Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2020

Route Planning and Design of Autonomous Underwater Mine Reconnaissance Through Multi-Vehicle Cooperation

Jakob Hanskov Palm

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

Route Planning and Design of Autonomous Underwater Mine Reconnaissance Through Multi-Vehicle Cooperation:

Jakob Hanskov Palm

LiTH-ISY-EX–20/5334–SE

Supervisor: **Pavel Anistratov**
ISY, Linköpings universitet
Simon Keisala
Saab Dynamics

Examiner: **Björn Olofsson**
ISY, Linköpings universitet

Division of Vehicular Systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

Autonomous underwater vehicles have become a popular countermeasure to naval mines. Saab's AUV62-MR detects, locates and identifies mine-like objects through three phases. By extracting functionality from the AUV62-MR and placing it on a second vehicle, it is suggested that the second and third phases can be performed in parallel. This thesis investigates how to design the second vehicle so that the runtime of the mine reconnaissance process is minimized. A simulation framework is implemented to simulate the second and third phases of the mine reconnaissance process in order to test various design choices.

The vehicle design choices in focus are the size and the route planning of the second vehicle. The route-planning algorithms investigated in this thesis are a nearest neighbour algorithm, a simulated annealing algorithm, an alternating algorithm, a genetic algorithm and a proposed Dubins simulated annealing algorithm. The algorithms are evaluated both in a static environment and in the simulation framework. Two different vehicle sizes are investigated, a small and a large, by evaluating their performances in the simulation framework. This thesis takes into account the limited travelling distance of the vehicle and implements a k-means clustering algorithm to help the route planner determine which mine-like objects can be scanned without exceeding the distance limit.

The simulation framework is also used to evaluate whether parallel execution of the second and third phases outperforms the current sequential execution. The performance evaluation shows that a major reduction in runtime can be gained by performing the two phases in parallel. The Dubins simulated annealing algorithm on average produces the shortest paths and is considered the preferred route-planning algorithm according to the performance evaluation. It also indicates that a small vehicle size results in a reduced runtime compared to a larger vehicle.

Acknowledgments

I would like to thank my supervisors at Saab, Alfons Råberg and Simon Keisala, who both have been fantastic at providing support and guidance. I would also like to extend my gratitude to everyone else at the Saab office who showed interest in my thesis and answered any questions I might have had.

I especially want to thank my supervisor at Linköping University, Pavel Anistratov, who was absolutely phenomenal. I would simply not have been able to finish my thesis at this time, had Pavel not been so generous with his time.

Lastly, I want to thank friends and family for supporting me in every way they possibly could during the stressful period of finishing my thesis and for attending my final presentation.

Linköping, September 2020
Jakob Palm

Contents

List of Figures ix

List of Tables x

Notation xi

1 Introduction 1
- 1.1 Vehicle modules . . . 2
  - 1.1.1 Size and kinematics . . . 2
  - 1.1.2 Positioning . . . 2
  - 1.1.3 Communication . . . 2
  - 1.1.4 Route planning . . . 3
- 1.2 Related research . . . 4
- 1.3 Problem formulation . . . 5
- 1.4 Outline . . . 5

2 Theory 7
- 2.1 Dubins path and vehicle model . . . 7
- 2.2 The Dubins travelling salesman problem . . . 9
- 2.3 Nearest neighbour . . . 9
- 2.4 Simulated annealing . . . 10
- 2.5 Alternating algorithm . . . 11
- 2.6 Genetic algorithm . . . 11
- 2.7 K-means clustering . . . 12

3 Implementation 15
- 3.1 Dubins path . . . 15
- 3.2 Nearest neighbour . . . 15
- 3.3 Simulated annealing . . . 16
- 3.4 Gradient of the seabed . . . 16
- 3.5 Alternating algorithm . . . 17
- 3.6 Genetic algorithm . . . 18
- 3.7 Dubins simulated annealing . . . 19

3.7.1 Rotation 20
3.7.2 Reversion 20
3.7.3 Insertion 21
3.7.4 Swap 22
3.8 Simulation setup 23
3.9 K-means clustering 24

4 Performance evaluation **27**
4.1 Static environment 27
4.1.1 Average results of multiple simulations in a static environment 28
4.2 Simulation framework 29
4.2.1 Average results of multiple simulations when applied to the investigated scenario 31
4.3 Vehicle sizes 32
4.3.1 Average results of multiple simulations with a small and a large vehicle 33
4.4 Trips 33
4.5 Parallel versus sequential execution 34
4.5.1 Average results of multiple simulations with parallel and sequential execution 36
4.5.2 Slower camera vehicle 37
4.5.3 Larger search area 38

5 Discussion **41**
5.1 Route planning of the camera vehicle 41
5.2 Size of the camera vehicle 43
5.3 Parallel versus sequential execution 43

6 Conclusions **45**
6.1 Which route-planning algorithm is best suited for the camera vehicle? 45
6.2 What torpedo size should the camera vehicle be based on? 45
6.3 How does the performance of parallel execution of phase 2 and phase 3 compare to sequential execution? 46
6.4 Future work 46

Bibliography **47**

List of Figures

1.1 An example path of the AUV62-MR as seen from above. 3

2.1 Three examples of the Dubins path. 8

2.2 An example path with headings generated by the alternating algorithm. 12

3.1 Angle for the angular-metric cost function [1]. 16

3.2 The heading limits caused by the gradient as seen from above. . . 17

3.3 An example of a heading being adjusted to fit the heading limits. . 18

3.4 An example of the rotation mutation. 20

3.5 An example of the reversion mutation. 21

3.6 An example of the insertion mutation. 22

3.7 An example of the swap mutation. 22

3.8 The first entry point to the search area. 24

3.9 The second entry point to the search area. 24

3.10 An example of how the k-means clustering algorithm determined which MLOs to skip and which to scan. 25

4.1 The result of each algorithm in a static environment with 50 MLOs. 28
4.2 Average performances of 30 simulations in a static environment. . 29
4.3 The result of each algorithm when applied to the investigated scenario with 50 MLOs. 30
4.4 The chosen headings when each algorithm was applied to the investigated scenario. 31
4.5 Average performances of 30 simulations when applied to the investigated scenario. 32
4.6 The result of the simulation examples with a small and a large camera vehicle with 50 MLOs. 33
4.7 Average performances of 20 simulations with a small and a large camera vehicle. 34
4.8 Average trips of 10 simulations. 35
4.9 The result of the simulations with parallel and sequential execution. 35
4.10 Average performances of 10 simulations with parallel and sequential execution. 36
4.11 Average performances of 10 simulations with a slower camera vehicle. 37
4.12 Average performances of 10 simulations in a larger search area. . . 38

List of Tables

2.1 Motion primitives and their corresponding steering inputs. 8

3.1 Simulated annealing mutations. 17
3.2 Genetic algorithm mutations. 19

4.1 The runtime and distance travelled by the camera vehicle during each simulation. 30
4.2 The runtime and distance travelled by the camera vehicle during the evaluation of vehicle sizes. 32
4.3 The runtime and total distance travelled during parallel and sequential simulations. 35

Notation

Notations

Notation	Description
$\rho_{\min}$	Minimum turn radius
$c_{\max}$	Maximum curvature
$v_{\max}$	Maximum velocity
x	x-coordinate
y	y-coordinate
θ	Heading angle
s	State (x, y, θ)
Λ	Set of points
P	Set of points, visiting order of Λ
S	Set of states
Θ	Set of headings
u	Steering input
$D(s_i, s_j)$	Dubins cost between state s_i and state s_j
$L_D(S)$	Dubins cost through a set of states S
$L_E(P)$	Euclidean cost through a set of points P

Abbreviations

Abbreviation	Description
AUV	Autonomous underwater vehicle
DSA	Dubins simulated annealing
DTSP	Dubins travelling salesman problem
ETSP	Euclidean travelling salesman problem
GA	Genetic algorithm
MLO	Mine-like object
MR	Mine reconnaissance
NN	Nearest neighbour
REA	Rapid environmental assessment
SA	Simulated annealing
SAS	Synthetic aperture sonar
TSP	Travelling salesman problem

1

Introduction

Naval mines were frequently used during the wars of the 20th century as a weapon against surface ships and submarines. Mines that are not triggered eventually rust and sink to the seabed, yet they are still highly capable of exploding. Saab estimates anywhere between 50 000–100 000 naval mines are still scrambling around in the Baltic Sea [2] causing a potential danger to ships and people alike. Autonomous underwater vehicles (AUVs) have become a popular countermeasure to naval mines through rapidly scanning large areas of the ocean without the need of a manned vehicle. Saab has developed an AUV called AUV62-MR with the purpose of detecting, locating and identifying mine-like objects (MLOs) [3]. This process is known as *mine reconnaissance* (MR) and for the AUV62-MR it includes the following three phases:

1. A rapid environmental assessment (REA) that makes a rapid scan of the ocean floor bathymetry.

2. A synthetic aperture sonar (SAS) search that generates a high-resolution bathymetry of the ocean, which later can be used to detect MLOs. This phase uses the rough bathymetry from the REA to identify which areas the SAS search can be performed on.

3. A camera search that closely scans the discovered objects from the SAS scan.

The AUV62-MR executes all three phases sequentially. The sensor system required for an SAS search is expensive, while the camera search requires movement close to the bottom of the ocean involving high risk for the vehicle. The camera system is significantly cheaper than the SAS system. Therefore, Saab would like to separate the two systems by mounting the camera module on a separate vehicle, from now on this vehicle is referred to as the camera vehicle.

Phase 2 and phase 3 could then theoretically be executed in parallel in order to optimize the speed at which the entire process is executed. It also opens up for the possibility of having multiple camera vehicles scanning MLOs in case a large area needs to be covered. The purpose of this thesis is to investigate how a camera vehicle should be designed in order to minimize the execution time of the mine reconnaissance process. In doing so, a framework for simulating and testing such a vehicle has to be developed.

1.1 Vehicle modules

The AUV62-MR is built out of a torpedo base and it was concluded at Saab that the camera vehicle should follow the same principal design. This leads to the possibility of reusing certain modules from the AUV62-MR, while other vehicle properties have to be redefined. Any module not mentioned in this thesis is assumed reusable from the AUV62-MR or simply not of interest at this time for Saab.

1.1.1 Size and kinematics

Torpedoes come in different sizes, and while the AUV62-MR is based on a heavy torpedo weighing approximately 1.4 tonnes, lighter bases are proposed for the camera vehicle. A larger vehicle allows for a bigger battery, while a smaller vehicle grants more flexibility when it comes to both deployment and vehicle dynamics. The vehicle dynamics of a torpedo are constrained by a minimum turn radius as well as a maximum velocity, these are from now on referred to as the kinematic constraints of the vehicle.

1.1.2 Positioning

The AUV62-MR uses different sensors to acquire an estimation of its own velocity, and with that it can calculate its relative position to its origin. This functionality is assumed to also work for the camera vehicle. The velocity estimation is influenced by measurement noise and given a long travelling distance it can provide a slight uncertainty in positioning. While the AUV62-MR can correct its internal position using data from the SAS scan, the camera vehicle must reset its internal position by rising to the surface to reach a GPS signal before the positioning error affects the proficiency of the camera scan. This is approximated with a maximum travelling distance before the camera vehicle has to return to its origin where it can either reset its internal position or be replaced by a fresh vehicle. For this thesis, the maximum travelling distance also covers the limitation of the battery.

