NTNU

Norwegian University of
Science and Technology

Automatic dynamic generation of drill sequences for tunneling

Eivind Jahr Kirkeby

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Jan Tommy Gravdahl, ITK
Co-supervisor: Håkon Håkonsen, Bever Control AS

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Assignment

In tunneling production with the drill and blast method blast holes are drilled by a tunneling drill rig. The drill rig has three booms each equipped with a drill that drills individual holes.

For a tunneling drill rig, it is possible to have automatic beam movements. Automatic beam movement is currently not used in practice because it requires good instinct to manually generate a usable sequence plan. In addition, there is no current functionality that ensures an optimal sequence that deals with disturbance while drilling. The objective of this thesis is to develop optimized strategies for dynamic generation/update of the sequence plan and optimizing automatic transitions. Write software for presentation and analysis of drill logs for the purpose of designing sequence plans. Chose appropriate methods and algorithms, and design automatic sequence plans that must take into account:

- Maximum utilization of booms - allocation on the different (typically 3) booms
- Minimizing time consumed for the transition from one hole to another
- Avoiding collision between beams
- Flexibility for variable drill time and disturbance in the drilling process - dynamic update

Compare the obtained results with real data from the drill logs

Preface

This master thesis is for the degree of M.Sc in engineering cybernetics. The thesis is written for NTNU and Bever Control AS. The background material used for this thesis is drill plans and drill logs in raw data format provided by Bever Control AS. All software and figures used in this thesis are made by the author. All algorithms and methods are self made unless otherwise stated. For a complete list of contributions, see section 1.5.

A special thanks to Jan Tommy Gravdahl at NTNU and Bever Control AS and especially Håkon Håkonsen for guidance throughout the work of this thesis.

Abstract

Tunneling is an important part of building infrastructure in most countries. It is important that tunneling production is done efficiently to reduce the cost of infrastructure projects. In tunneling, multiple holes are drilled for explosives with a tunneling rig. The tunneling rig has multiple booms that drills holes simultaneously. Analysis of drill logs in this thesis provides an understanding of the current efficiency and how it can be improved. The method for drilling a sequence in this thesis discusses an even allocation between booms and dynamically updating this. The optimal path for the allocated holes is calculated by solving the traveling salesman problem using integer programming. The optimal path is constrained so that collisions between booms are avoided. The optimal path is recalculated whenever the allocation changes. The method is tested in a simulator and is estimated to save eight percent of the total drilling time for a sequence from current practice.

Sammendrag

Produksjon av tunneler er en viktig del av bygging av infrastruktur i de fleste land. Det er viktig at tunnelproduksjon gjøres effektivt for å redusere kostnader for infrastrukturprosjekter. I tunnelproduksjon borres det mange hull for eksplosiver med en tunellborerigg. Boreriggen har flere armer som borrer hull samtidig. Analyser av borelogger i denne avhandlingen skaper en forståelse for effektiviteten i dagens metode og hvordan den kan forbedres. Metoden for å borre en sekvens i denne avhandlingen diskuterer en jevn fordeling av hull mellom borarmene og en dynamisk oppdatering av denne. Den optimale stien mellom hullene for hver arm er regnet ut ved å løse "the traveling salesman problem" ved bruk av heltallsprogrammering. Den optimale stien er begrenset slik at kollisjoner mellom armene unngås. Når fordelingen endrer seg regnes den optimale stien ut på nytt. Metoden er testet i en simulator og er estimert til å spare åtte prosent av den totale bortiden for en sekvens sammenlignet med dagens praksis.

Contents

List of Figures

Glossary

blast hole A hole drilled to be filled with explosives

bolt hole A hole drilled for rock bolting

boom A single robotic arm on a drill rig

drill face The rock surface at the end of an incomplete tunnel. This is the surface where drilling occurs

rock bolt Expansion bolts used to secure the rock

Chapter 1

Introduction

Tunneling is an important part of building infrastructure. This thesis presents an optimization of the current drill and blast method. More specific an optimization of the drilling of holes for explosives. Optimizing drilling can be divided into two sub-problems:

- Maximize drilling speed or minimize the drilling time.
- Minimize transition time between two holes.

This thesis considers the last of these problems. Transition time can be divided into three.

- Waiting for the operator to be available
- Moving to next hole
- Changing drill bit

The operator operates three booms drilling the holes. The operator cannot control all three booms simultaneously. As a consequence sometimes the boom has to wait for the operator to be available. This is only a case in manual mode and not in automatic mode. This will be discussed further in this thesis. The boom has to re-position itself for drilling the nest hole. This thesis mainly discusses this task. The main goal is to create an automatic way to determine an optimal drill sequence minimizing the time spent moving the booms. The changing of drill bits is done periodically when they are worn out. This is simply done by a worker screwing off the old bit and screwing on a new bit.

1.1 Motivation

Efficiency in tunnel construction is important to create infrastructure projects faster and more cost efficient. An automatic dynamic generation of drill sequences for tunneling as discussed in this thesis is both a step towards making a more efficient sequence and enabling the use of automatic drilling. At a tunnel construction site, a small cut in time can be a significant cut in expenses, both in the cost of labor, the cost of equipment and other expenses.

1.2 Literature review

In Eide [2009] a topic about automatic drilling is discussed. Automatic drilling is the use of the drill rig without manual control of the drill booms from the operator. The booms move automatically to the next hole. Automatic drilling reduces time spent on moving the booms and results in a more precise drilling. Edvardsen also claims in Edvartsen [n.d.] that automatic drilling is more efficient than manual drilling with the same arguments as Eide. The article also discusses why automatic drilling is so little used despite the documented advantages. The operator often feels passive when only monitoring a process and not controlling the process. In the mind of the operator, the automatic drilling can take more time because he feels passive and not active. A translated part from Helleseth [2000] described automatic drilling this way:

> When the author arrived at the construction site automatic drilling was little used, but as the sequence plans got better automatic drilling was more used. Automatic drilling is though little used for drilling the contour holes and never for the floor holes. The reason automatic drilling was not used more was that the conditions for drilling automatically was not always good. Automatic drilling was inconvenient in relation to manual drilling. With inconvenient, the operators meant that if you had to cancel the drilling or change the sequence plan it was too much pushing of buttons and therefore easier to do manually.

There is clearly a need for an automatic way to generate a sequence and dynamically update it to efficiently use automatic drilling.

A literature search for construction of sequence plans for tunneling shows that little to no

work has been done to automatically generate an optimal sequence for tunneling rigs.

1.3 Assumptions

The changing of drill bits is not logged in any way in the drill logs. As a consequence, there is no way to include the changing of the drill bit in any of the analysis. This thesis assumes there are no changing of drill bits. A change of drill bits does not affect the result in any considerable way.

1.4 Background

1.4.1 Drill and blast tunneling production method

The drill and blast tunneling production method consists of the controlled use of explosives to excavate a tunnel. It is a process involving several steps that are repeated throughout the tunnel. These steps are shown in figure 1.1. The first step, drilling holes for explosives, is to the upper left. Advanced drill rigs is currently doing this. The goal is to drill a sequence of holes on the drill face or surface in front of the rig. These holes are defined in a drill plan construed with a pattern that ensures a good distribution of explosives.

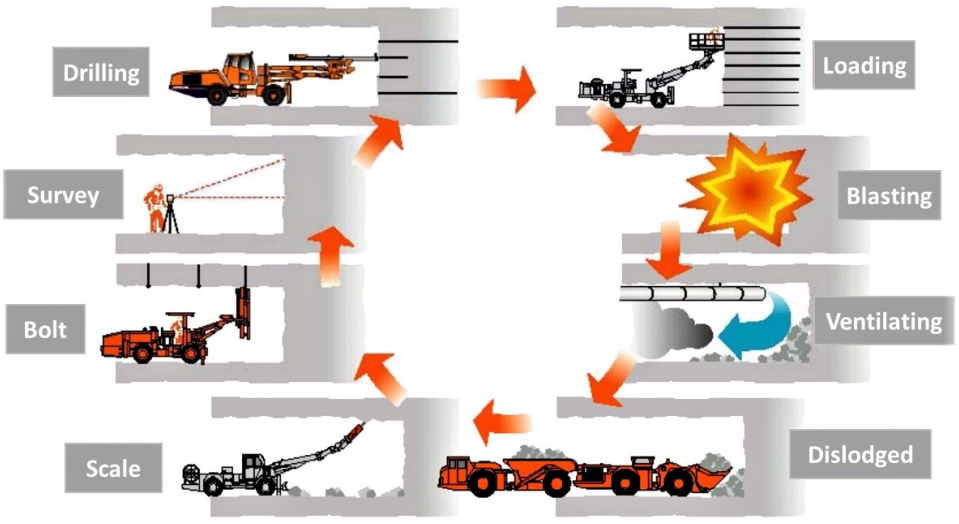


Figure 1.1: The drill and blast tunneling cycle

The second and third step is to load the drilled holes with explosives and detonating the blast. Blasting creates dust and blast fumes that are ventilated followed by removal of the blasted rock. The blasting leaves loos rock on the new surfaces that are scaled off. This is followed by bolting for security purposes. The newly created area is then 3D scanned to document the newly blasted surfaces.

