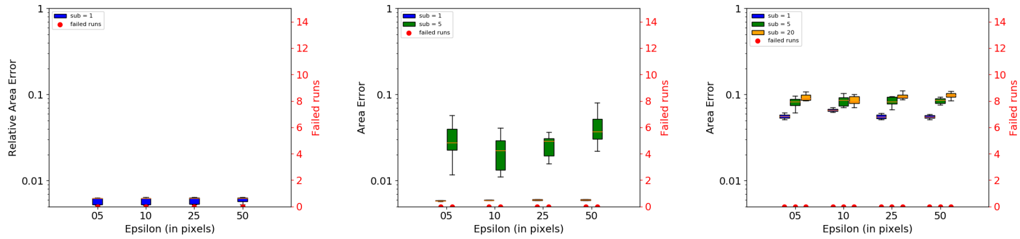


Figure 4.1: Test image and visualized algorithm output



(a) Population of 1 (b) Population of 5 (c) Population of 20

Figure 4.2: Influence of different epsilon settings without error model and a step-size of 1

4.5 Experiment for parameter evaluation

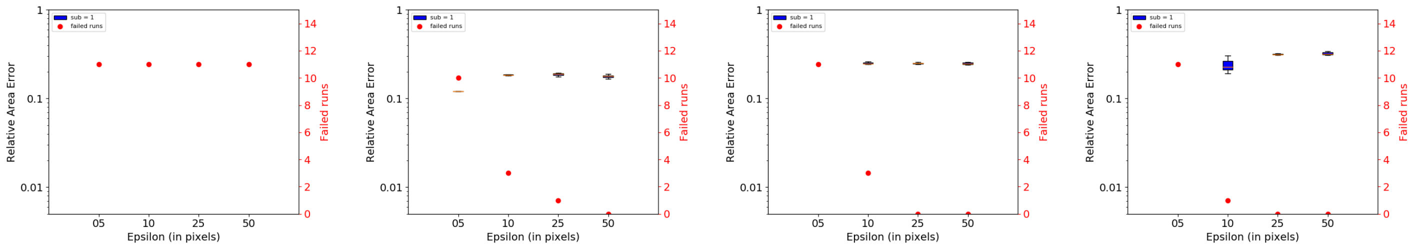
As described previously, this experiment is done on a single test function using all available parameter combinations to evaluate the influence of individual parameter changes. The test image, along with an example output of the algorithm, is shown in fig. 4.1.

The first parameter to be evaluated is the minimum distance to the contour for entering the trace phase, ϵ. Fig 4.2 shows the influence of the parameter for the different population sizes when no error is applied. Shown is the distribution of the relative error to the absolute area, along with any runs for which the algorithm failed. The colour of the box plots indicates the sub-swarm size used for the experiment.

It is interesting that the ϵ value seems to have little to no influence on the solution quality. The initial assumption was that the parameter would improve the solution quality by restricting particles to a closer area around the contour, but this is not the case.
The second thing to note is that with increasing population size the solution quality decreases. A reason for this could be that particles interfering with each others trajectories due to the application of a CPSO repulsion approach.