1.1.3 Communication

The vehicles transmit information to one another through sound waves in the water, generally known as acoustic communication. The SAS scan performed by the AUV62-MR also uses sound waves through sonar (*sound navigation and*

ranging), but two different sound waves can disrupt one another at collision. For this reason, the camera vehicle and the AUV62-MR cannot communicate with each other while the AUV62-MR is scanning. Sound waves generally travels at a velocity of roughly 1500 m/s through water, but the communication between the vehicles is approximated as instant for this thesis. It is also approximated as perfect, meaning that it experiences no measurement noise or restrictions such as a limited distance.

1.1.4 Route planning

When executing phase 2 of the MR process, the AUV62-MR follows a simple lawnmower pattern shown in Figure 1.1. The path of this pattern is generated offline during the initiation of the mission. During its search, the AUV62-MR detects MLOs and transmits them to the camera vehicle whenever possible. For this reason, the camera vehicle is required to plan a route online which leads to the desire of a low computation time.

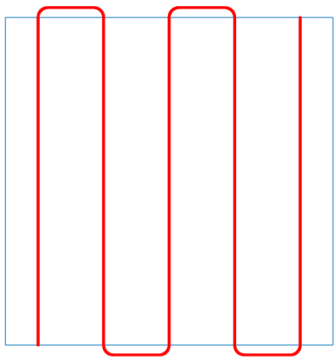


Figure 1.1: *An example path of the AUV62-MR as seen from above.*

A long computation time can cause the planned route to become outdated, if it is not accounted for in the route-planning problem. This is caused by the vehicle already having moved away from the position that the planned route originated from. Note that a torpedo-based AUV cannot stay put at a location, but must keep moving in order not to sink. This thesis aims to find a route-planning algorithm that computes in negligible time, and does not account for the computation time in the route-planning problem.

The objective of the route planner for the camera vehicle is to find the shortest route passing all the transmitted MLOs while keeping each trip below the maximum travelling distance. When the vehicle approaches its maximum travelling distance the route planner must decide which MLOs to ignore for now so that the vehicle can reset before either the battery runs out or the uncertainty in positioning grows above the threshold of affecting the proficiency of the search.

The AUV62-MR cannot differentiate between MLOs and the camera vehicle in case it were to be scanned by SAS, causing the camera vehicle having to avoid entering any part of the search area that has not yet been scanned by the AUV62-MR. Since the search area potentially contains explodable mines, it is preferable to spend the least amount of time possible in the search area and so the camera vehicle should leave the area when awaiting new MLOs to scan.

The data acquired from the SAS scan include the locations of the detected MLOs as well as the gradient of the seabed at each location. The route planner should take into account the gradients in order to minimize the risk of hitting the bottom.

1.2 Related research

A kinematically constrained vehicle such as the camera vehicle is generally referred to as a non-holonomic vehicle [4] and is often modelled with a Dubins vehicle [5]. Except for the non-holonomic constraint, the route-planning problem of the camera vehicle is closely related to the travelling salesman problem (TSP), which aims to find the shortest route through a given set of points by sorting the set of points according to visiting order [6]. The TSP often uses Euclidean distances between each pair of points and is then referred to as the Euclidean TSP (ETSP). A Dubins vehicle, however, cannot traverse the ETSP solution because of its minimum turn radius. In order to find the optimal path for a Dubins vehicle, it was shown in [5] that the optimal heading at each point must also be known. The problem of both sorting a given set of points as well as finding the optimal heading for each point expands the generic TSP and is referred to as the Dubins TSP (DTSP) [7].

The DTSP is an NP-hard problem [8], meaning that an exact solution cannot be found within polynomial time. Instead, approximate solutions are usually of interest. Finding the best approximate solution to the DTSP is widely studied and a large variety of approaches has been proposed.

Generally, the DTSP is either approached as one single problem, simultaneously finding the optimal visiting order as well as the headings, or as two separate problems. Examples of the latter include [9], which proposes solving the DTSP by first solving the initial ETSP using the renowned Lin-Kernighan heuristic [10], after which the authors propose discretizing the otherwise continuous headings to reduce the size of the problem. At this point, the authors of [9] set up the problem as a network in which every point is a layer and solve it using an A*-algorithm [11].

The authors of [12] instead propose a simpler, yet popular, method to assign approximate headings to every point known as the alternating algorithm. It only requires one iteration through the given set of points and is therefore often used as a lower bound and for performance comparison for more advanced algorithms. Other proposed methods of solving the ETSP include the nearest neighbour algo-

rithm and simulated annealing [13].

A common approach of solving the DTSP as a single problem is through a genetic algorithm [14, 15], which aims to find an increasingly more optimal route through evolutionary methods.

1.3 Problem formulation

This thesis investigates the performance of the nearest neighbour algorithm and simulated annealing when they are used along with the alternating algorithm to solve the DTSP. They are compared to the performance of a genetic algorithm and a proposed Dubins simulated annealing algorithm. To determine which MLOs to ignore when the camera vehicle approaches its maximum travelling distance, the performance of the Matlab built-in KMEANS clustering algorithm [16] is investigated.

Two different vehicle sizes are tested, meaning that two different minimum turn radii, maximum velocities and maximum travelling distances are compared. Finally, it is investigated whether parallel execution outperforms the current sequential execution of phase 2 and phase 3.

Ultimately, the aim of this thesis is to answer the following questions:

- Which route-planning algorithm is best suited for the camera vehicle?
- What torpedo size should the camera vehicle be based on?
- How does the performance of parallel execution of phase 2 and phase 3 compare to sequential execution?

1.4 Outline

This thesis is divided into six chapters. Chapter 2 presents the theory behind the investigated algorithms, after which Chapter 3 describes how they were implemented for this thesis. Chapter 4 then evaluates different scenarios in which the algorithms are applied. The performance evaluations are discussed in Chapter 5 and conclusions are drawn in Chapter 6.

2
Theory

This chapter briefly presents the underlying theory of this thesis. Firstly, the theory used to model the vehicles and generate their paths is presented. Secondly, the DTSP is described in further detail, and finally all the algorithms studied in this thesis are presented. Note that the algorithms were adopted from the literature and the reader is referred to the literature for more in-depth information regarding each algorithm.

2.1 Dubins path and vehicle model

For this thesis, a vehicle is defined by a state $s = (x, y, \theta)$, where (x, y) are the coordinates of the vehicle and θ is the heading of the vehicle, a maximum velocity $v_{\max}$, and a minimum turn radius $\rho_{\min}$. For such a vehicle, it was shown in [5] that the shortest path from an initial state s_I to a goal state s_G consists of no more than three motion primitives, where each motion primitive has to follow either a path of maximum curvature or a straight line for an interval of time. With the maximum curvature defined as

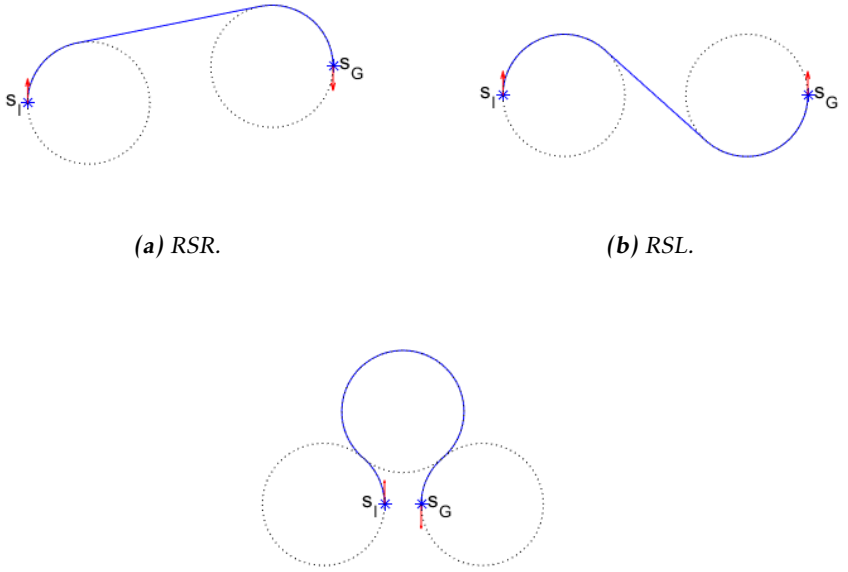
$$c_{\max} = \frac{1}{\rho_{\min}}, \tag{2.1}$$

the vehicle has to turn as sharply as possible in order to follow a path of such curvature, which limits the choice of steering input u. The motion primitives are presented in Table 2.1 with their corresponding steering inputs. It was also proven in [5] that only six concatenations of motion primitives can possibly result in an optimal path: LRL, LSL, RLR, RSR, LSR and RSL. Examples of Dubins paths are shown in Figure 2.1. Note that the red arrows indicate the headings.

Table 2.1: *Motion primitives and their corresponding steering inputs.*

Motion primitive	**Steering input** u
L	1
S	0
R	-1

Left (L), Straight (S) and Right (R)



(a) *RSR.* ***(b)*** *RSL.*

(c) *LRL.*

Figure 2.1: *Three examples of the Dubins path.*

The Dubins vehicle model is now derived as

$$\begin{aligned} \dot{x} &= v_{\max} \cdot \cos\theta, \\ \dot{y} &= v_{\max} \cdot \sin\theta, \\ \dot{\theta} &= u \cdot c_{\max}. \end{aligned} \tag{2.2}$$

2.2 The Dubins travelling salesman problem

While the Dubins path only connects two given states, it was proven that the optimal path through more than two states is obtained by concatenating Dubins paths [17]. Let $\Lambda = (\lambda_1, \ldots, \lambda_n)$ be a given set of points. The objective of the DTSP is to find a permutation $P = (p_1, \ldots, p_n)$ of Λ and a set of matching headings $\Theta = (\theta_1, \ldots, \theta_n)$ that minimizes the total path length L_D. Note that a permutation P of Λ is equivalent to a visiting order of the given set of points. Let $S = (s_1, \ldots, s_n)$ denote the states along the produced path, where $s_i = (p_i, \theta_i) = (x_i, y_i, \theta_i)$ for $i = 1, \ldots, n$. The Dubins distance between a state s_i and another state s_j is denoted by $D(s_i, s_j)$, which corresponds to the cost of travelling from s_i to s_j for the Dubins vehicle. The total path length is now given by

$$L_D(S) = \sum_{i=1}^{n-1} D(s_i, s_{i+1}). \tag{2.3}$$

In order to minimize L_D, the DTSP is often divided into two separate problems as described in Section 1.2. This requires P to instead be approximated by attempting to minimize the total Euclidean distance of the path

$$L_E(P) = \sum_{i=1}^{n-1} \sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2}. \tag{2.4}$$

2.3 Nearest neighbour

In order to find the visiting order P of a given set of points, the nearest neighbour algorithm starts out with a chosen starting point and iteratively extends the path by adding the nearest unvisited point until the path has been completed [18], see Algorithm 1.

Algorithm 1: Nearest neighbour

input : $\Lambda = (\lambda_1, \ldots, \lambda_n)$
output: $P = (p_1, \ldots, p_n)$

$p_1 \leftarrow$ starting point

for $i \leftarrow 2$ **to** n **do**
  $p_i \leftarrow$ nearest unvisited point to p_{i-1}
end

2.4 Simulated annealing

Simulated annealing combines statistical mechanics with combinatorial optimization to approximate the global optimum of a function. The algorithm is based on annealing in metallurgy, in which the material exhibits defections when its temperature is dropped too rapidly. The temperature must be slowly and carefully lowered in order to find the ground state of the matter. The authors of [13] took inspiration from this process and saw similarities to combinatorial optimization problems such as the TSP.