In addition to the steps in the figure, shotcrete is applied to the tunnel roof in between some of the blastings. Water coming into the tunnel is usually solved by grouting. Grouting is the pressurized injection of concrete into cracks.

The main competing technology for drill and blast is TBMs, short for Tunnel Boring Machines. TBMs works without the use of explosives. They consist of a rotating circular head that digs through the rock with metal teeth.

1.4.2 The drilling rig

The drilling rig is the one performing the first task in the drill and blast cycle. Its purpose is to drill holes that are to be loaded with explosives or holes for bolts. This section aims to give an understanding of how the drilling rig works. An image of the rig is presented in figure 1.2.

The drilling rig has four robotic arms, or booms, whereof three are fitted with drilling machines, and one is fitted with a basket for personnel use. The three drilling arms have nine degrees of freedom each. Of the nine joints, seven are revolute and two are prismatic. All of the joints are hydraulically powered and equipped with sensors.

The booms and drilling machines are controlled from the cabin showed in figure 1.3. The picture also shows the placement of the booms, the three drill booms are placed side by side. This naturally constrains them to a working area to the left, in the center and to the right.

Figure 1.2: AMV tunneling rig. Courtesy of AMV

Figure 1.3: View from rig cabin. Drill booms marked with names. Courtesy of AMV

1.5 Contributions

For this thesis Bever Control AS has provided drill logs and drill plans. The drill logs are in raw text format and the drill plans are in XML format. All plotting and analyzing of the logs have been contributed by this thesis. None of the software or plots was provided as background. All methods and algorithms have been made and selected by the author unless otherwise stated. Following is a complete list of the work done in this thesis.

- Collection of data from drill logs and drill plans
- Plotting of drill plans
- Plots for visualizing the drilled sequences
- Qualitative and quantitative analysis of drilled sequences from Kjørholttunnelene
- An algorithm to allocate holes to the different booms
- An algorithm for calculating an optimal path for each boom with respect to collision avoidance
- A dynamical update of the allocation and of the optimal solution for each boom
- A simulator to simulate the results of the algorithm
- A video of the simulation
- A field trip to Kjørholttunnelene to observe a sequence

1.6 Outline

This thesis first presents the theory behind the traveling salesman problem and collision avoidance for robotic manipulators in the same workspace in chapter 2. Then analysis on how drilling is done today by considering the drill logs are done in chapter 4. Chapter 5 discusses an algorithm and an implementation on how to automatically allocate drill holes to different booms and optimize each boom sequence. In addition chapter 5 includes an algorithm for avoiding collisions based on the theory presented about collision avoidance. A way to dynamically update the optimal solution is also discussed. This solution considers both variable drill time and collision avoidance. Lastly the results of the method is discussed using results from simulations.

Chapter 2

Theory

2.1 Traveling Salesman Problem

The Traveling Salesman problem, or TSP, is a well-used problem statement. The problem is to find the shortest route that enables the salesman to visit all cities in a map, and each city is only visited once. There are roads between cities with a specific length. The traveling salesman problem can be represented as a graph where each city is a node and all paths between cities are edges.

The problem can be formulated as an integer programming problem. An extended standard way to write an integer programming problem is presented in equation 2.1 where equality and inequality constraints both are represented. Nemhauser and Wolsey [1988]

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} && \text{(2.1a)} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} && \text{(2.1b)} \\ & \mathbf{A}_{\mathbf{eq}}\mathbf{x} = \mathbf{b}_{\mathbf{eq}} && \text{(2.1c)} \\ & \mathbf{x} \geq 0 && \text{(2.1d)} \\ & \mathbf{x} \in \mathbb{Z} && \text{(2.1e)} \end{aligned}$$

The goal is to minimize the objective function 2.1a subject to the constraints 2.1b-2.1e. $\mathbf{x}$

is a vector of variables. $\mathbf{A_{eq}}$ and $\mathbf{A}$ are known matrices of coefficients, $\mathbf{b_{eq}}$, $\mathbf{b}$ and $\mathbf{c}^T$ are known vectors of coefficients. It is the selection of $\mathbf{x}$ that minimizes the objective function subject to the constraints that is the goal. The first constraint is the inequality constraint, the second constraint is the equality constraint and the third constraint is limiting $\mathbf{x}$ to positive numbers. These three constraints are normal for a linear programming problem. What differs integer linear programming from linear programming is the last constraint, limiting $\mathbf{x}$ to integers.

Rewriting this some can give a formulation for the traveling salesman problem. The first step is the selection of $\mathbf{x}$, the vector of variables. In the traveling salesman problem decisions have to be made about which roads to use. This can be done by defining a decision variable x_{ij} as defined as in equation 2.2. This variable decides if there is a used road from city i to city j and can only have the value zero or one. Dantzig [2016]

$$x_{ij} = \begin{cases} 0, & \text{if the road goes from city } i \text{ to } j \\ 1, & \text{otherwise} \end{cases} \tag{2.2}$$

The coefficients in c_{ij} would then be the length of the road between city i and city j. With x_{ij} and c_{ij} defined, the objective function is the sum of the length of all used roads. This is presented in equation 2.3a where the total numbers of roads are n.

$$\min \quad \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \tag{2.3a}$$

$$\text{s.t.} \quad 0 \leq x_{ij} \leq 1 \qquad i, j = 1, \ldots, n \tag{2.3b}$$

$$u_i \in \mathbb{Z} \qquad i = 1, \ldots, n \tag{2.3c}$$

$$\sum_{i=1}^{n} \sum_{j=i}^{n} x_{ij} = n - 1 \tag{2.3d}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{2.3e}$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{2.3f}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \qquad \forall S \subset N \tag{2.3g}$$

To properly constrain x_{ij} to accept only zero and one, constraint 2.3b and 2.3c is introduced. Constraint 2.3b constrains x_{ij} to be minimum zero and maximum one. Constraint 2.3c constrains x_{ij} to only accept integer values. Together they constrain x_{ij} to be either zero or one. Constraint 2.3d defines the number of total roads to be used to $n - 1$. The number of roads to use cannot be more or less than the number of cities minus one.

To make sure that the path is continuous, equality constraints 2.3e and 2.3f is introduced. They specify that each city needs to be arrived at from exactly one other city and that there is a departure to exactly one other city.

Constraint 2.3f makes sure that there is only a single big tour connecting the cities and not multiple sub-tours. The constraint states that for any subset of cities S, the tour must enter and exit that set. In other words, there cannot be a closed tour in any subset of cities S.

2.2 Collision Avoidance

Collision avoidance for manipulators with overlapping workspaces is a well-discussed topic in robotics. There are several ways to solve the problem. For this thesis a collision detector with a fast an simple calculation is preferable. A detector based on simple geometric shapes will provide the required results. Creating cylindrical bodies around each link of the manipulator and calculating overlapping is commonly used to detect collision. The methodology in Ennen et al. [2014] is what this thesis is using. There are other methodologies using spherical shells like Bosscher and Hedman [2009], among other methods for detecting and avoiding collisions. The simplest of all is dividing the workspace between the manipulators. This is however often not possible as manipulators have to perform tasks in the same workspace and that usually results in inefficient systems.