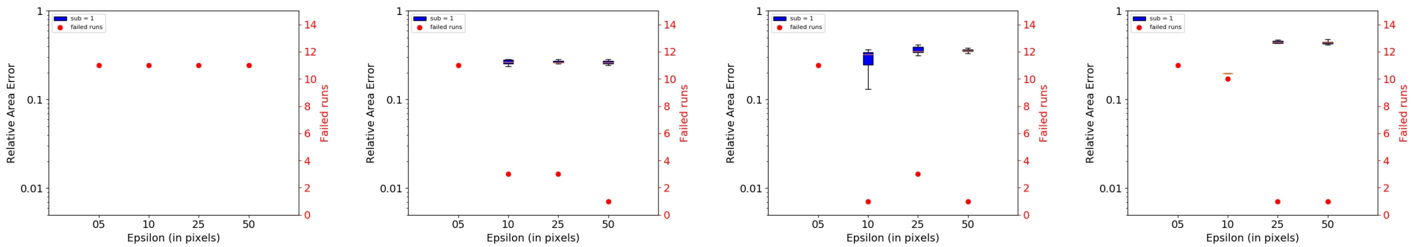
The ϵ parameter only becomes relevant when an error value is introduced, as shown in figures 4.3 and 4.4: For small values of ϵ the algorithm fails to trace the contour completely, with all runs failing to produce a viable contour. This makes sense intuitively, as the error is applied to both particle velocity and the measurement position. With larger error values, a small ϵ increasingly restricts the window in which particles enter the contour tracing phase relative to the imprecision in particle movement. If no particle ever manages to enter the contour tracing phase, the algorithm produces no output from which a contour can be constructed.

The data displayed also shows the importance of the step-size, which seems to have a similar influence on the algorithms performance: With lower step sizes, the algorithm becomes significantly more prone to failure. Figure 4.5 shows the data for the step sizes without any error model applied. The data is already trimmed to not include infeasible ϵ values. As the exact value of ϵ had no noticeable influence above 25, this was chosen as the value size to use. Instead of ϵ variation, different population sizes are shown for the step variations.



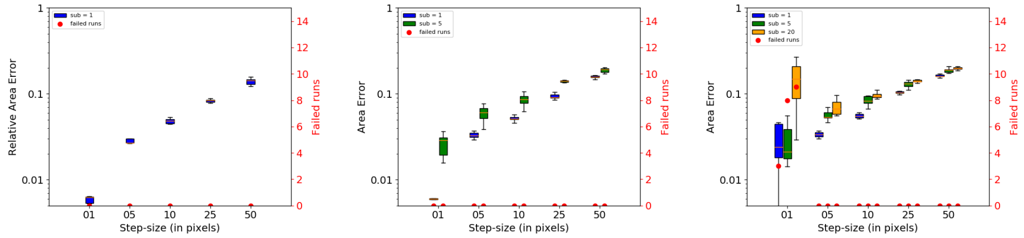
(a) Step-size 1 (b) Step-size 10 (c) Step-size 25 (d) Step-size 50

Figure 4.3: Influence of ϵ for different step sizes with a fixed population of 1 and σ of 5



(a) Step-size 1 (b) Step-size 10 (c) Step-size 25 (d) Step-size 50

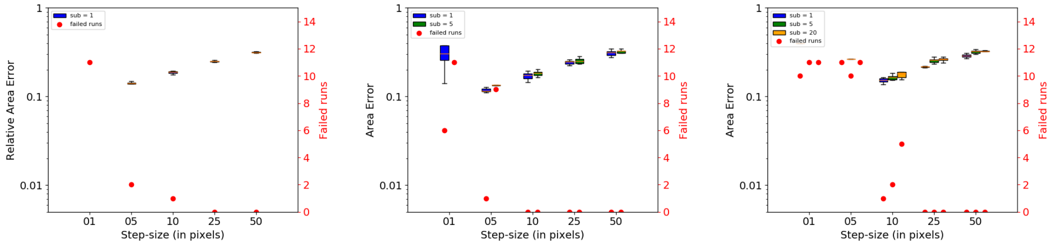
Figure 4.4: Influence of ϵ for different step sizes with a fixed population of 1 and σ of 10