Let $\Lambda = (\lambda_1, \ldots, \lambda_n)$ be a given set of points. The optimal visiting order of Λ is approximated by iteratively generating permutations of Λ and continuously keeping the best one. The lower the cost of the permutation is, the better it is. For the TSP, the cost is equivalent to the length of the path. The temperature of the process corresponds to a chance of a worse performing permutation to be chosen. For each iteration the temperature is lowered, but it is kept steady during a certain amount of subiterations. The temperature is one of the main traits of simulated annealing and is meant to force the search out of local optimums. See Algorithm 2 for pseudocode of simulated annealing, as adopted from [13], when it is used to determine the visiting order of Λ.

Algorithm 2: Simulated annealing

input : $\Lambda = (\lambda_1, \ldots, \lambda_n)$
output: $P = (p_1, \ldots, p_n)$

$P \leftarrow$ random permutation of Λ
$C_{best} \leftarrow \text{cost}(P)$
$T \leftarrow$ initial temperature
$\alpha \leftarrow$ temperature decrease rate

for $i \leftarrow 1$ **to** $iterations$ **do**
  for $j \leftarrow 1$ **to** $subiterations$ **do**
    $P_{new} \leftarrow$ random permutation of Λ
    $C_{new} \leftarrow \text{cost}(P_{new})$
    $\gamma \leftarrow$ random value between 0 and 1
    if $C_{new} < C_{best}$ ***or*** $\gamma < e^{-\frac{C_{new} - C_{best}}{T}}$ **then**
      $C_{best} \leftarrow C_{new}$
      $P \leftarrow P_{new}$
    end
  end
  $T \leftarrow \alpha \cdot T$
end

2.5 Alternating algorithm

The alternating algorithm was first proposed in 2005 in [12] with the purpose of assigning headings at all points and has since then become a popular metric of performance for other algorithms. Let $P = (p_1, \ldots, p_n)$ be the approximate optimal visiting order of a given set of points. Let $\Theta = (\theta_1, \ldots, \theta_n)$ be a set of headings so that $S = (s_1, \ldots, s_n)$ and $s_i = (p_i, \theta_i)$ for $i = 1, \ldots, n$. Given P, the alternating algorithm obtains Θ by retaining all the odd-numbered edges and replacing all even-numbered edges with Dubins paths, see Algorithm 3.

Algorithm 3: Alternating algorithm

input : $P = (p_1, \ldots, p_n)$
output: $S = (s_1, \ldots, s_n)$

$\theta_1 \leftarrow$ orientation of segment from p_1 to p_2
$s_1 \leftarrow (p_1, \theta_1)$

for $i \leftarrow 2$ **to** $n - 1$ **do**
  if *i is even* **then**
    $\theta_i \leftarrow \theta_{i-1}$
  else
    $\theta_i \leftarrow$ orientation of segment from p_i to p_{i+1}
  end
  $s_i \leftarrow (p_i, \theta_i)$
end

```
/* Assumes the last point must connect to the starting
   point                                                 */
```

if *n is even* **then**
  $\theta_n \leftarrow \theta_{n-1}$
else
  $\theta_n \leftarrow$ orientation of segment from p_n to p_1
end
$s_n \leftarrow (p_n, \theta_n)$

Note that the generic alternating algorithm assumes that the traveller must return to the starting point. An example path with headings generated by the alternating algorithm is shown in Figure 2.2.

2.6 Genetic algorithm

The genetic algorithm is a global search algorithm, which aims to find an approximate optimum by mimicking the natural evolution process [19]. A population in this context is a collection of individuals, where each individual corresponds to a solution of the investigated problem. The genetic algorithm seeks the global optimum by applying survival of the fittest on the population over multiple generations.

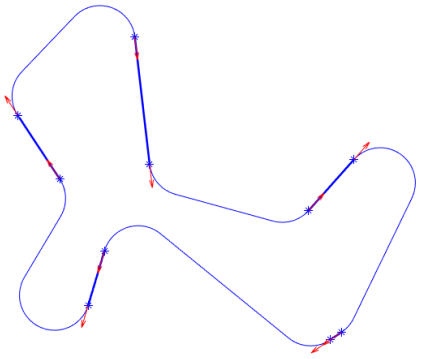


Figure 2.2: *An example path with headings generated by the alternating algorithm. All straight edges are highlighted.*

Let $\Lambda = (\lambda_1, \ldots, \lambda_n)$ be a given set of points. Each individual in the population corresponds to a permutation of Λ. The fitness of each individual is proportionate to the length of its path. Individuals are chosen for breeding the next generation through roulette wheel selection, which means that the higher the fitness, the higher the chance of being chosen. The chosen individuals are known as parents, and the next generation represents their children. Breeding is done through crossover operations on the parents with hope of taking the best of each parent to produce a more fit offspring. To avoid local optima, the children also have a chance of exhibiting mutations. Mutations are minor modifications to the individual so that its structure is changed. Using routes as individuals, a mutation represents altering the visiting order of a subset of the given set of points, whereas a crossover operation generally operates on the entire set. Algorithm 4 represents a generic genetic algorithm, as adopted from [14] and [15], when applied to the TSP.

2.7 K-means clustering

Let $\Lambda = (\lambda_1, \ldots, \lambda_n)$ be a given set of points. In case the camera vehicle is approaching its maximum travelling distance, meaning that it cannot visit all given points before reaching its limit, the route planner must determine which points of Λ that are ignored until the vehicle has reset. Clustering is a common approach of classifying data sets by partitioning the data points into clusters. The k-means algorithm iteratively assigns the points to a cluster based on the Euclidean distance between each point and the centers of the clusters [20]. Once all points have been assigned to clusters, the center of each cluster is updated to fit its newly assigned points. This process repeats until convergence, which happens when the centers of the clusters no longer changes between iterations [20]. See Algorithm 5 for pseudocode of k-means clustering.

Algorithm 4: Genetic algorithm

input : $\Lambda = (\lambda_1, \ldots, \lambda_n)$
output: $P = (p_1, \ldots, p_n)$

$population \leftarrow$ set of randomized initial permutations of Λ
$fitness \leftarrow$ array of the fitness of all individuals in $population$
$\gamma_m \leftarrow$ mutation rate

for $i \leftarrow 1$ **to** *generations* **do**
  $parent1 \leftarrow$ roulette wheel selection based on $fitness$
  $parent2 \leftarrow$ roulette wheel selection based on $fitness$
  $children \leftarrow$ breed($parent1, parent2$)
  for *each child in children* **do**
    $\gamma \leftarrow$ random value between 0 and 1
    if $\gamma < \gamma_m$ **then**
      $child \leftarrow$ mutate($child$)
    end
  end
  $population \leftarrow children$
  $fitness \leftarrow$ array of the fitness of all individuals in $population$
end

$P \leftarrow$ the most fit individual in $population$

Algorithm 5: K-means clustering

input : $\Lambda = (\lambda_1, \ldots, \lambda_n), k$
output: $ClusterIDs = (id_1, \ldots, id_n)$

$ClusterCenters \leftarrow$ initialize k cluster centers

while *ClusterCenters changes between iterations* **do**
  /* Assign all points in Λ to a cluster */
  for $i \leftarrow 1$ **to** n **do**
    $id_i \leftarrow$ index of the cluster with the lowest Euclidean distance between its center and λ_i
  end
  /* Update the cluster centers */
  for $i \leftarrow 1$ **to** k **do**
    $ClusterCenters(i) \leftarrow$ mean of all points assigned to cluster i
  end
end

3
Implementation

This chapter presents how each algorithm was implemented to fit the investigated scenario. Note that the camera vehicle has a physical position and also a desired goal, which is to exit the search area. In order for any of the algorithms, which aim to solve the DTSP or parts of it, to be optimal, it must take into account the cost of travelling from the current position of the camera vehicle to the first MLO as well as the cost of travelling from the last MLO to the desired goal position. This modification is what makes an algorithm fit the investigated scenario.

3.1 Dubins path

An implementation of the Dubins path was found on MathWorks [21]. It was modifiable and showed better performance than the built-in DUBINSCONNECTION function in Matlab. The implementation was modified in order to allow the option of only calculating the cost of the path and not generate the entire path to reduce the computation time when only the cost is of interest.

3.2 Nearest neighbour

The current position of the camera vehicle is set as the starting point for the nearest neighbour algorithm. Because of its simplicity, it was implemented using an angular-metric cost function instead of Euclidean distances. The idea is taken from [1], where it was proposed to use the angular-metric cost function for better results for the DTSP. Figure 3.1 showcases how the angle for the angular-metric cost function is calculated.

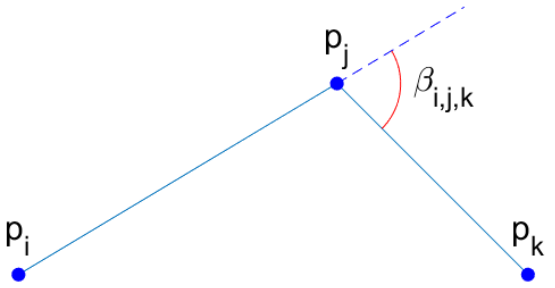


Figure 3.1: *Angle for the angular-metric cost function [1].*

Using Figure 3.1 as an example, the cost of adding p_k to the path would be

$$cost(p_i, p_j, p_k) = L_E([p_j, p_k]) + \beta_{i,j,k} \cdot \rho_{\min} \tag{3.1}$$

where L_E is defined in (2.4) and $\rho_{\min}$ is the minimum turn radius of the vehicle. Note that this method is not used for any other algorithm in this thesis because of its higher computational complexity over Euclidean distances. The alternating algorithm is used to produce headings once the nearest neighbour algorithm has resolved the visiting order of the MLOs.

3.3 Simulated annealing

An implementation of simulated annealing was found on MathWorks [13] and was modified to fit the investigated scenario. Instead of trying randomized permutations of the given set of points as presented in Section 2.4, it uses mutation-like methods in attempt of improving the path. The mutations that were implemented for simulated annealing are presented in Table 3.1. The cost of a permutation is calculated using Euclidean distances as defined in (2.4). The amount of iterations is set to $5n$, where n is the amount of MLOs, and the amount of subiterations is set to n. The alternating algorithm is used to produce headings once simulated annealing has resolved the visiting order of the MLOs.

3.4 Gradient of the seabed

In order for the camera vehicle to avoid travelling towards the inclination, the possible headings are limited accordingly. Each MLO is represented by a location (x, y) and a gradient (α, β), where β is the inclination and α the direction at which the seabed is inclined. For this thesis, β is limited to the interval $[0, \frac{\pi}{4}]$.

In Figure 3.2, the gradient is represented by the red arrow, here $\alpha = \frac{\pi}{2}$ and $\beta = \frac{\pi}{8}$. The length of the red arrow is negatively proportional to β, meaning that the

Table 3.1: *Simulated annealing mutations.*

Mutation	Description
Reversion	Select two random indices i_1 and i_2, reverse the visiting order of all MLOs between i_1 and i_2.
Insertion	Select two random indices i_1 and i_2, move the MLO at index i_1 and place it after the MLO at index i_2.
Swap	Select two random indices i_1 and i_2, swap the MLOs at index i_1 and i_2.

shorter the arrow, the more inclined the gradient is.