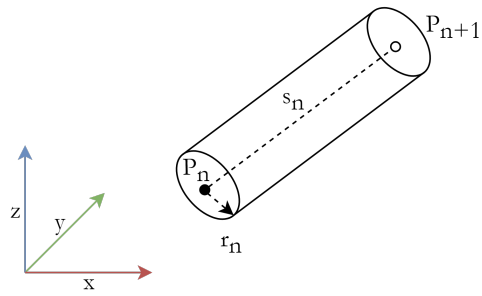


Figure 2.1: Manipulator link model. Inspired by Ennen et al. [2014]

For each robotic component, otherwise known as link, there is a line $\vec{\mathbf{s}}_n$ spanning through it from joint P_n to joint P_{n+1}. The parameter μ_n is the position on the line. This line is showed in figure 2.1 and its equation is showed in 2.4. P_n is the current joint origin and P_{n+1} is the next joint origin. $\vec{\mathbf{s}}_n$ is the line between these points, and μ_n is the line parameter. r_n is the is the radius of the cylinder. $|r_n|$ is selected to be the distance from the line $\vec{\mathbf{s}}_n$ to the point furthest away from the line. The line and the distance together define a cylinder that completely covers the component.

$$\vec{\mathbf{s}}_n(\mu) = \mathbf{P_n} + \mu_n(\mathbf{P_{n+1}} - \mathbf{P_n}) \tag{2.4}$$

Collision is happening at timestamp τ between two robot lines $\vec{\mathbf{s}}_n$ and $\vec{\mathbf{s}}_m$ if their minimal distance is less than the sum of their related radiuses r_n and r_m. Defining the minimal distance between $\vec{\mathbf{s}}_n$ and $\vec{\mathbf{s}}_m$ at timestamp τ as $d_{nm}(\tau)$ provides equation 2.5.

$$d_{nm}(\tau) < r_n + r_m \tag{2.5}$$

Without considering velocities the robots will collide before the collision avoidance stops it. By expanding equation 2.5 to equation 2.6 velocities are included. Also a safety reserve S is included.

$$d_{nm}(\tau) - 2t_r\|\vec{\mathbf{v}}_n(\tau-1,\tau) - \vec{\mathbf{v}}_m(\tau-1,\tau)\|_2 - S < r_n + r_m \tag{2.6}$$

d_{nm} is the minimum distance between the two finite lines. Depending on how two finite lines are configured in relation to one another the calculation of their minimum distance

d_{nm} is different. The three different cases are shown in figure 2.2 and the resulting equations are shown in equation set 2.7. In case of A, the minimum distance is the minimum distance between the lines $\vec{\mathbf{s}}_m$ and $\vec{\mathbf{s}}_n$, in case B the minimum distance is between a point P_m and a line $\vec{\mathbf{s}}_n$ and in case C the minimum distance is the distance between two points P_m and P_n.

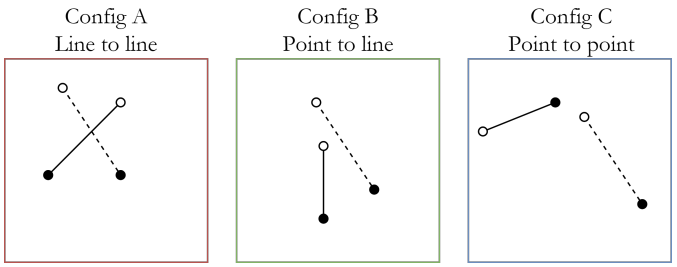


Figure 2.2: Three configurations for shortest distance between finite lines. Inspired by Ennen et al. [2014]

$$d_{nm,A}(\vec{\mathbf{s}}_m, \vec{\mathbf{s}}_n) = \frac{\left\| \det \begin{bmatrix} \mathbf{P_n} - \mathbf{P_m} & \mathbf{P_{n+1}} - \mathbf{P_n} & \mathbf{P_{m+1}} - \mathbf{P_m} \end{bmatrix} \right\|_2}{\|(\mathbf{P_{n+1}} - \mathbf{P_n}) \times (\mathbf{P_{m+1}} - \mathbf{P_m})\|_2} \tag{2.7a}$$

$$d_{nm,B}(\mathbf{P_m}, \vec{\mathbf{s}}_n) = \frac{\|(\mathbf{P_n} - \mathbf{P_m}) \times (\mathbf{P_{n+1}} - \mathbf{P_n})\|_2}{\|(\mathbf{P_{n+1}} - \mathbf{P_n})\|_2} \tag{2.7b}$$

$$d_{nm,C}(\mathbf{P_n}, \mathbf{P_m}) = \|\mathbf{P_n} - \mathbf{P_m}\|_2 \tag{2.7c}$$

While these are the three different equations there are in total 9 different cases of configuration. One for configuration A, four for configuration B and four for configuration C.

μ_n and μ_m describes which configuration to consider. μ_n and μ_m is the point of applied minimum distance on the lines. With a line parameter $\mu \in [0, 1]$ the line is considered, with $\mu < 0$ point $\mathbf{P_m}$ or $\mathbf{P_n}$ is considered. If $\mu > 1$ the point $\mathbf{P_{m+1}}$ or $\mathbf{P_{n+1}}$ is considered.

To determine the value of μ_n and μ_m a temporary plan can be constructed. In the case of n the temporary plane is defined with the plane i equation 2.8. This provides a plane that includes line $\vec{\mathbf{s}}_m$ and that is orthogonal to $\vec{\mathbf{s}}_m$ and $\vec{\mathbf{s}}_n$.

$$\vec{\mathbf{q}} = \mathbf{P_m} + \alpha\,(\mathbf{P_{m+1}} - \mathbf{P}) + \beta(\mathbf{P_{m+1}} - \mathbf{P_m}) \times (\mathbf{P_{n+1}} - \mathbf{P_n}) \tag{2.8}$$

Equation 2.9 describes the intersection of the temporary plane and the line $\vec{\mathbf{s}}_n$. Completing the calculation results in μ_n. The same can be done for μ_m by changing all m with n and vice versa. Determination of the configuration can be done now that both line parameters are calculated. The decision tree for determining the configuration is shown in figure 2.3.

$$\begin{bmatrix} \alpha \\ \beta \\ \mu_n \end{bmatrix} = \begin{bmatrix} \mathbf{P_{m+1}} - \mathbf{P_m} & (\mathbf{P_{m+1}} - \mathbf{P_m}) \times (\mathbf{P_{n+1}} - \mathbf{P_n}) & \mathbf{P_n} - \mathbf{P_{n+1}} \end{bmatrix}^{-1} [\mathbf{P_n} - \mathbf{P_m}] \quad (2.9)$$

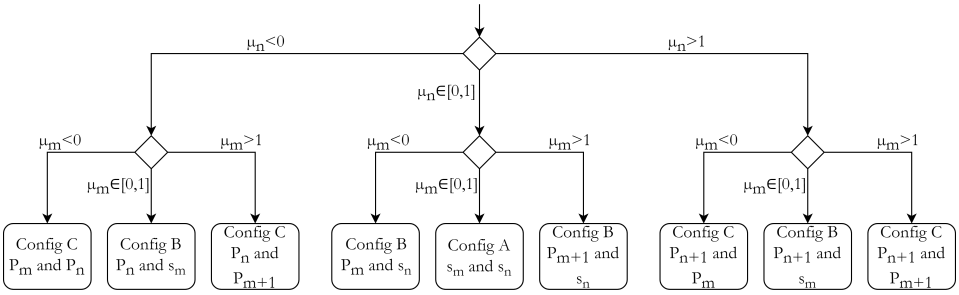


Figure 2.3: Decision tree used to determine configuration and which points or lines to consider. Inspired by Ennen et al. [2014]

Using the decision tree in figure 2.3 will result in which equation to use and which lines or points to use in the calculation.

The calculation of equation 2.6 is then complete. Collision needs to be determined for each combination of robotic components.

Chapter 3

The drill plan

To start drilling a drill plan needs to be provided. The drill plan is either automatically generated in a computer program or manually constructed. A drill plan contains all information needed to start manual drilling. It contains information about each hole as well as other information used to identify the plan. Each hole has information about start coordinates, end coordinates, hole type and more.

The drill plan is represented as an XML-file with all the relevant information. The plan contains both the start and the end coordinates of each hole. All of the start coordinates are in the same plane. The 3D start coordinates are projected onto the somewhat uneven drill face when drilling. The start coordinates for a typical drill plan is plotted in figure 3.1. This drill plan is for a 2-lane freeway tunnel and will be used later for showing how to optimize the sequence.