(a) Population size 1 (b) Population size 5 (c) Population size 20

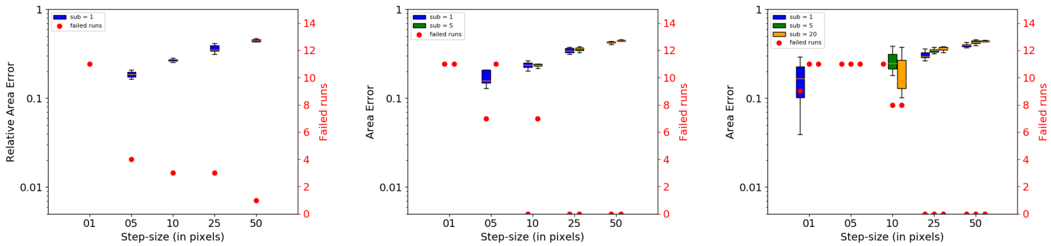
Figure 4.5: Influence of step-size for different population sizes with a fixed ϵ of 25 and no applied errors

Immediately noticeable is the impact the step-size has on the relative error of the contour area. This concurs with expectations: The step size has direct influence on the velocity of the particles during the contour trace phase, in which the boundary is approximated. More precisely, the step-size directly sets the distance travelled along the estimated tangent of the contour, i.e. the smaller this distance, the closer the particle remains to the actual contour.
Figures 4.6 and 4.7 show the data including the error model. In addition to the influence on solution quality, the step-size has a similar impact on the algorithm robustness as ϵ: To small values prevent the algorithm from tracing the contour. The reason for this is that for the smaller step-sizes the error regarding position measurement can become greater than the travelled velocity in the contour trace phase. Due to this, a particles last known position inside and outside of the contour could become identical, causing the particle to become stuck.
The data also shows another influence of the population size: While a population size of 1 shows a theoretical best performance when not subjected to errors, a population size greater than 1 improves robustness of the algorithm when the error model is included, reducing the failure rate to 0 given a sufficiently large step-size.



(a) Population size 1 (b) Population size 5 (c) Population size 20

Figure 4.6: Influence of step-size for different population sizes with a fixed ϵ of 25 and σ of 5



(a) Population size 1 (b) Population size 5 (c) Population size 20

Figure 4.7: Influence of step-size for different population sizes with a fixed ϵ of 25 and σ of 10

Category	Parameter	Values
General	Population Size	5
	Sub-swarm Size	1
Contour search/trace phase	ϵ	25
	Step-size s	25

Table 4.2: Parameter values used for the following experiments

As for the second parameter concerning population size, the sub-swarm size,the data shown in this section indicates that a size of 1 yields the best performance. This can be explained by the fact that particles do not exchange information during the contour trace phase and therefore nothing is gained by a larger sub-swarm size. However, if the contour has a hole, only a sub-swarm of size greater than one can locate this hole, since particles from the same sub-swarm occasionally exchange their memories, as explained in section 3.
Table 4.2 shows the best parameter values extracted from this experiment.

4.6 Experiment on a convex contour

With the necessary parameter settings established, the next experiment is testing the algorithm against the OpenCV contour search on the convex boundary of a circle. Fig. 4.8 shows the input image along with an example output produced by the algorithm.

Fig. 4.9 shows the data exported from the experiment, including the contour Area calculated by OpenCV

While the OpenCV contour search provides a more accurate result even for no error model, with the relative error being one order of magnitude smaller even the no-error case, it is worth noting that the parameters were selected for optimal robustness instead of accuracy. The PSCS algorithm however has shown capability of similar accuracy for a more complex shape (fig. 4.2 (a)) for a less robust parameter set. Another thing to note is the fact that the algorithm performs reliably with little variation in solution quality.