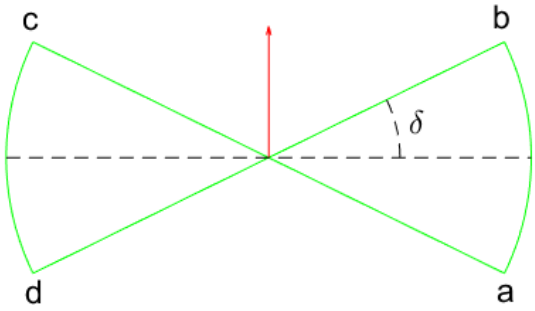


Figure 3.2: *The heading limits caused by the gradient (red arrow) as seen from above.*

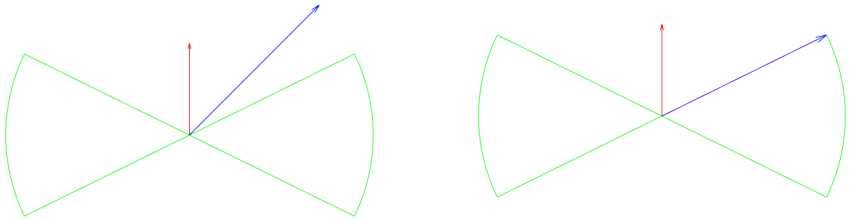
The limits are set according to

$$\begin{aligned} a &= \alpha - \pi/2 - \delta, \\ b &= \alpha - \pi/2 + \delta, \\ c &= \alpha + \pi/2 - \delta, \\ d &= \alpha + \pi/2 + \delta, \end{aligned} \tag{3.2}$$

where $\delta = \frac{\pi}{2} \cdot e^{-2\beta}$ is used for this thesis.

3.5 Alternating algorithm

Once the alternating algorithm has resolved headings for the DTSP, the headings are compared to the heading limits. If a chosen heading is outside of the limits, it is set to the nearest limit. Figure 3.3 shows an example of this.



(a) *A heading chosen outside of the limits.* ***(b)*** *The adjusted heading.*

Figure 3.3: *An example of a heading being adjusted to fit the heading limits.*

3.6 Genetic algorithm

The implemented genetic algorithm solves the DTSP as a single problem, which means it takes into account both the visiting order as well as the headings for the given set of MLOs. The Euclidean distance is not affected by the headings, which means it cannot be used as a metric to determine the performance of a path. Instead, the Dubins cost is used, which is more accurate but more computational heavy. The fitness of each individual is calculated as

$$fitness = \frac{1}{L_D} \tag{3.3}$$

where L_D is defined in (2.3). The shorter the path, the higher the fitness.

The 2-point crossover [15] is used as breeding method. It randomly selects two indices i_1 and i_2 and produces a child that consists of every state between the two indices from the first parent, and the rest from the second parent. For example, let

$$parent1 = [s_1, s_2, s_3, s_4, s_5, s_6]$$

correspond to the first parent and

$$parent2 = [s_3, s_4, s_5, s_1, s_2, s_6]$$

to the second parent. Using $i_1 = 2$ and $i_2 = 4$ the child of $parent1$ and $parent2$ would be

$$[s_5, s_2, s_3, s_4, s_1, s_6].$$

Here $[s_2, s_3, s_4]$ were picked from the first parent and $[s_5, s_1, s_6]$ from the second parent. The respective orders were kept during the crossover process, which is why this method is occasionally called the order crossover as in [15]. The amount of generations for this implementation is set to $10n$ and the population size is set

to 8 individuals.

The mutations used for the genetic algorithm are shown in Table 3.2.

Table 3.2: *Genetic algorithm mutations.*

Mutation	Description
Reversion	Select two random indices i_1 and i_2, reverse the visiting order of all MLOs between i_1 and i_2 including their headings.
Insertion	Select two random indices i_1 and i_2, move the MLO at index i_1 and place it after the MLO at index i_2.
Swap	Select two random indices i_1 and i_2, swap the MLOs at index i_1 and i_2.
Rotation	Select a random index i, randomly select a new heading $\theta_i \in \{[a, b] \cup [c, d]\}$ for the MLO at index i.

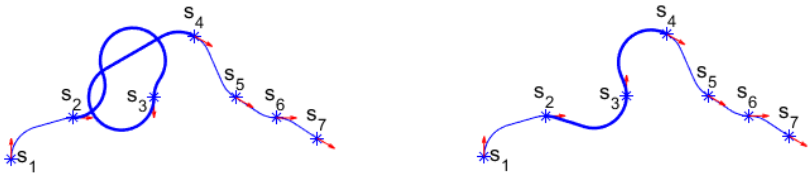
Another important part of the genetic algorithm is the choice of initial population. A popular method is to use a completely randomized set of individuals as was done by both [14] and [15]. However, for the implementation of the genetic algorithm in this thesis, an initial individual is produced by the nearest neighbour algorithm along with the alternating algorithm. The individual is then randomly mutated to produce more individuals until the initial population has been filled.

3.7 Dubins simulated annealing

As will be shown in Chapter 4, the computational complexity skyrockets when using Dubins costs over Euclidean distances. Since mutations only affect subsets of the set of points and headings, only a few edges actually change cost during a mutation. With this in mind, a modified version of simulated annealing was implemented, in which the costs of all edges in the path are stored instead of just the total path length. Through this modification, when performing a mutation, at most four edge costs have to be calculated instead of recalculating the cost of every edge in the path. The modified version of simulated annealing was named Dubins simulated annealing (DSA), and was implemented with the specific purpose of decreasing the amount of cost calculations in order to limit the computational complexity when using the Dubins cost function. Reversion, insertion, swap and rotation are used as mutations for the DSA. The amount of iterations is set to $5n$ and the amount of subiterations is set to n.

3.7.1 Rotation

Figure 3.4 showcases an example of the rotation mutation, in which the state s_3 is rotated and the two affected edges are highlighted. Note that the red arrows represent the headings. Clearly, the edges not highlighted remain constant throughout the mutation and do therefore not need to be recalculated. Since the original edges are all stored from the previous iteration, they do not have to be recalculated. Thus, only two edges need to be computed during a rotation mutation, no matter the amount of MLOs.



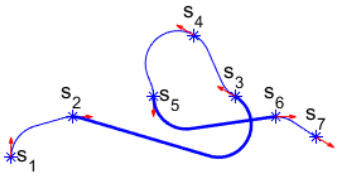
(a) *The original path.* ***(b)*** *The mutated path.*

Figure 3.4: *An example of the rotation mutation. Here the third state s_3 is rotated.*

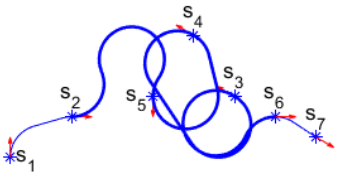
The visiting order of the states remain unchanged during the rotation mutation.

3.7.2 Reversion

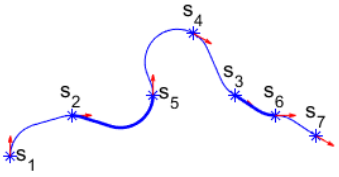
Figure 3.5 showcases an example of the reversion mutation. Here the visiting order of all states from s_3 to s_5 is reversed. Figure 3.5b shows the result of the reversion if only the visiting order is reversed and not the headings. It is evident that the Dubins cost is asymmetric, meaning that the cost differs depending on which way you traverse the edge. However, also reversing the headings during the mutation, which is equivalent to rotating the headings by π, removes the asymmetric behaviour as seen in Figure 3.5c. Similarly to rotation, reversion only requires two edges to be computed when both the visiting order and the headings are reversed as indicated by the highlighted edges in Figure 3.5a and Figure 3.5c.



(a) The original path.



(b) Visiting order reversed.



(c) Visiting order and headings reversed.

Figure 3.5: *An example of the reversion mutation. Here the reversion mutation is performed on the subset* (s_3, s_4, s_5).

The visiting order of the states changes from

$$[s_1, s_2, s_3, s_4, s_5, s_6, s_7]$$

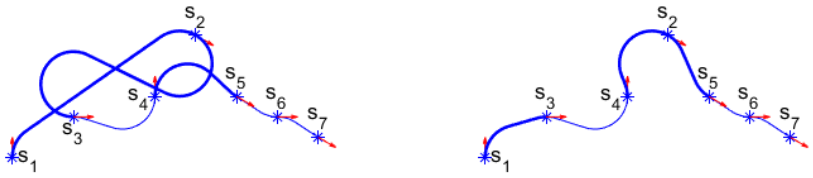
to

$$[s_1, s_2, s_5, s_4, s_3, s_6, s_7]$$

during the reversion mutation for this example.

3.7.3 Insertion

An example of the insertion mutation is shown in Figure 3.6. Here the second state s_2 is taken out of the visiting order and inserted after the fourth state s_4. Insertion requires three edges to be computed as indicated by the highlighted edges.



(a) The original path. *(b)* The mutated path.

Figure 3.6: *An example of the insertion mutation. Here the second state s_2 is inserted after the fourth state s_4.*

The visiting order of the states changes from

$$[s_1, s_2, s_3, s_4, s_5, s_6, s_7]$$

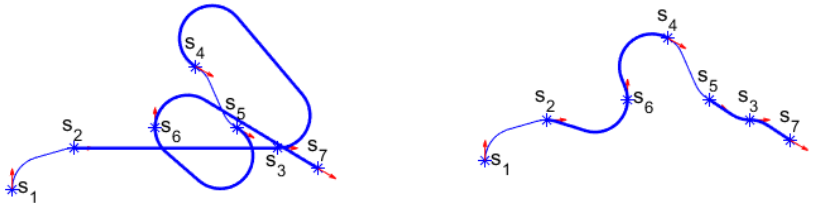
to

$$[s_1, s_3, s_4, s_2, s_5, s_6, s_7]$$

during the insertion mutation for this example.

3.7.4 Swap

Figure 3.7 shows an example of the swap mutation. Here the visiting order of the third state s_3 and the sixth state s_6 is swapped. The swap mutation requires four edges to be computed as indicated by the highlighted edges.



(a) The original path. *(b)* The mutated path.

Figure 3.7: *An example of the swap mutation. Here the visiting order of the third state s_3 and the sixth state s_6 is swapped.*

The visiting order of the states changes from

$$[s_1, s_2, s_3, s_4, s_5, s_6, s_7]$$

to

$$[s_1, s_2, s_6, s_4, s_5, s_3, s_7]$$

during the swap mutation for this example.

3.8 Simulation setup

In order to run the simulation, the size of the search area is specified along with the amount of MLOs that should be randomly generated in the area. Different vehicle sizes can be tested by varying their velocities, minimum turn radii and maximum travelling distances.

The vehicles are deployed from a ship located outside of the search area. The location of the ship is referred to as the origin of the vehicles. During a simulation, the AUV62-MR follows the lawnmower pattern shown previously in Figure 1.1 and detects MLOs that are in range of the SAS sensors. The SAS sensors are specified to have a range of 100 m to each side of the AUV62-MR. Once the vehicle exits the search area, it transmits all the MLOs that have been detected so far to the camera vehicle. Next, the camera vehicle calculates its optimal path and starts scanning. Meanwhile, the AUV62-MR continues its search until the entire search area has been scanned with SAS, at which point it returns to its origin. Every time it exits the search area, all the latest detected MLOs are transmitted to the camera vehicle.