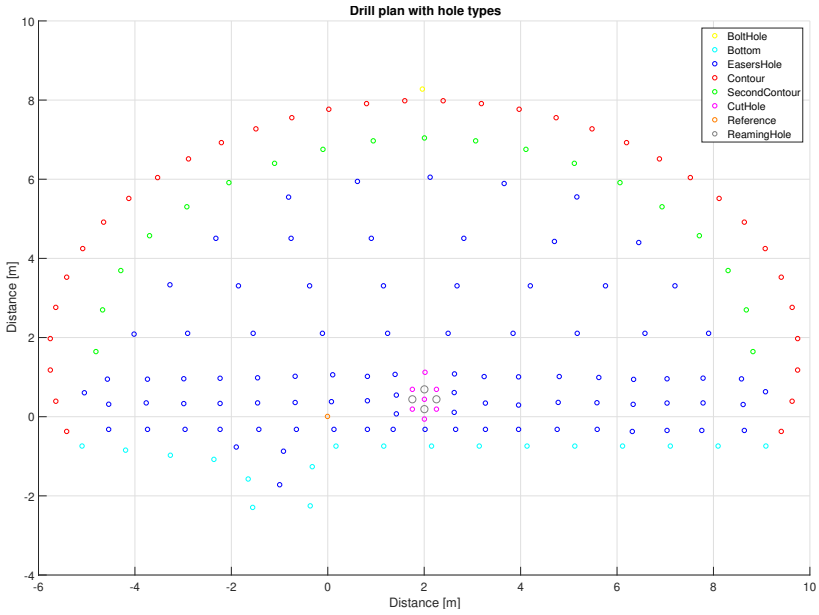


Figure 3.1: Drill plan from the Kjørholt-tunnel. Used from pel 13960 to 14410. Start coordinates plotted in 2D

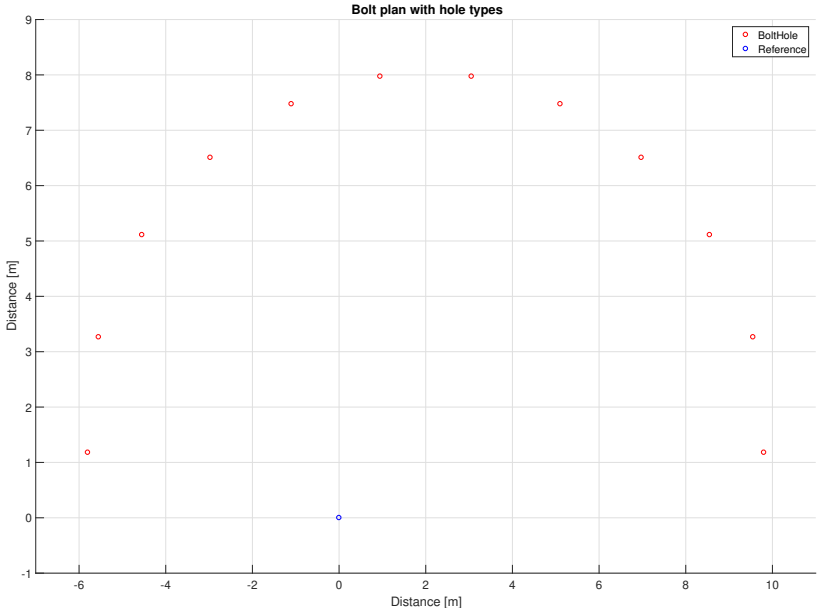


Figure 3.2: Bolt plan from the Kjørholt-tunnel. Used from pel 13960 to 14410. Start coordinates plotted in 2D

To understand the drill plan there is essential to have some basic understanding of blasting with explosives. When blasting, the rock has to move somewhere, if the rock doesn't have somewhere to move not much will happen. As a consequence the blasting happens sequentially with small time delays. The first part that is blasted is the cut. The cut is the center area consisting of the cut holes and the reaming holes. This part blasts the rock out of the hole away from the drill face. It creates an air pocket for the remaining parts to blast inwards against. The last part detonated is the contour.

The roof of the tunnel is drilled to provide holes for rock bolting. Rock bolting is done to provide stability to the excavations. The bolts used are expansion bolts, and are usually 5 meters long, but can differ in length depending on the stability of the rock. The most common bolting practice today is bolting in a two by a two-meter grid in the entire roof of the tunnel. For the holes drilled for bolts in the roof, a separate plan is provided as an additional xml-file. A typical plan corresponding to the contour size of figure 3.1 is displayed in figure 3.2. The bolt plan contain start coordinates and end coordinates for a cross section of the tunnel. Only the start coordinates is displayed in figure 3.2.

The bolt plan is repeated with a fixed interval. To provide a two by two meter grid with this plan it is repeated for every two meters. As seen later in this thesis multiple bolt plans are drilled for each sequence. With the normal drill depth around five meters the plan is often repeated two or three times. A composition of both the drill plan and bolt plan is displayed in figure 3.3.

It is only the start coordinates of the holes that are relevant for this thesis. No matter what the end coordinates are the boom have to move to the start coordinates to start drilling.

In figure 3.4 a screenshot of the operator screen is displayed. The operator interface display both the drill plan and the bolting plan in the same image, as well as the direction of each hole, and the position and direction of each boom.

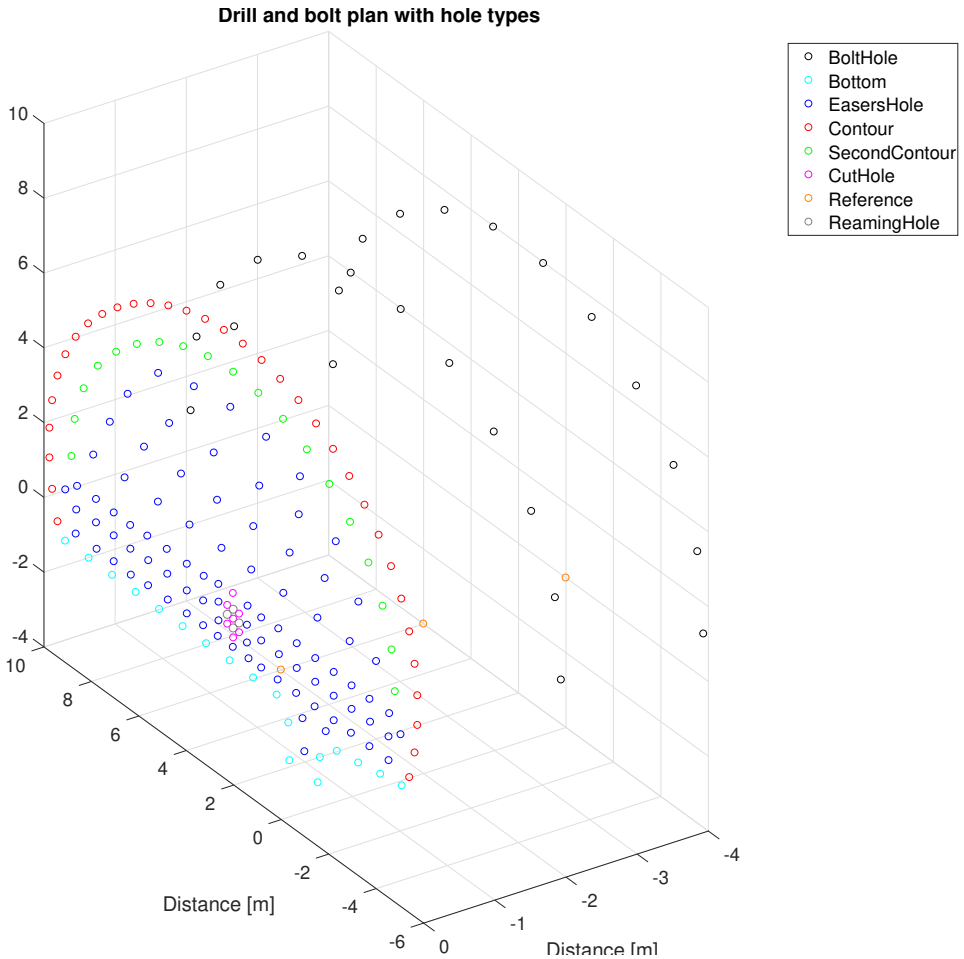


Figure 3.3: Drill and bolt plan from the Kjørholt-tunnel. Used from pel 13960 to 14410. Start coordinates plotted in 3D