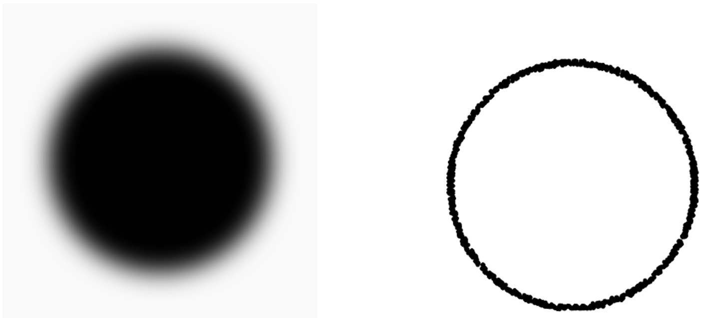


Figure 4.8: Convex test image and visualized algorithm output

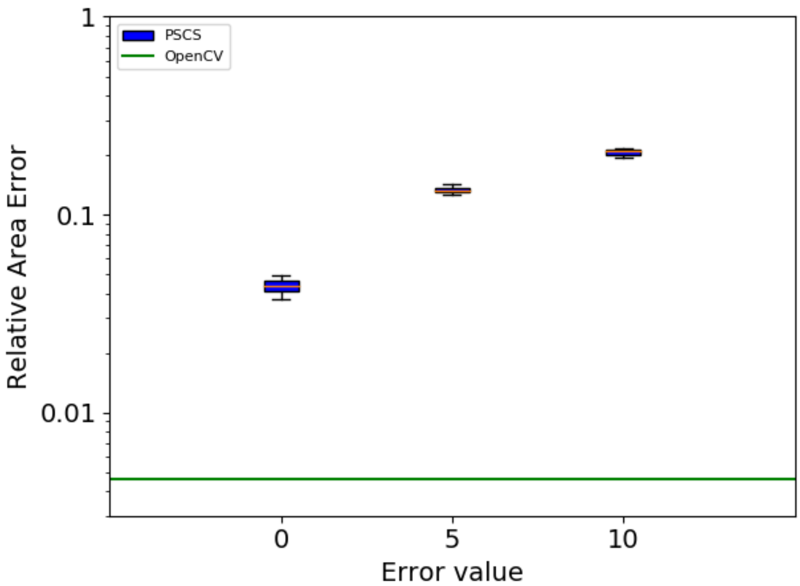


Figure 4.9: Experimental results for the convex shape given various error values

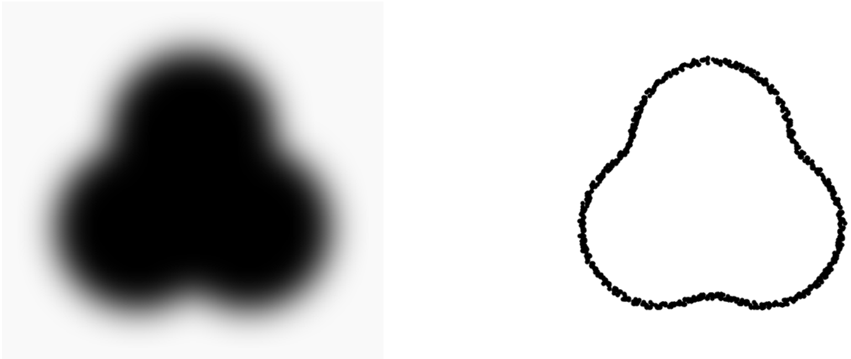


Figure 4.10: Concave test image and visualized algorithm output

4.7 Experiment on a concave contour

The next type of shape to be evaluated is a concave object shown in fig. 4.10

Fig. 4.11 shows the data exported from the experiment. While the OpenCV algorithm loses some accuracy, the performance of PSCS method remains relatively stable, with a small increase in the range of error values. However, the OpenCV remains about an order of magnitude more accurate even in the case of zero errors.

4.8 Optimum Performance

The previous experiments were done to evaluate the algorithm using parameters suited for real-world applications. For a theoretical case disregarding potential error sources a much improved performance should be achieved. This is tested with the parameter set shown in table 4.3. For this experiment a blurred version of the batman image used for parameter evaluation in section 4.5 is employed, as it combines both convex and concave elements and as such provides a more generalized shape representation. To include a second image, the concave "blot" image displayed in section 4.7 is also used.