In order to minimize the risk of the camera vehicle entering any part of the search area that has not yet been scanned by the AUV62-MR, it uses the last lower entry point of the AUV62-MR as entry point to the search area as indicated by the black cross in Figure 3.8. When the camera vehicle has finished scanning all its available MLOs and awaits more MLOs, it returns to its origin in order to minimize the time it spends in the search area and uses the latest lower entry point of the AUV62-MR as exit point.

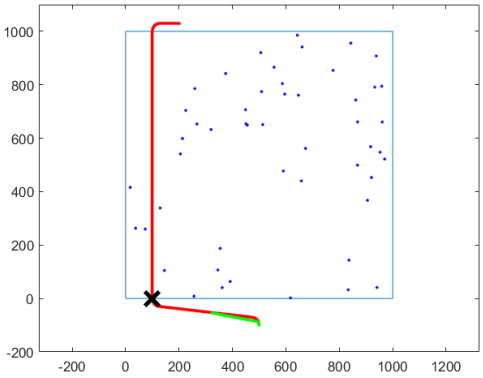


Figure 3.8: *The first entry point to the search area.*

The AUV62-MR is represented by the red path and the camera vehicle by the green path. Figure 3.9 shows the second entry point to the search area as indicated by the black cross. Note that the first entry point was also used as exit point for the camera vehicle.

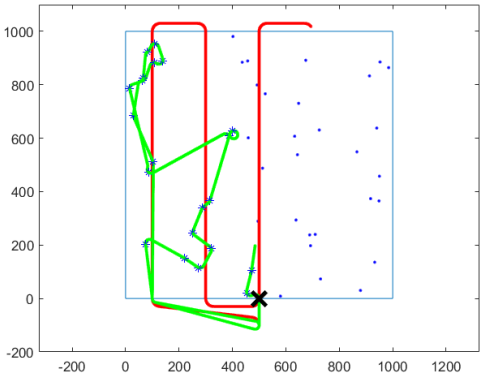


Figure 3.9: *The second entry point to the search area.*

The simulation is finished once all MLOs have been scanned and both vehicles have returned to the ship.

3.9 K-means clustering

The Matlab built-in function KMEANS is used as a clustering method. Once the route planner realizes that the camera vehicle cannot scan all acquired MLOs

before reaching its distance limit, it determines which MLOs to scan by dividing the MLOs into two clusters. If the vehicle can scan all MLOs in one of the clusters, it will do so. Otherwise, the MLOs are divided into three clusters. The MLOs are iteratively divided into at most four clusters in hope of finding a cluster of MLOs that the vehicle can scan on its way to its origin where it can reset. If no cluster is found, the vehicle directly returns to its origin. Figure 3.10 showcases an example in which the k-means clustering algorithm was used in the simulation framework.

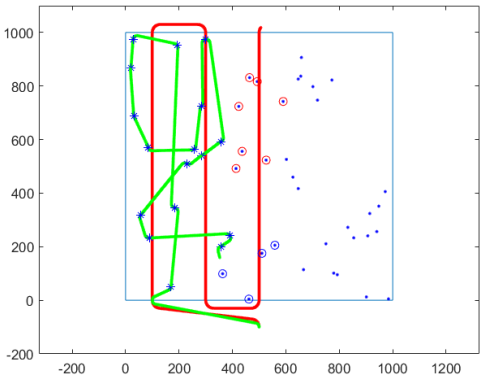


Figure 3.10: *An example of how the k-means clustering algorithm determined which MLOs to skip (red circles) and which to scan (blue circles).*

4

Performance evaluation

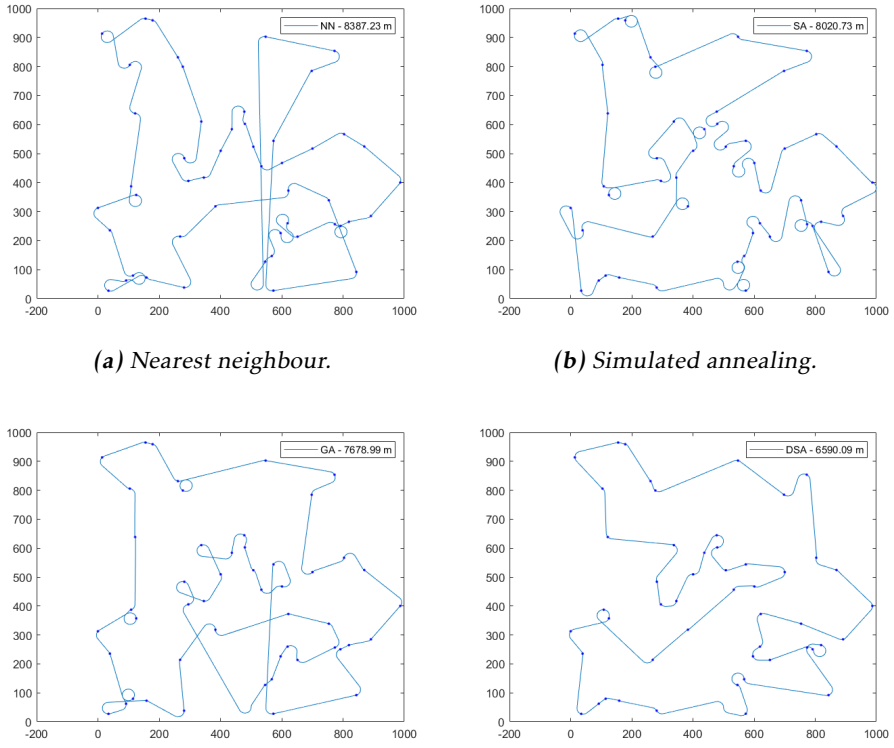
The algorithms for solving the DTSP are evaluated first in a static environment, in which the vehicle is given a full set of randomized MLOs and plans the shortest path through all of them by applying each of the algorithms. The algorithms are then applied to the investigated scenario and evaluated according to the runtime of the simulation as well as the distance travelled by the camera vehicle.

The performances of two different vehicle sizes are simulated and evaluated based on their runtimes and travelled distances. The behaviour of the camera vehicle is then evaluated through its average amount of trips during a simulation. Finally, parallel versus sequential execution of phase 2 and phase 3 is evaluated based on their average runtimes and the total distance travelled by the vehicles.

4.1 Static environment

For an initial evaluation of the algorithms, they were used to find the shortest path through a random set of MLOs. Here, a vehicle with turn radius $\rho_{\text{min}} = 20$ m was used and the size of the area was set to 1 km^2. An example using 50 MLOs is shown in Figure 4.1.

For this example, DSA found a path that was roughly 14 % shorter than the path found by the genetic algorithm, and 21 % shorter than the path found by the nearest neighbour algorithm.



(a) Nearest neighbour.

(b) Simulated annealing.

(c) Genetic algorithm.

(d) Dubins simulated annealing.

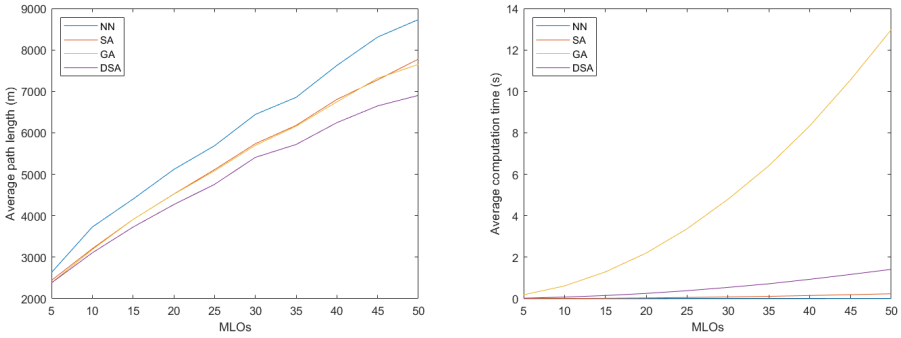
Figure 4.1: *The result of each algorithm in a static environment with 50 MLOs.*

4.1.1 Average results of multiple simulations in a static environment

Trivially, one example of the algorithms is not enough for proper evaluation. The amount of MLOs was therefore varied from 5 to 50 with a step size of 5 and each amount of MLOs was randomly generated 30 times. The average path lengths and computations times of the 30 simulations are shown in Figure 4.2.

With only five MLOs in the area, the difference between the algorithms appears almost negligible. As the amount of MLOs increases, the Dubins simulated annealing algorithm finds relatively shorter and shorter paths.

Including both the iterations and subiterations of DSA, it iterates a total of $5n^2$ times, where n is the amount of MLOs. Each iteration tries to improve the solution through a mutation, which leads to a total of $5n^2$ solutions having been compared once the algorithm has finished. For each generation, the genetic algo-



(a) *Average path lengths.* ***(b)*** *Average computation times.*

Figure 4.2: *Average performances of 30 simulations in a static environment.*

rithm produces a population of 8 individuals. In total, $8 \cdot 10n = 80n$ individuals are therefore produced, which is equivalent to $80n$ solutions.

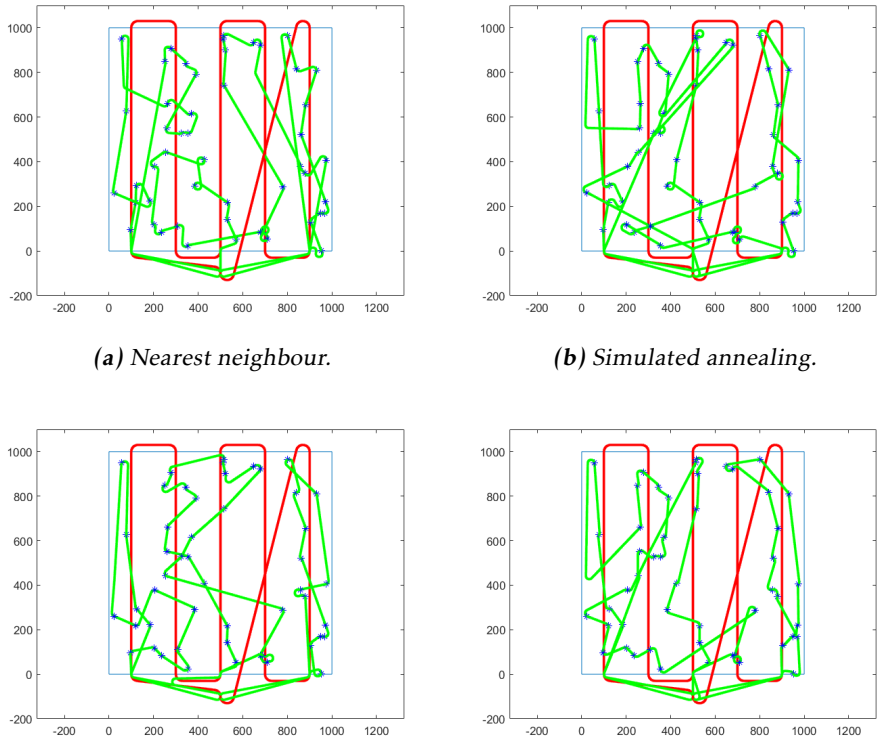
The fitness of an individual for the genetic algorithm requires n edge costs to be calculated, whereas the modified mutations limit the amount of edge costs of DSA to at most 4 calculations, as described in Section 3.7. The total amount of cost calculations for the genetic algorithm is therefore $80n^2$, whereas the total amount of cost calculations for DSA is less than or equal to $20n^2$.