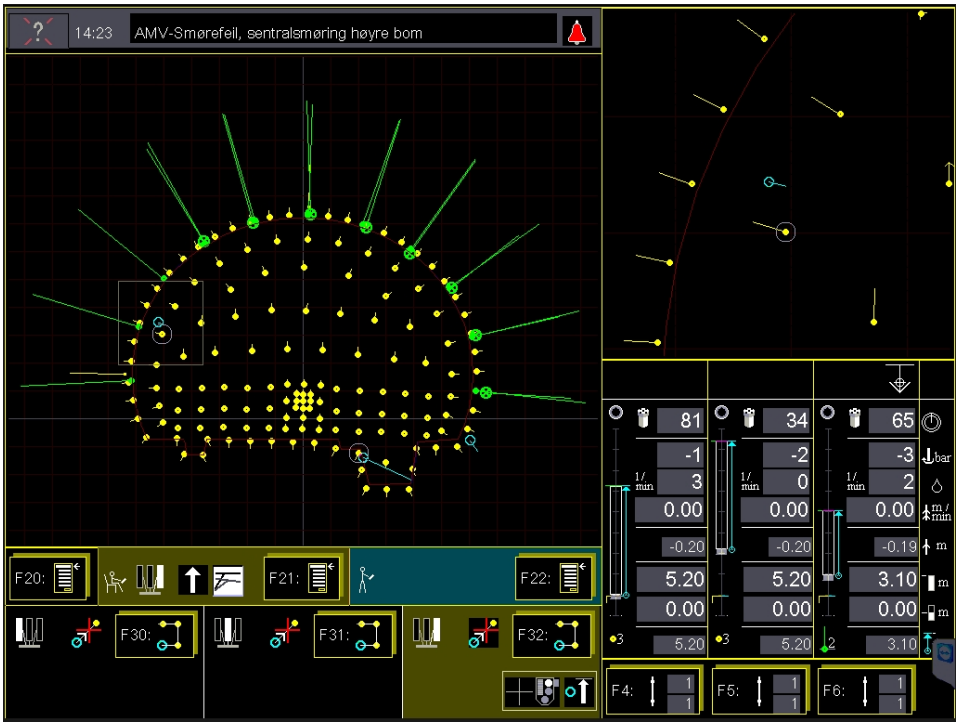


Figure 3.4: Screenshot of operator screen showing the drill plan as well as position of booms. Courtesy of Bever Control

Chapter 4

Analysis of drill logs

An important part of providing a good solution for a problem is understanding the way the present method works. The drill logs contain this information. The logs contain information about every drill hole drilled by Bever Control systems. They contain information about hole placement, hole type, boom used and more. The logs are saved as raw text files. To analyze the logs, code for reading the text files has been constructed for this thesis.

For this thesis, all the drill logs analyzed are from Kjørholttunnelene. Kjørholttunnelene is located in Telemark, Norway and is an ongoing highway tunneling project as this thesis is being written. The logs provided by Bever Control are from 2017 and 2018. Having fresh logs will give a clear picture of the current method when this thesis is being written.

Information from the logs will also provide a baseline for improvement. Parameters and calculations from the logs can be used to compare the improvements this thesis suggests.

4.1 Sequence

The first goal of the analysis is to visualize all the drilled holes in a sequence. A sequence is all the holes drilled from a drill plan. A typical sequence drilled manually by a operator is showed in figure 4.1. The holes drilled by different booms are colored in different colors. For each boom there is a sequence of the drilled holes. The sequence is presented by a numbering of holes for each boom. Following the arrows provides the next hole. The arrows presents the path of the boom. The reason for the x-axis being flipped is because

the plot is considered from the position of the rig, and that does not always provide a x-axis positive to the right. The surface of the drill face is not perfectly flat, so the holes shown in this figure are projected onto a perfect plane at the drill face. All the holes in this figure are drilled away from the viewpoint provided in the figure.

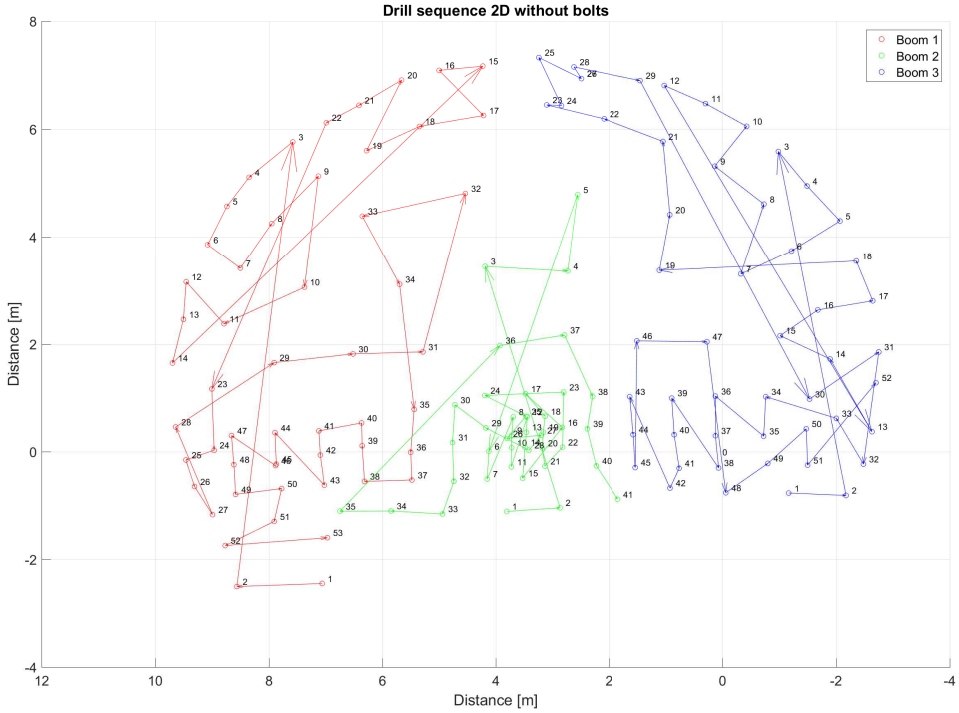


Figure 4.1: Drill sequence projected onto a 2D plane on the drill face without bolt holes

It is clear that the operator is trying to put some strategy to use for the drilling order. The lower holes are drilled in a zigzag pattern. On the outer contour, the operator often follows the contour, but not consistently. It is reasonable to imagine a potential for some sequence optimization based on this figure. It is also clear from the plot that each boom has worked its own area. This will be discussed later in this chapter.

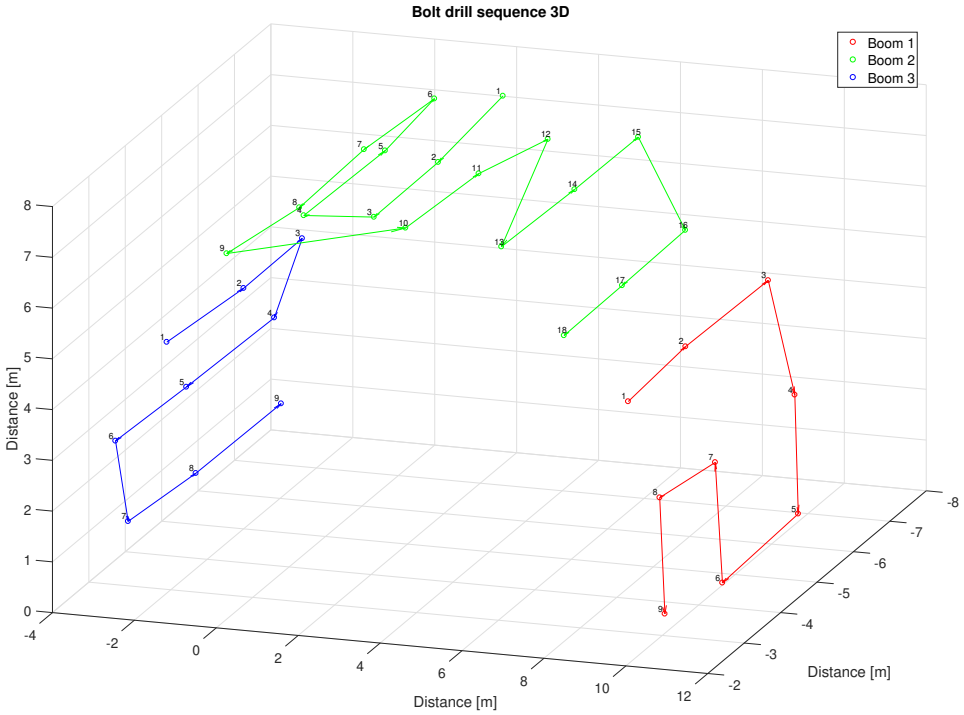


Figure 4.2: Bolt hole drilling sequence

The holes drilled from the bolting plan is shown in figure 4.2. The same applies to this plot, the sequence is illustrated by arrows from hole to hole. The two by two-meter bolting pattern used here is evident. It is clear that the center boom drills most of the bolting holes in this sequence.