As the standard deviation for this parameter set is to small to be properly displayed in a box plot, only the median of the algorithm outputs was calculated instead. The results are shown in table 4.4, with the PSCS algorithm

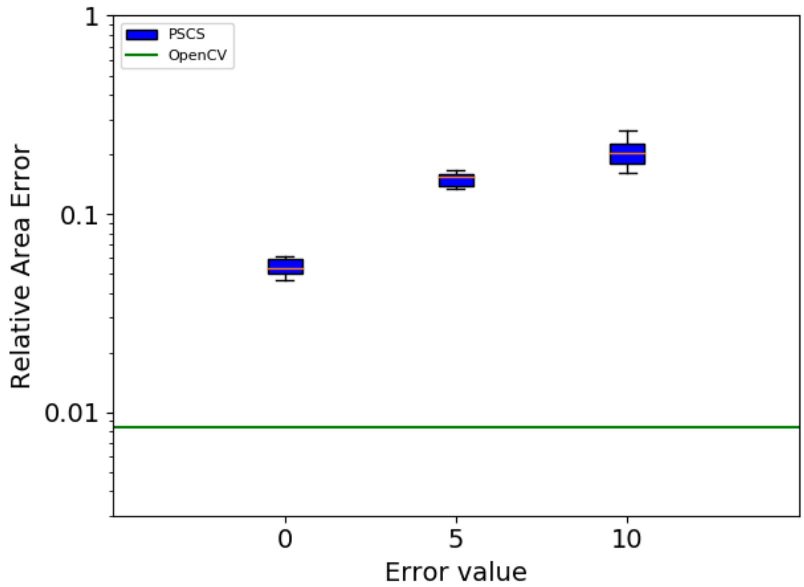


Figure 4.11: Experimental results for the concave shape given various error values

Category	Parameter	Values
General	Population Size	1
	Sub-swarm Size	1
Contour search/trace phase	ϵ	25
	Step-size s	1

Table 4.3: Parameters for optimum performance with no error model

outperforming the OpenCV method and showing consistent performance over all trial runs with very low standard deviation. While only applied to this limited set of two test images, this still shows the potential of the algorithm in a perfect, i.e. error-free environment.

Image	Algorithm	Relative error	Standard deviation
Batman	PSCS	0.011	0.00038
	OpenCV	0.014	-
Blot	PSCS	0.006	0.00006
	OpenCV	0.008	-

Table 4.4: Results of the optimum parameter set applied to two images

4.9 Non-continuous shapes

Another type of contour are the given by non-continuous shapes, i.e. shapes with holes. These contours can not be evaluated using the methods of the above sections, as no concave hull can be constructed. However examples can still be given that the algorithm is capable of finding such holes. For this, a changed parameter set is used to improve the exploration of the algorithm. Especially the sub-swarm size is set to the total population as this facilitates more data exchange between all particles, which helps the algorithm find holes. The changed parameters are detailed in table 4.5.
Fig. 4.12, shows two examples images with the corresponding output of the PSCS algorithm for these images.

Category	Parameter	Values
General	Population Size	20
	Sub-swarm Size	20
Contour search/trace phase	ϵ	25
	Step-size s	10

Table 4.5: Parameters for exploring the interior of shapes

Another example for shapes of this type would be multiple unconnected shapes. Given enough sub-swarms, the PSCS method can potentially find a number of shapes equal to the number of sub-swarms depending on the global search pattern. Again, the parameters were tweaked to suit this specific contour type (table 4.6) and the results are displayed in fig. 4.13