At 50 MLOs, the genetic algorithm computes 200 000 cost calculations and compares 4 000 solutions. In comparison, the Dubins simulated annealing algorithm computes at most 50 000 cost calculations and compares 12 500 solutions at 50 MLOs. This shows that the modified mutations are a major improvement to the originals, and explains why DSA outperforms the genetic algorithm.

4.2 Simulation framework

The algorithms are also evaluated based on their performance when applied to the investigated scenario in the simulation framework. For this evaluation, a vehicle with minimum turn radius $\rho_{\min} = 30$ m and maximum velocity $v_{\max} = 5$ m/s was used as the AUV62-MR. Its maximum travelling distance was set to infinity. The minimum turn radius of the camera vehicle was set to $\rho_{\min} = 15$ m, its maximum velocity to $v_{\max} = 8$ m/s, and its maximum travelling distance to 5 km. The size of the search area was set to 1 km^2. An example of such a simulation for each algorithm using 50 MLOs is shown in Figure 4.3. The runtime and distance travelled by the camera vehicle during each simulation are presented in Table 4.1.

The result indicates that DSA performs the best when applied to the investigated scenario. Evidently, the distance travelled by the camera vehicle is linearly pro-



(a) Nearest neighbour.

(b) Simulated annealing.

(c) Genetic algorithm.

(d) Dubins simulated annealing.

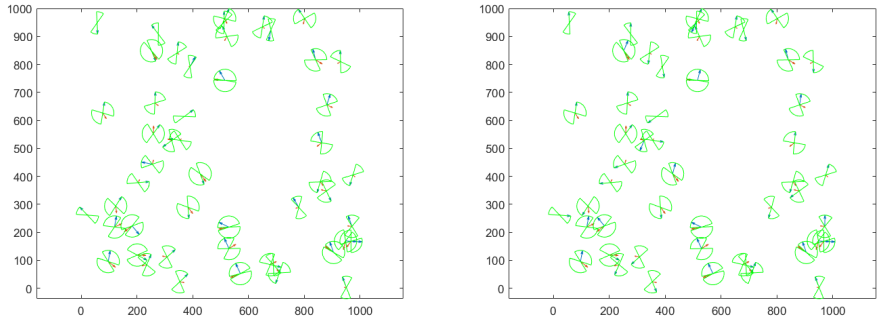
Figure 4.3: *The result of each algorithm when applied to the investigated scenario with 50 MLOs.*

Table 4.1: *The runtime and distance travelled by the camera vehicle during each simulation.*

Algorithm	**Runtime**	**Distance travelled**
Nearest neighbour	00:30:58	12566 m
Simulated annealing	00:31:20	12740 m
Genetic algorithm	00:30:04	12127 m
Dubins simulated annealing	00:28:48	11531 m

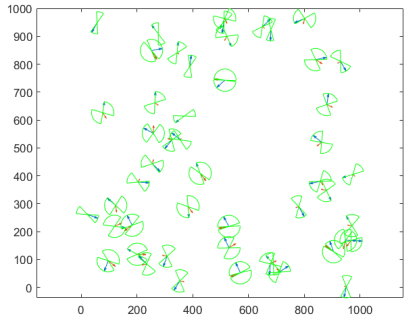
portional to the runtime of the simulation, which is trivially explained by its constant velocity.

While the chosen headings shown in Figure 4.4 may not provide any new results, it does confirm that each heading has been chosen within the acceptable limits. This guarantees that the gradients of the seabed are accounted for and allows the

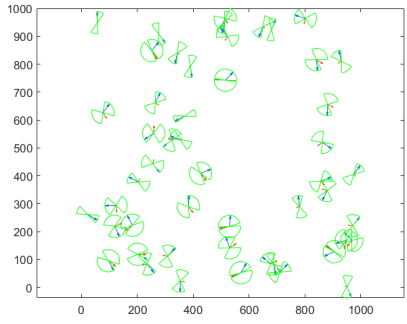


(a) *Nearest neighbour.*

(b) *Simulated annealing.*



(c) *Genetic algorithm.*



(d) *Dubins simulated annealing.*

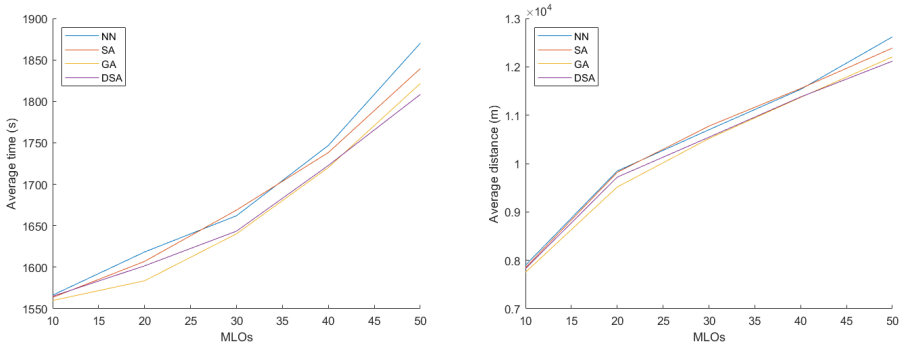
Figure 4.4: *The chosen headings when each algorithm was applied to the investigated scenario.*

camera vehicle to avoid unnecessary risks of hitting the seabed.

4.2.1 Average results of multiple simulations when applied to the investigated scenario

The amount of MLOs was varied from 10 to 50 with a step size of 10. Each amount of MLOs was simulated 30 times. The average runtimes and distances travelled are shown in Figure 4.5.

This evaluation contradicts the previous indication that DSA is the obviously better performing algorithm. The differences in performance between the algorithms here are almost negligible. At 50 MLOs, DSA shows a very slight improvement over the other algorithms, which explains the indication of the previously shown example.



(a) *Average runtimes.* ***(b)*** *Average distances travelled.*

Figure 4.5: *Average performances of 30 simulations when applied to the investigated scenario.*

4.3 Vehicle sizes

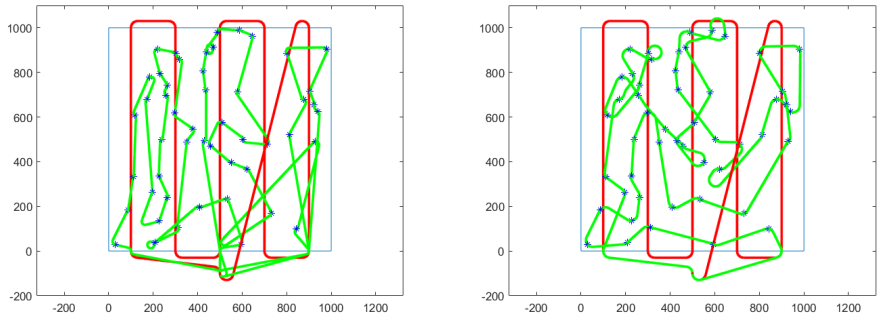
In order to investigate what torpedo size works best for the camera vehicle, the simulation framework was used to evaluate the performance of two different vehicle sizes: one small and one large. A smaller vehicle means a shorter turn radius and maximum travelling distance. It also means a higher velocity because the vehicle would be less valuable making the higher velocity tolerable. For this evaluation, the Dubins simulated annealing algorithm was used as route-planning algorithm. The size of the search area was set to 1 km^2.

The small vehicle was specified to have a turn radius of 15 m, a velocity of 8 m/s and a maximum travelling distance of 5 km, whereas the large vehicle was specified to have a turn radius of 30 m and a velocity of 5 m/s. The large vehicle was assumed to be capable of completing the search without the need of resetting, and so its maximum travelling distance was set to infinity. Figure 4.6 shows a simulation example using 50 MLOs for each vehicle size and Table 4.2 presents the corresponding runtimes and distances travelled by the camera vehicle.

Table 4.2: *The runtime and distance travelled by the camera vehicle during the evaluation of vehicle sizes.*

Vehicle size	Runtime	Distance travelled
Small	00:30:45	12468 m
Large	00:38:09	10011 m

The small vehicle appears to outperform the larger vehicle through a 20 % shorter runtime, but the distance travelled is almost 25 % longer for the small vehicle.



(a) *Small vehicle.* ***(b)*** *Large vehicle.*

Figure 4.6: *The result of the simulation examples with a small and a large camera vehicle with 50 MLOs.*

4.3.1 Average results of multiple simulations with a small and a large vehicle

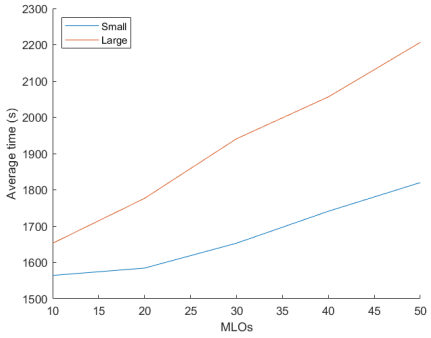
To properly test the two vehicle sizes, 10 to 50 MLOs were randomly generated with a step size of 10. Each amount of MLOs was simulated 20 times. The average runtimes and distances travelled of the 20 simulations are shown in Figure 4.7 along with the quotients of the results. The quotients are the results of dividing the runtimes and distances of the small vehicle with the runtimes and distances of the large vehicle.

The quotients show that as the amount of MLOs increases in the area, the small vehicle becomes increasingly better compared to the large vehicle reaching a 17 % runtime reduction on average at 50 MLOs. The small vehicle does, however, bring an increase in distance it needs to travel, which could potentially cause an increase in fuel costs. At 10 MLOs in the search area, the small vehicle travels almost 30 % longer than the large vehicle, but reduces the runtime by only 5 %.

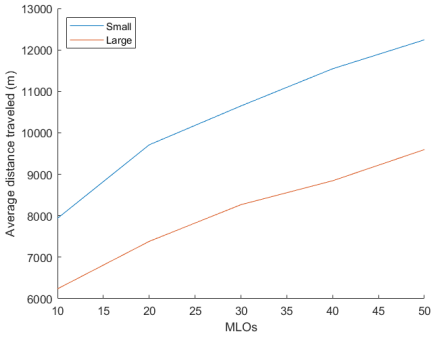
4.4 Trips

The small camera vehicle and Dubins simulated annealing were used to simulate the scenario 10 times for each amount of MLOs, while recording the average number of trips and the average trip length of the camera vehicle. A trip is referred to as the path the camera vehicle takes in between having been at its origin. Starting at the origin, entering the search area and returning to the origin is defined as one trip. The average number of trips and the average trip length for each amount of MLOs are shown in Figure 4.8.

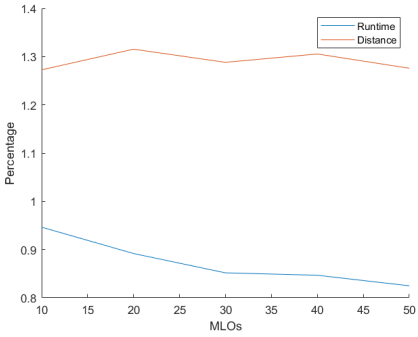
An average trip length of roughly 2500 m is only half of the maximum travelling distance of the camera vehicle. In theory, the average amount of trips could there-



(a) Average runtimes.