The holes in figure 4.1 and figure 4.2 are drilled as the same sequence. While the machine is in position it drills both holes for blasting and holes for bolting. The total sequence is shown in figure 4.3 as a 3D plot. The sequence in this plot is quite hard to observe. For that reason a figure with both blast holes and bolt holes projected onto a 2D plane at the drill face is displayed in figure 4.4.

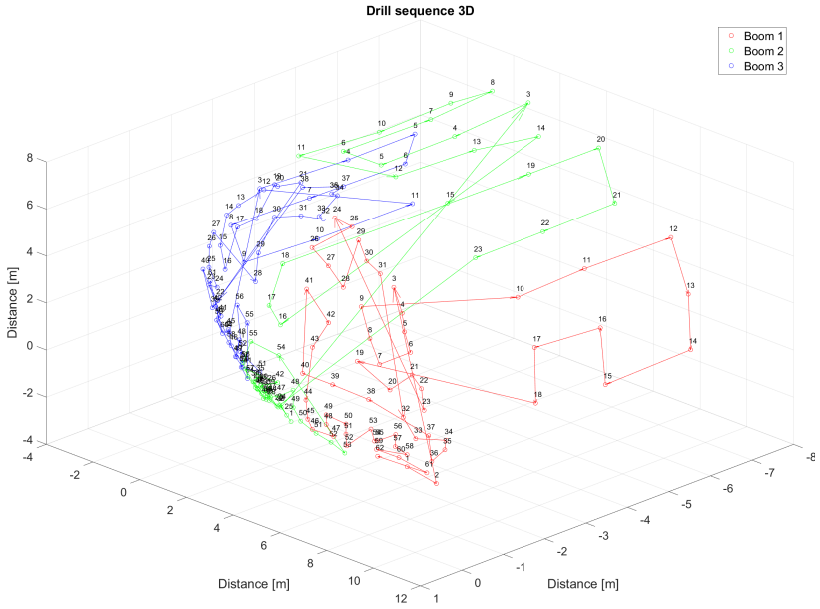


Figure 4.3: Drill sequence in 3D

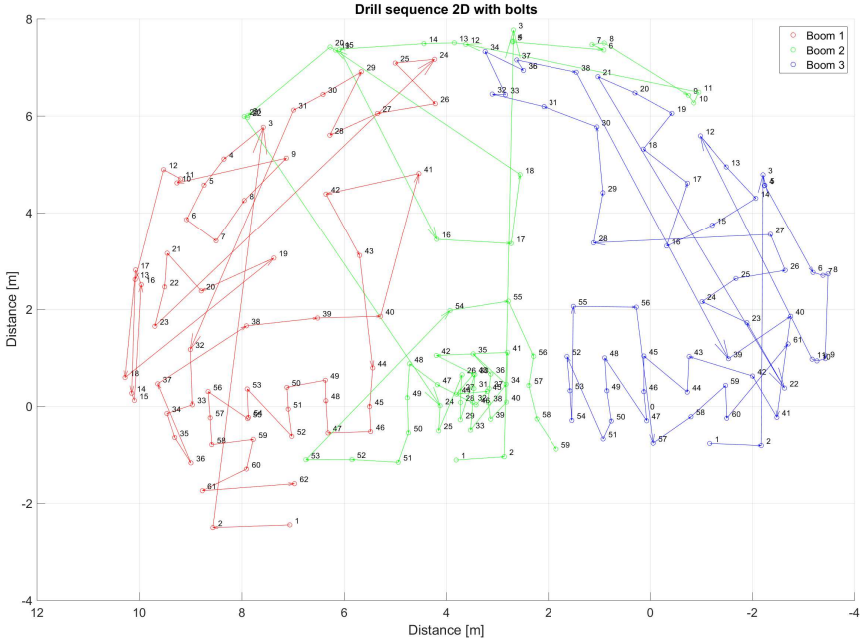


Figure 4.4: Drill sequence projected onto a 2D plane on the drill face

To get an understanding of the hole direction and the hole depth of the holes a figure 4.5 is plotted. The figure shows the start point of each hole and an arrow indicating the direction and depth of the hole. The bolt holes form a crown in the tunnel roof for rock bolting. The rest of the holes point in the direction of the tunnel shaft. It is these holes that define the area to be carved out from the blasting. All of the holes in this figure is about five meters deep.

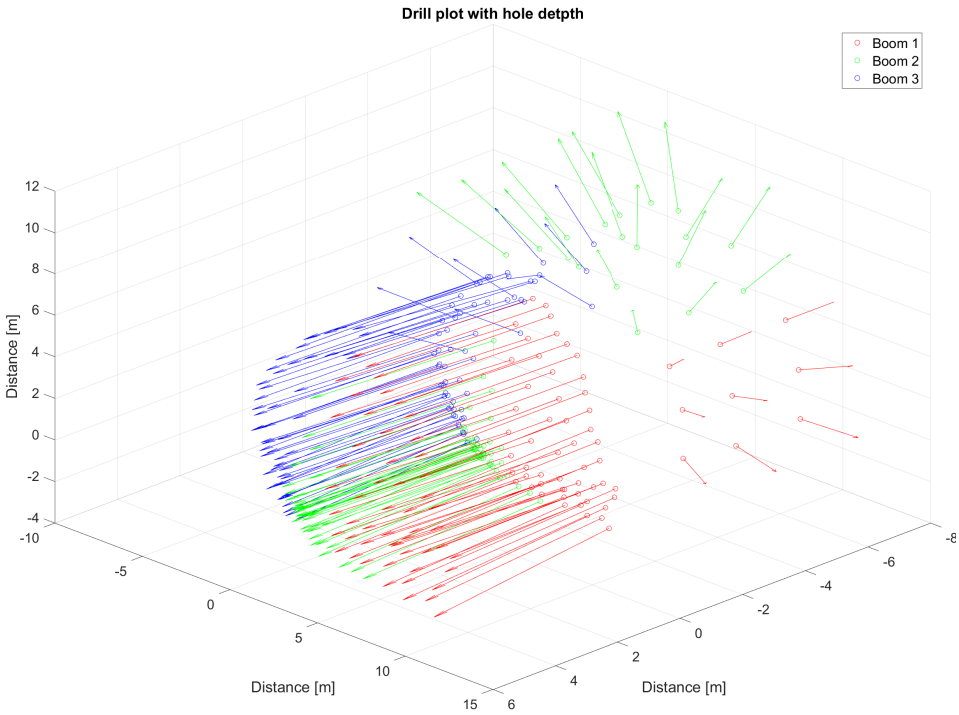


Figure 4.5: Plot of a drilled sequence with hole depth and direction represented as arrows. The origin of the plot is located at the coordinates of the drill face origin

.

4.2 Boom work area

By considering a single sequence, it is hard to get any repeating patterns, but consideration of multiple sequences at once will give a clearer view of how the drilling is usually done. In this section, the goal is to present the area in which each boom usually works.

In figure 4.6 all drill logs in Kjørholttunnelen corresponding to the drill plan in figure 3.1

is presented. There are a total of 68 drill logs plotted on top of each other. The plot is only showing the blast holes and not the bolt holes. This gives an understanding of the work area of the booms. The position of each hole in the drill plan is seen as a cluster of, one would assume, 68 points each.

Trying to define the work area with simple geometrical properties provides the lines in the figure. Boom 2 works within a circle with origin at $y = 0$ at in the center of the points along the x-axis. Boom 1 and 3 works on each side of the line drawn orthogonal to the road surface. This geometrical division of points is used to automatically allocate points between each boom later in the thesis in section 5.1.