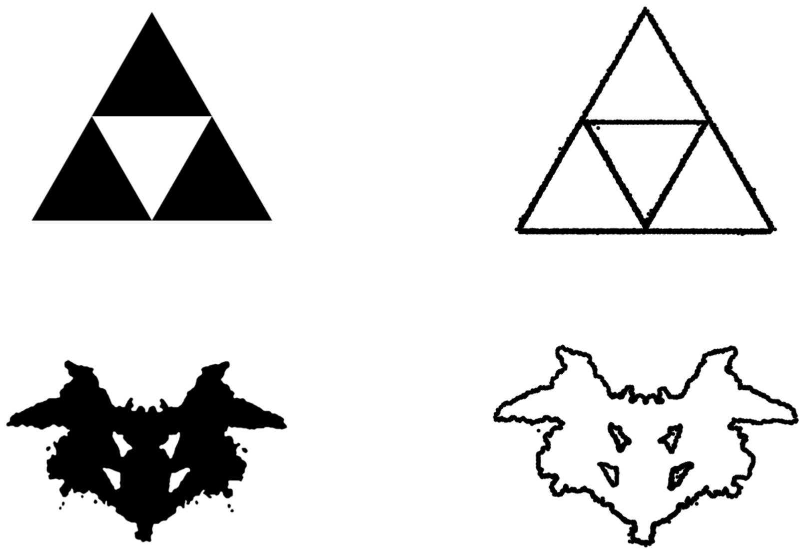


Figure 4.12: Test images and visualized algorithm outputs for shapes including holes

Category	Parameter	Values
General	Population Size	5
	Sub-swarm Size	1
Contour search/trace phase	ϵ	25
	Step-size s	10

Table 4.6: Parameters for exploring disconnected shapes

4.10 Discussion

The experimental results are promising. The algorithm succeeded in finding the contours of a variety of shapes. With the implemented error models, the ability of the algorithm to perform in a realistic scenario has been confirmed. Furthermore, the algorithm has even shown capability to match or outperform image-recognition based methods in a error-free environment. However, once an error model is introduced, the algorithm cannot perform as well as the OpenCV method, which is not hindered by the constraints of a robotic application, but would be hindered by other constraints related to imaging

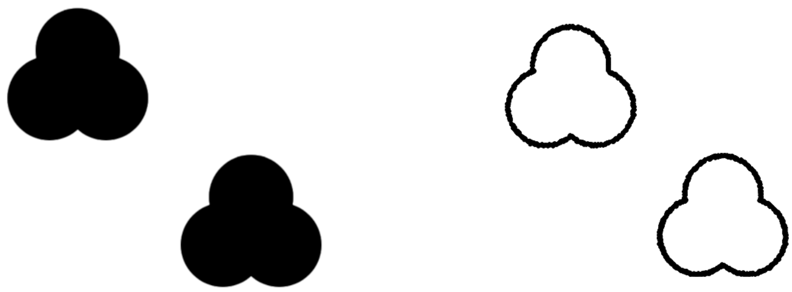


Figure 4.13: Test image and visualized algorithm output for a disconnected shape

sensors which were not considered for the experiments.

Most importantly, the initial experiments found a parameter set for which the algorithm performs reliably, proving the basic viability of the approach. While errors encountered in robotic applications hinder the algorithm, a contour can be found reliably with acceptable accuracy.
The main influence on solution quality was found to be the step-size, with population size and the epsilon value only having lesser influence. The main reason the optimum, i.e. minimal step-size can not be applied in every scenario is the fact that particles get stuck given high enough position and velocity errors.

In the case of non-continuous shapes, an analysis of the algorithm could not be performed in the same way, as it is not possible to create a concave hull from the resulting point clouds. However, it was shown that the algorithm is capable of finding holes in the traced contour as well as trace multiple distinct contours.

5 Conclusion

In this thesis a new algorithm was designed to apply the PSO approach to the problem of 2D-contour search. The algorithm was implemented and tested for various test cases. The first goal of the thesis was to create the algorithm and show that it works in a theoretically perfect, i.e. error-free environment, which was confirmed by the experiments.

The second goal and a large part of the motivation for this thesis was to establish if the algorithm could be applied under real-world robotic constraints such as velocity errors and sensor noise. The experiments conducted provide conclusive evidence that this was achieved, with a parameter set having been established under which the algorithm performs reliably in such environments.