(b) Average distances travelled.



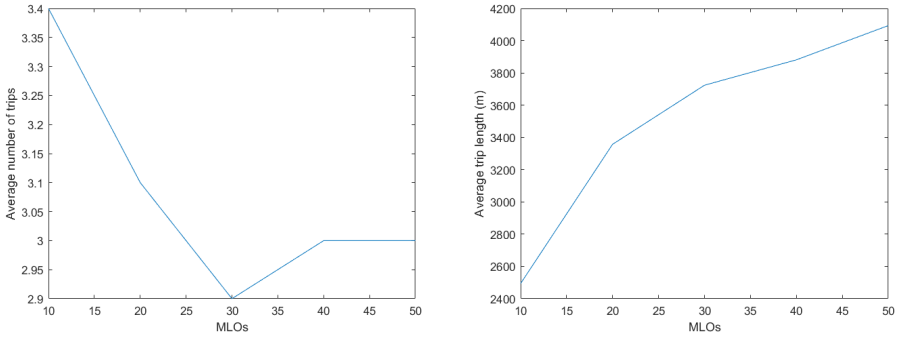
(c) Quotients of small/large.

Figure 4.7: *Average performances of 20 simulations with a small and a large camera vehicle.*

fore be halved at 10 MLOs. It is evident that the camera vehicle on average takes too short and too many trips, especially when the amount of MLOs in the search area is low. The average trip length increases as the amount of MLOs increases. The expected behaviour would have been for the average number of trips to also increase as the amount of MLOs increases, but surprisingly this appears not to be the case.

4.5 Parallel versus sequential execution

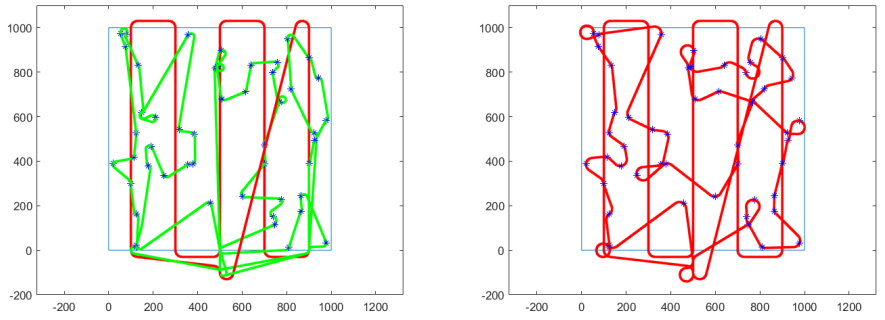
Here, the AUV62-MR was specified to have a minimum turn radius of 30 m, velocity of 5 m/s and no maximum travelling distance. The camera vehicle was set to have a minimum turn radius of 15 m, velocity of 8 m/s and maximum travelling distance of 5 km. The size of the search area was set to 1 km^2 and the Dubins simulated annealing algorithm was used as route-planning algorithm. A simula-



(a) *Average number of trips.* ***(b)*** *Average trip length.*

Figure 4.8: *Average trips of 10 simulations.*

tion example using 50 MLOs for each of the algorithms is shown in Figure 4.9 and Table 4.3 presents the corresponding runtimes and total distances travelled. Note that for the parallel execution, the total distance travelled is the sum of the distances travelled by both vehicles.



(a) *Parallel execution.* ***(b)*** *Sequential execution.*

Figure 4.9: *The result of the simulations with parallel and sequential execution.*

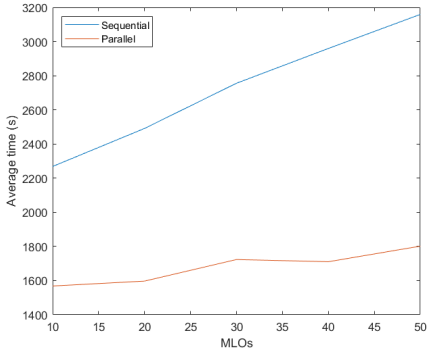
Table 4.3: *The runtime and total distance travelled during parallel and sequential simulations.*

Execution	Runtime	Total distance travelled
Parallel	00:30:10	19884 m
Sequential	00:53:10	15952 m

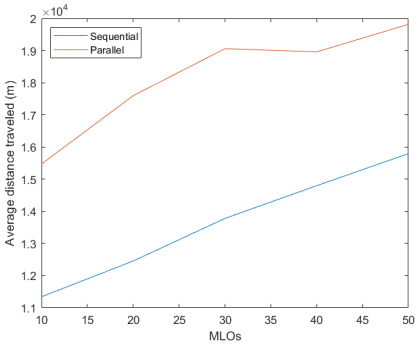
The simulation examples provide an initial indication that parallel execution significantly reduces the runtime of the mine reconnaissance process. It does, however, increase the total distance travelled by the vehicles, which could potentially cause an increase in fuel costs.

4.5.1 Average results of multiple simulations with parallel and sequential execution

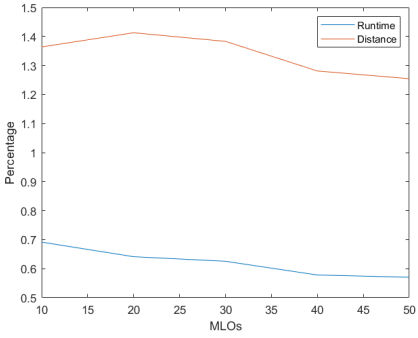
To properly test parallel versus sequential execution, 10 to 50 MLOs were randomly generated with a step size of 10. Each amount of MLOs were simulated 10 times. The average runtimes and distances travelled are shown in Figure 4.10 along with the quotients of the results. The quotients are the results of dividing the parallel runtimes and distances with the sequential runtimes and distances.



(a) Average runtimes.



(b) Average distances travelled.



(c) Quotients of parallel/sequential.

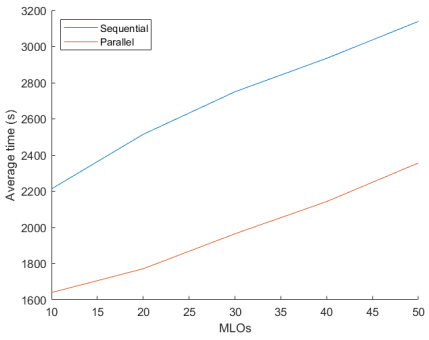
Figure 4.10: *Average performances of 10 simulations with parallel and sequential execution.*

The results confirm the previous indication that parallel execution significantly reduces the runtime. The quotients show that an impressive 43 % runtime re-

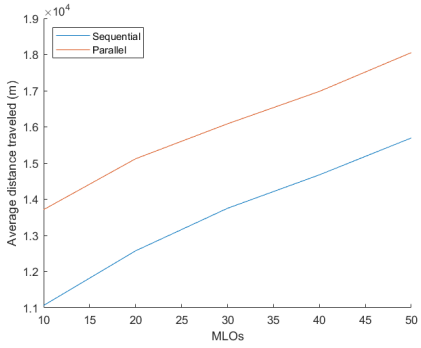
duction is gained on average at 50 MLOs. While the runtime appears to be continuously reduced as the amount of MLOs is increased, the increase in distance travelled is halted and even appears to decrease. It maxes out at an approximate 40 % increased distance travelled at 20 MLOs, but is less than a 26 % increase at 50 MLOs.

4.5.2 Slower camera vehicle

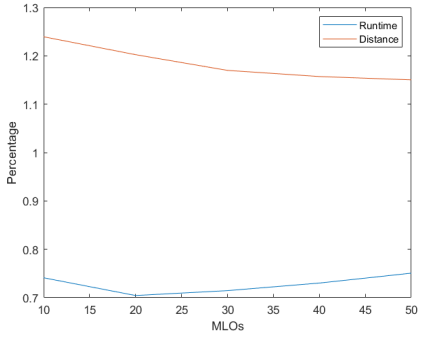
Parallel versus sequential execution is furthermore evaluated based on their performances when the camera vehicle is limited to the same velocity as the AUV62-MR. The velocities of both vehicles were therefore set to 5 m/s. Each amount of MLOs was simulated 10 times. The average runtimes and average distances travelled are shown in Figure 4.11 along with the quotients of the results.



(a) Average runtimes.



(b) Average distances travelled.



(c) Quotients of parallel/sequential.

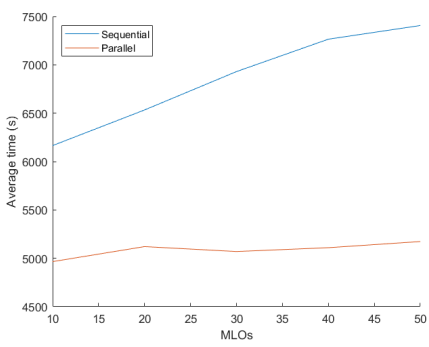
Figure 4.11: *Average performances of 10 simulations with a slower camera vehicle.*

While the runtime reduction is evidently slightly less when the camera vehicle is limited to the same velocity as the AUV62-MR, the increase in distance trav-

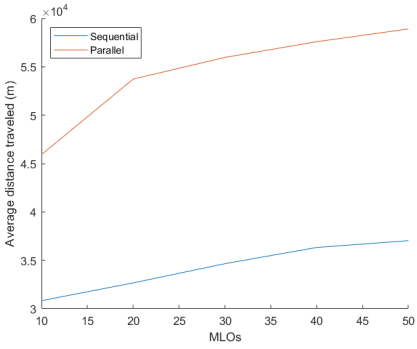
elled by the vehicles is significantly reduced. The total distance travelled is only 15 % longer on average during parallel execution over sequential execution at 50 MLOs, but still a 25 % runtime reduction is gained.

4.5.3 Larger search area

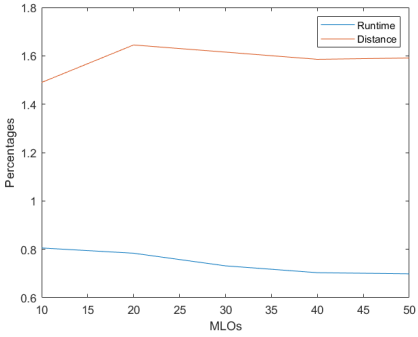
Parallel versus sequential execution is also evaluated based on their performances in a larger search area. Here the size of the search area was set to 4 km^2. The velocity of the camera vehicle was set to 8 m/s and its maximum travelling distance was increased to 8 km. 10 to 50 MLOs were randomly generated with a step size of 10 and each amount of MLOs was simulated 10 times. The average runtimes and average distances travelled are shown in Figure 4.12 along with the quotients of the results.



(a) Average runtimes.



(b) Average distances travelled.



(c) Quotients of parallel/sequential.

Figure 4.12: *Average performances of 10 simulations in a larger search area.*

As the search area size is quadrupled, and the maximum travelling distance of the camera vehicle is increased by 60 %, evidently parallel execution requires a significantly longer total distance to be travelled by the vehicles. A roughly 60 %

longer distance must be travelled by the vehicles, whereas the runtime reduction has not increased compared to the smaller search area. This result was unexpected, but can be explained by the fact that the camera vehicle must perform more trips. Each trip adds a certain distance to and from the origin that is not spent on actually scanning MLOs.