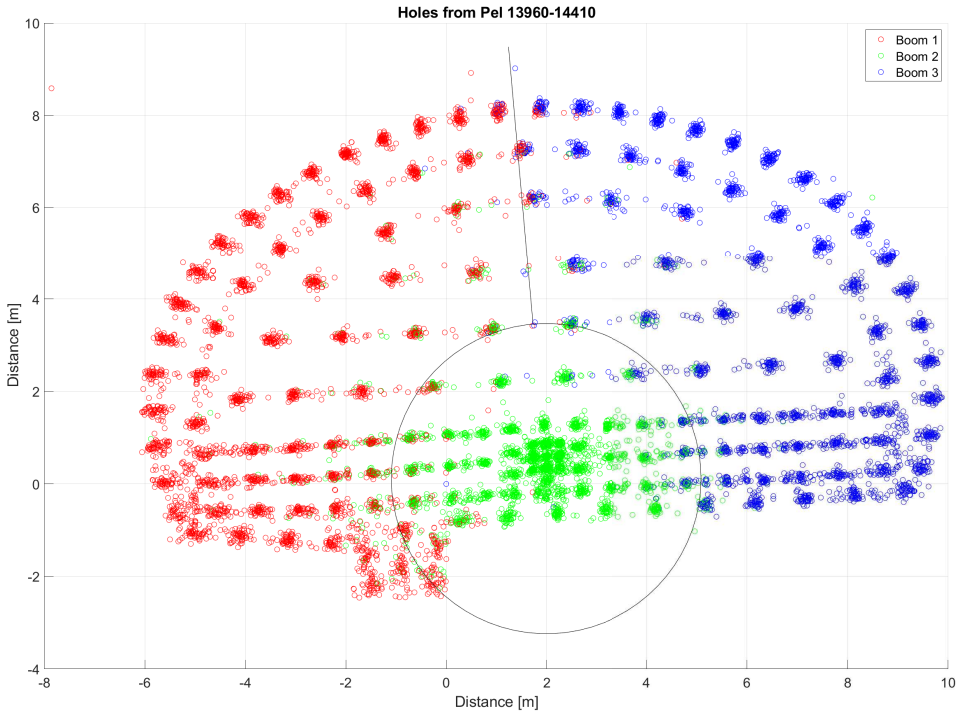


Figure 4.6: All drill logs from pel 13960-14410 in the same plot, with area lines

Bolt holes from the same 68 drill logs are shown in figure 4.7. It is clear that boom 1 and boom 3 always keeps to each side of the roof. Boom 2, however, has holes scattered around the entire roof. This means that boom 2 can drill any hole in the roof, and if desirable all holes in a sequence.

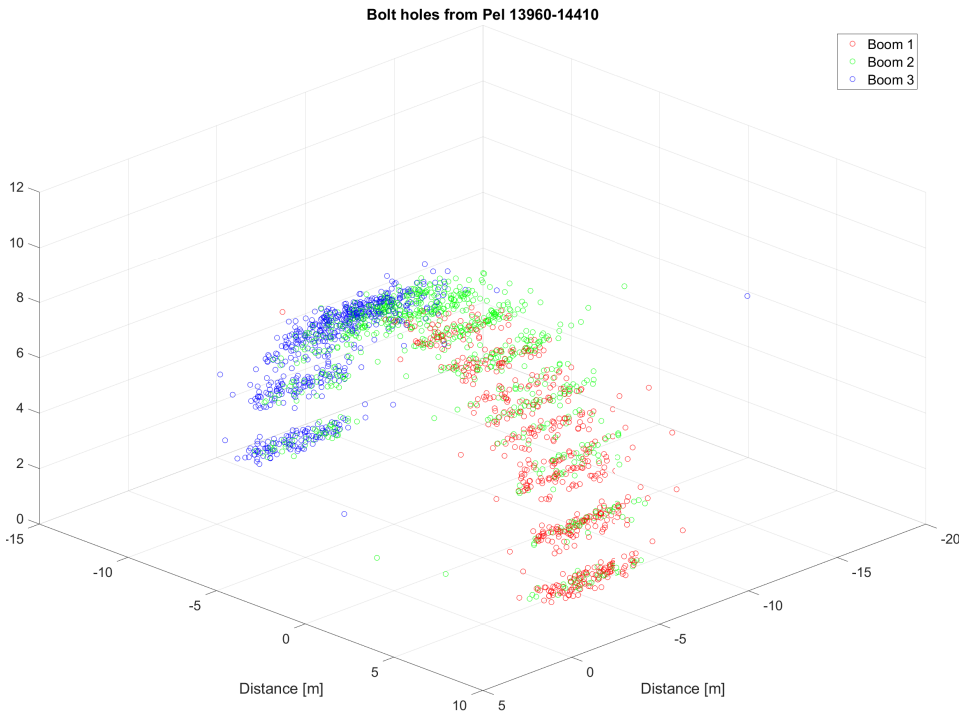


Figure 4.7: All drill logs from pel 13960-14410 in the same plot, with area lines

4.3 Time usage

It is important to get an understanding of how the time is used when working a sequence. The time is the desired parameter to minimize. In figure 4.8 a Gantt chart of a typical sequence is displayed.

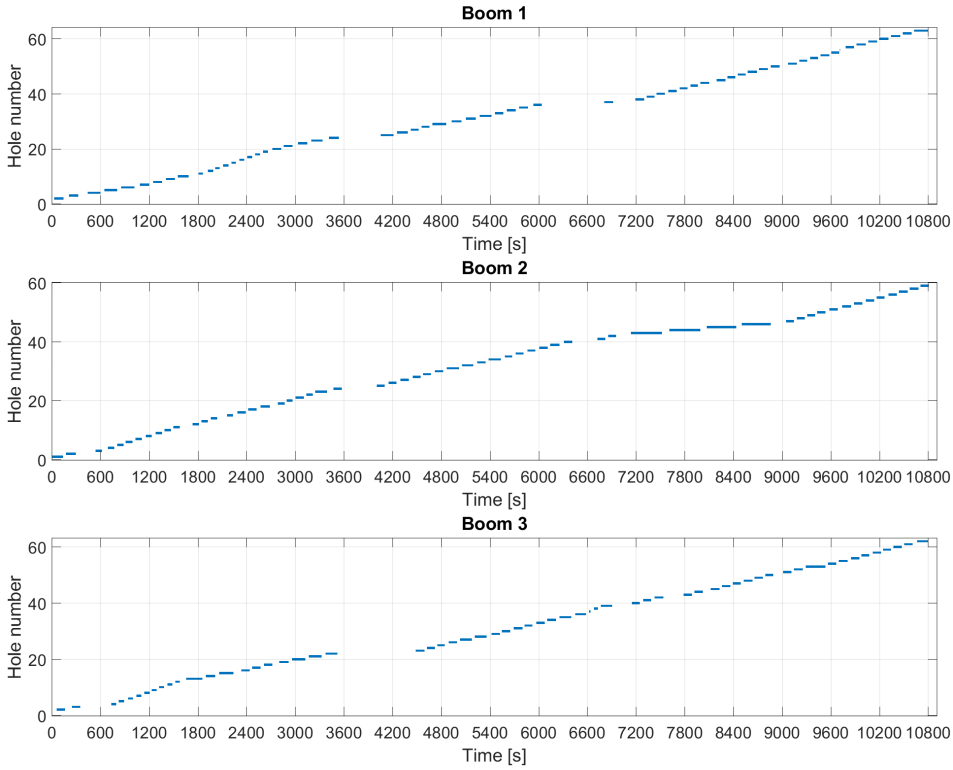


Figure 4.8: Gantt

Each blue horizontal bar represents the duration of the drilling of a hole with a start time and an end time. Between all the holes there is some time where each boom is not drilling. This time is the time spent on moving the boom and starting drilling. Sometimes there are complete stops in drilling. It is reasonable that this downtime has nothing to do with the effectiveness of the sequence, but is due to some other reasons. It may just be that there needs to be personnel in front of the machine. If there are personnel in front of the machine, the booms have to stop drilling. The figure 4.8 is just one drill sequence, to get a better understanding of the total picture, values for multiple sequences have to be considered.

From the drill logs, the time spent drilling for each hole can be calculated. Presenting the time spent drilling for each hole in a histogram results in figure 4.9.