The last goal was to compare the performance of the PSCS algorithm to a contour search algorithm based on image recognition. For this comparison the OpenCV contour search algorithm was chosen. Without an error model, PSCS was shown to outperform the OpenCV algorithm, albeit in a limited number of test cases. However, with the introduction of the error model, the OpenCV algorithm outperformed the PSCS method by an order of magnitude.

The main limiting factor for the performance of the algorithm was found to be the step size: The smaller the step size, the better the algorithm approximates the traced contour. However, for larger error values a small step-size causes the particles to get stuck. Preventing this behaviour could yield significant improvements to the algorithms performance.

While not a perfect replacement for image based contour search yet, the work done in this thesis shows a promising potential alternative for cases which render imaging technologies infeasible.

Bibliography

[1] Edward Argyle and Avi Rosenfeld. Techniques for edge detection. *Proceedings of the IEEE*, 59(2):285–287, 1971.

[2] Tim Blackwell and Jürgen Branke. Multi-swarm optimization in dynamic environments. In *Workshops on Applications of Evolutionary Computation*, pages 489–500. Springer, 2004.

[3] Tim M Blackwell. Swarms in dynamic environments. In *Genetic and Evolutionary Computation Conference*, pages 1–12. Springer, 2003.

[4] Tim M Blackwell and Peter J Bentley. Dynamic search with charged swarms. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 19–26. Morgan Kaufmann Publishers Inc., 2002.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] John Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

[7] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698, 1986.

[8] Sheetal Doctor, Ganesh K Venayagamoorthy, and Venu G Gudise. Optimal pso for collective robotic search applications. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 2, pages 1390–1395. IEEE, 2004.

[9] F. Heppner and U. Grenader. A stochastic nonlinear model for coordinated bird flocks. *S. Kasner, ed., The Ubiquity of Chaos. AAAS Publications*, 1995.

[10] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[11] J Kennedy and R Eberhart. Particle swarm optimization (pso). In *Proc. IEEE International Conference on Neural Networks, Perth, Australia*, pages 1942–1948, 1995.

[12] Xiaodong Li and Khanh Hoa Dam. Comparing particle swarms for tracking extrema in dynamic environments. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 3, pages 1772–1779. IEEE, 2003.

[13] Raman Maini and Himanshu Aggarwal. Study and comparison of various image edge detection techniques. *International journal of image processing (IJIP)*, 3(1), 2008.

[14] Sanaz Mostaghim, Christoph Steup, and Fabian Witt. Energy aware particle swarm optimization as search mechanism for aerial micro-robots. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2016.

[15] Konstantinos E. Parsopoulos and Michael N. Vrahatis. *Particle Swarm Optimization and Intelligence: Advances and Applications.* Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2010.

[16] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

[17] Riccardo Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008, 2008.

[18] QGIS Development Team. *QGIS Geographic Information System.* Open Source Geospatial Foundation, 2009.

[19] Jonghoon Seo, Seungho Chae, Jinwook Shim, Dongchul Kim, Cheolho Cheong, and Tack-Don Han. Fast contour-tracing algorithm based on a pixel-following method for image sensors. *Sensors*, 16(3), 2016.

[20] Lisa L Smith, Ganesh K Venayagamoorthy, and Phillip G Holloway. Obstacle avoidance in collective robotic search using particle swarm optimization. 2006.

[21] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30:32–46, 1985.

[22] Yitzhak Yitzhaky and Eli Peli. A method for objective edge detection evaluation and detector parameter selection. *IEEE Transactions on pattern analysis and machine intelligence*, 25(8):1027–1033, 2003.

[23] Mauricio Zambrano-Bigiarini, Maurice Clerc, and Rodrigo Rojas. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In *2013 IEEE Congress on Evolutionary Computation*, pages 2337–2344. IEEE, 2013.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Dominik Weikert April 25, 2019