5
Discussion

In this chapter, the performance evaluation presented in Chapter 4 is discussed. Firstly, the performances of the route-planning algorithms are considered. Secondly, the evaluated vehicle sizes are discussed, and finally parallel versus sequential execution is analysed.

5.1 Route planning of the camera vehicle

The evaluation of the route-planning algorithms in the simulation framework showed that there is no gain in using an algorithm that finds shorter paths on average. Since the camera vehicle returns to its origin once it has scanned all its known MLOs, the shorter the path through those MLOs is, the faster the vehicle will return to its origin. In case the vehicle is returning to its origin when more MLOs are received, the vehicle has potentially travelled an unnecessary distance towards its origin if it is still not approaching its maximum travelling distance. This behaviour causes an algorithm that finds shorter paths to occasionally perform worse than an algorithm that finds longer paths.

If the camera vehicle would be allowed to stay within the search area while waiting for more MLOs, it could circulate in position at a reduced velocity, decreasing the unnecessary distance it travels. This would, however, introduce the problem of determining how close to its maximum travelling distance it needs to be before returning to its origin. Another possibility is to introduce the velocity of the camera vehicle into the route-planning problem, instead of keeping it constant. With a reduced velocity during the first few scans, it would take longer before the vehicle starts returning to its origin and allow for more MLOs to be received in time. Since the current position and path of the AUV62-MR is always

known to the camera vehicle, in theory it could balance its velocity to reach the last MLO on its current path just in time as the AUV62-MR exits the search area. This would completely remove the unnecessary distance it travels towards its origin. The camera vehicle could also stay docked at the ship until the AUV62-MR has completed more than just the first strip of the lawnmower pattern, allowing a larger amount of MLOs to be initially transmitted and therefore takes longer time before the first time the camera vehicle has to return to its origin.

While the computation time was not accounted for in the route-planning problem in the simulation framework, the evaluation in the static environment showed that the genetic algorithm takes drastically longer time to compute than nearest neighbour, simulated annealing and Dubins simulated annealing. The computation time of the genetic algorithm reached as high as 13 seconds for 50 MLOs. With a velocity of 8 m/s, the camera vehicle would be 104 meters off its position before the path has been planned. This means that the planned path would originate from a point 104 meters behind the current position of the vehicle. In comparison, the Dubins simulated annealing algorithm only reached 1.4 seconds at 50 MLOs, which would correspond to being 11.8 meters off its position before the path has been planned. The average amount of trips at 50 MLOs was shown to be 3, which corresponds to an average of less than 20 MLOs per trip. At 20 MLOs, the Dubins simulated annealing algorithm averaged a 0.25 second computation time, corresponding to being 2 meters off position. The average computation time of the genetic algorithm at 20 MLOs corresponds to being 17.6 meters off position once the path has been planned.

Given that the camera vehicle still has MLOs to scan once it receives more MLOs, the route-planning algorithm could take into account the computation time by originating the new path from the next MLO in its current path instead of from the current position of the vehicle. Similarly, if the vehicle has no MLOs in its current path when it receives more MLOs, it could generate a path to the nearest MLO and while travelling to that MLO find the shortest path through the rest of the MLOs. This solution takes into account the computation time of the algorithms and allows the vehicle to compute a path without the risk of being off position once the path has been planned. However, if the computation time of the used route-planning algorithm is low, the distance the vehicle would be off position with would be short. By using the Dubins simulated annealing algorithm at 50 MLOs, the camera vehicle would on average be roughly 2 meters off position of the planned path, which seems low considering the average trip length of roughly 4100 meters. Whether being initially 2 meters off the planned path on average would total a longer travelled distance than taking the computation time into account in the route-planning problem, as suggested above, remains to be tested. At lower amounts of MLOs in the search area, the computation times of the route-planning algorithms are considered completely negligible with the exception of the genetic algorithm.

5.2 Size of the camera vehicle

Under the assumption that a small vehicle would be allowed to travel faster than a large vehicle, the performance evaluation showed that a 17 % runtime reduction is reached at 50 MLOs using the small vehicle over the larger vehicle. The large vehicle does, however, result in a shorter total distance travelled. A larger vehicle not only allows a larger battery, but it could also provide the opportunity to add more hardware such as more sensors. A small vehicle instead allows for easier deployment. An easier deployment could in turn require less people and therefore a reduced personnel cost. The difference of a maximum travelling distance of 5 km and having no maximum travelling distance was shown to be of great importance. An increase of nearly 30 % in distance travelled could correspond to a major increase in fuel costs.

5.3 Parallel versus sequential execution

Parallel execution seems to be a reasonable improvement over the current execution of phase 2 and phase 3 of the mine reconnaissance process. It was shown to reduce the runtime by a significant amount, but also to increase the total distance travelled. The dimensions of the search area were moderately tested, but a larger search area was indicated to decrease the gain of parallel execution. It is clearly important to consider the size of the search area when choosing parallel versus sequential execution as well as the size of the camera vehicle.

A concern from the initial evaluation was that the runtime reduction was caused mainly by the fact that the camera vehicle was allowed to travel faster than the AUV62-MR. It was, however, confirmed that parallel execution in itself reduces the average runtime of the mine reconnaissance process in relation to sequential execution when the vehicles were specified to travel with the same velocity.

An important gain of parallel execution is the fact that the SAS system on the AUV62-MR would no longer be at risk when performing phase 3 of the mine reconnaissance process, which was the initial goal. It would also allow for smaller and cheaper camera vehicles to further test phase 3 in real scenarios.

A runtime reduction could potentially lead to a decrease in personnel cost as well as provide the opportunity of removing mines faster from the sea. This thesis was, however, limited to civilian usage of mine reconnaissance, which is limited to detecting and identifying old mines from previous wars. If a war were to break out and new mines were dropped in the sea, clearly the importance of a low runtime would increase drastically.

The simulation framework specified the last entry point of the AUV62-MR as entry point to the search area for the camera vehicle. This was done to reduce the chance of the camera vehicle being detected by the SAS system of the AUV62-MR. It is still possible for the camera vehicle to be detected, but the frequency of this happening was not covered in this thesis. Since none of the MLOs of the camera

vehicle are located in range of the AUV62-MR, the only chance of being detected is when the camera vehicle makes a turn that happens to enter SAS range of the AUV62-MR just as it passes by. Other options could be investigated, such as restraining the movement of the camera vehicle.

Once the AUV62-MR is finished with phase 2, it is no longer scanning and the camera vehicle could be allowed to enter and exit the search area at any given point. It could also be allowed to enter and exit the search area in between all its previous entry points in order to potentially find a shorter path.

6
Conclusions

This chapter finalizes the thesis by answering the questions formulated in Section 1.3 and suggesting future work. Conclusions are drawn based on the performance evaluation in Chapter 4 and the discussion in Chapter 5.

6.1 Which route-planning algorithm is best suited for the camera vehicle?

The proposed Dubins simulated annealing algorithm clearly finds the shortest paths amongst the investigated algorithms. Its computation time is considered negligible for low amounts of MLOs, but it remains to be tested whether the computation time should be taken into account in the route-planning problem for high amounts of MLOs.

6.2 What torpedo size should the camera vehicle be based on?

A small vehicle base is shown to reduce the runtime of the mine reconnaissance process in relation to a large vehicle base. The runtime reduction comes at the cost of a significantly increased total distance travelled. A small vehicle also comes with easier deployment, which should be considered when choosing the torpedo size.

6.3 How does the performance of parallel execution of phase 2 and phase 3 compare to sequential execution?

Parallel execution is shown to significantly reduce the runtime of the mine reconnaissance process, reaching an impressive 43 % reduction compared to sequential execution. The SAS system on the AUV62-MR would no longer be at risk during phase 3 of the mine reconnaissance process, but it would potentially come at the cost of an increased fuel cost.

6.4 Future work

This thesis only scratches the surface of designing a camera vehicle. It provides a simulation framework for investigating different vehicle sizes and route-planning algorithms, and opens up for further testing of various decision-making choices. Instead of only being allowed to enter the search area at a specific entry point, the camera vehicle could be allowed to enter and exit the area anywhere in between two points, which would allow for smoother paths.

In case of even larger search areas than what was investigated in this thesis, multiple camera vehicles could potentially be used. Whether this provides any practical use is left for future work to investigate.

Endless amounts of algorithms exist that solve the DTSP, or parts of it, and because of time limits only a handful of algorithms were investigated in this thesis. The proposed Dubins simulated annealing algorithm did, however, outperform several algorithms from the literature, such as a genetic algorithm.

The acoustic communication between the vehicles was approximated as perfect in this thesis, but in reality it experiences measurement noise and potentially a limited range. It would require further investigations to determine its true effects on parallel execution.

Bibliography

[1] D. G. Macharet, A. A. Neto, V. F. da Camara Neto, and M. F. M. Campos. Nonholonomic path planning optimization for Dubins' vehicles. In *IEEE International Conference on Robotics and Automation*, pages 4208–4213, 2011.

[2] F. Magnusson. 40 000 sjöminor finns i Östersjön - deras jobb är att spränga dem, 2019. `https://mitti.se/nyheter/sjominor-ostersjon-spranga/`, (last accessed on 2020-08-12).

[3] Saab Dynamics. AUV62-MR. `https://saab.com/naval/underwater-systems/underwater-sensor-systems/auv62_mr/`, (last accessed on 2020-05-13).

[4] M. S. Wiig, T. R. Krogstad, and Ø. Midtgaard. Autonomous identification planning for mine countermeasures,. *IEEE/OES Autonomous Underwater Vehicles (AUV)*, pages 1–8, 2012.

[5] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,. *American Journal of Mathematics*, 79(3):497–516, 1957.

[6] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis. Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory*, pages 33–42, 1974.

[7] K. Savla, E. Frazzoli, and F. Bullo. Traveling salesperson problems for the Dubins vehicle,. *IEEE Transactions on Automatic Control*, 53(6):1378–1392, 2008.

[8] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. ISBN 9781450374644.

[9] L. Babel. New heuristic algorithms for the Dubins traveling salesman problem,. *J Heuristics*, 2020.

[10] K. Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] K. Savla, E. Frazzoli, and F. Bullo. On the point-to-point and traveling salesperson problems for Dubins' vehicle. In *Proceedings of the American Control Conference*, pages 786–791 vol. 2, 2005.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[14] Xin Yu and J. Y. Hung. A genetic algorithm for the Dubins traveling salesman problem. In *IEEE International Symposium on Industrial Electronics*, pages 1256–1261, 2012.

[15] K. D. Hansen and A. La Cour-Harbo. Waypoint planning with Dubins curves using genetic algorithms. In *European Control Conference (ECC)*, pages 2240–2246, 2016.

[16] Matlab. kmeans. `https://se.mathworks.com/help/stats/kmeans.html`, (last accessed on 2020-08-12).

[17] A. Sadeghi and S. L. Smith. On efficient computation of shortest Dubins paths through three consecutive points. In *IEEE 55th Conference on Decision and Control (CDC)*, pages 6010–6015, 2016.

[18] G. Kizilates and F. Nuriyeva. On the nearest neighbor algorithms for the traveling salesman problem. *Advances in Intelligent Systems and Computing*, 225:111–118, 01 2013.

[19] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989. ISBN 0201157675.

[20] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[21] E. Kang. Dubins curve for matlab. `https://www.github.com/EwingKang/Dubins-Curve-For-MATLAB`, (last accessed on 2020-05-13).