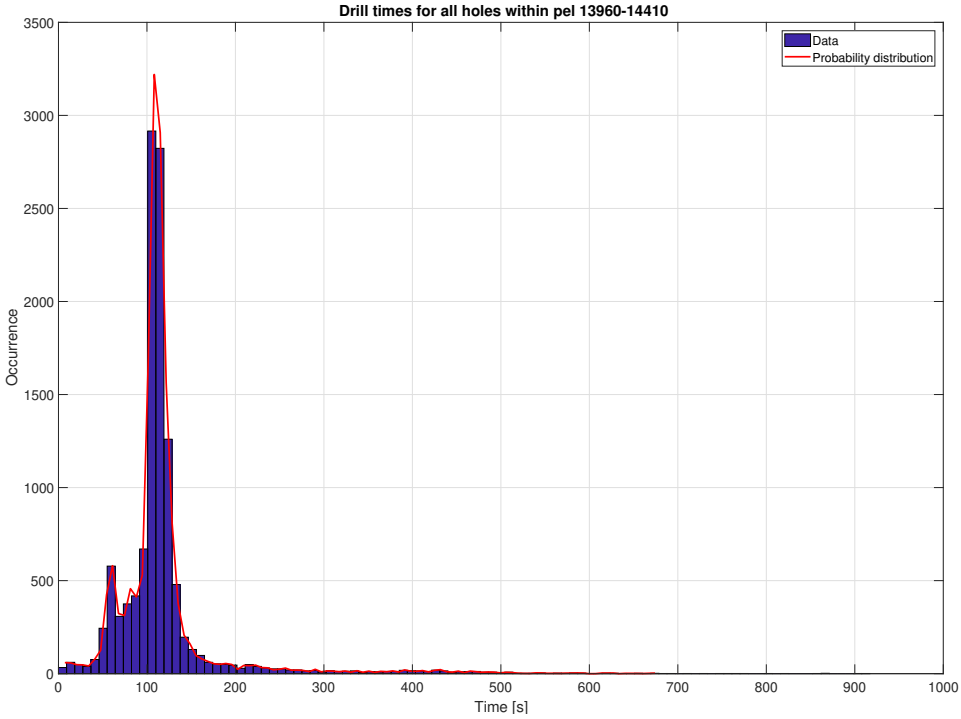


Figure 4.9: Histogram of drill time for each hole.

The histogram shows that drill times for each hole can vary significantly. The reason for these variations is rock quality and anti-jamming. The rock quality can vary from hard to soft rock and the rock could have some cracks speeding up or delaying drilling. To prevent jamming in the event of bent drill rod, an anti-jamming process occurs. This stops the drilling, withdraws the drill rod some and then continues. This will delay drilling. The average drill time in the probability distribution in figure 4.9 is 118.3 seconds. The variance is 4421.5, and that is significant.

4.4 Measurement of drill efficiency

While doing a drill sequence there is two main task performed. One is drilling, and the other is the transition of the boom to the next hole. In this thesis, the efficiency of the drilling is not the main task. The main task is to minimize the time spent on transitions to a new hole. The start time and end time of each hole are logged, so it is possible to calculate the time spent doing transitions. Doing this for a big set of sequences provides a

distribution of transition times. One for each boom, and one total which is the average or the three booms. This is showed in figure 4.10.

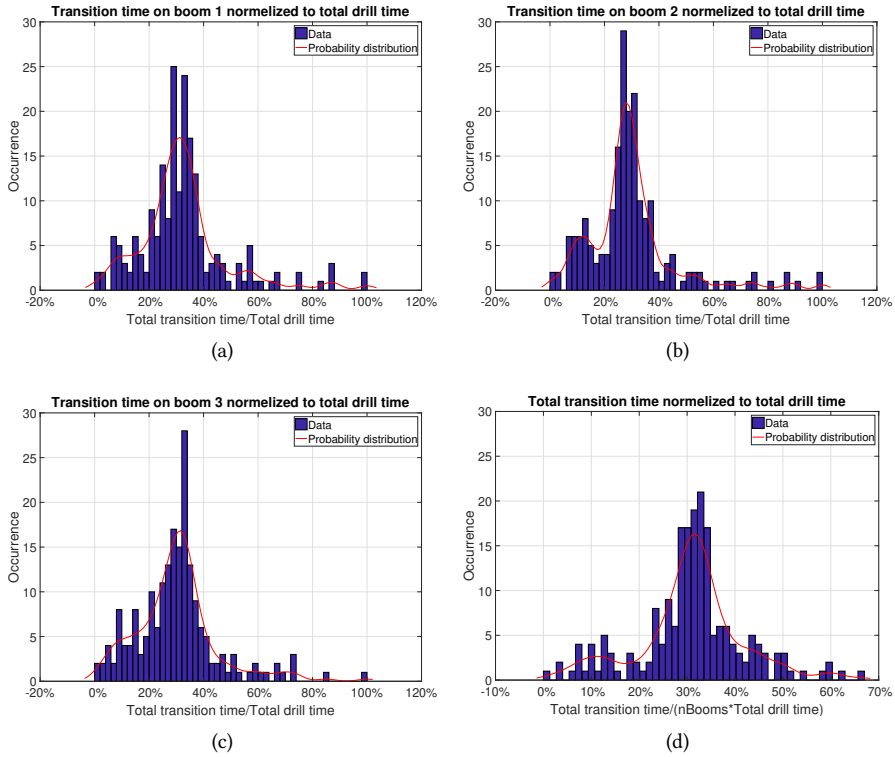


Figure 4.10: (a)-(c): Histogram of time spent on transitions normalized to total drilling time for boom 1-3 (d): Histogram of sum of time spent on transitions for each sequence normalized to total drilling time.

Transition time is defined by when the current boom is not drilling and at least one of the others are drilling. If no booms are drilling, the time is not added to the transition time. This is because pauses unrelated to the sequence should not be added towards a direct measurement of the efficiency of the sequence. It does not mean that pauses in drilling should not be reduced, but it is important to consider cause and effect. The task is to optimize the sequence. A bad sequence cannot be measured by how long pauses in drilling are.

It is important to normalize the result with respect to total drill time, as done in figure 4.10. This is because the total drill time varies with respect to the number of holes in the

drill plan, and the quality of the rock. The averages for each of the histograms in figure 4.10 is presented in table 4.1.

	Average normalized transition time
Boom 1	32,53 %
Boom 2	29,98 %
Boom 3	30,11 %
Total	30,88 %

Table 4.1: Averages from figure4.10

The averages for each boom is approximately the same. It is interesting that the center boom has approximately the same average time as the two other booms. This means that there is no significant difference having a boom on each side to having a boom only at one side and a rock surface on the other side. The average transition time for each boom also provides a max for the potential of optimizing the drill sequence. The max possible time savings cannot exceed about 30.88% of the total drill time.

Another way of measuring the efficiency of the sequence is to measure the distance moved for each boom for each hole. Measuring the distance a boom moves to drill a hole is an efficiency measurement of the sequence without other time losses like waiting for the operator to be available. The less movement the more optimal a sequence is. A histogram of distance moved for each hole is presented in figure 4.11. The data is collected from all the drill logs. The average distance the boom is moved to drill a hole is 1.22 meters and the variance is 1.32 meters squared. This is very relevant numbers to compare to the efficiency of the algorithm proposed in this thesis.

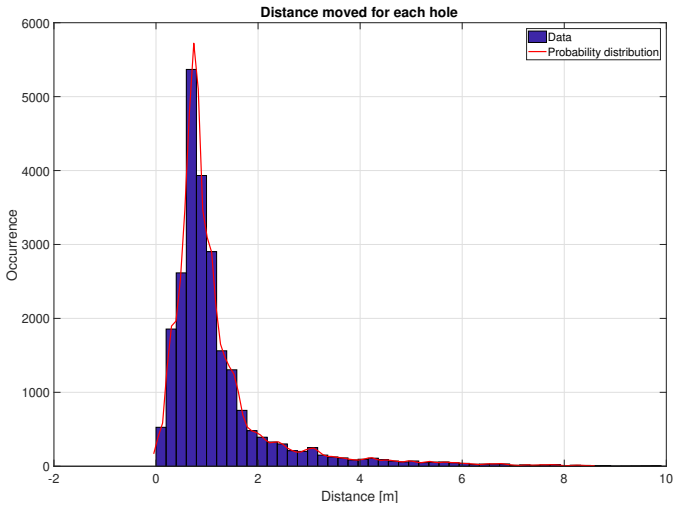


Figure 4.11: Histogram of distance moved for each drilled hole.

When drilling a sequence a sub-goal of efficiency is for the booms to finish working at the about same time, or finish synchronized. In that way little time is lost from booms being inactive at the end of the sequence. Measuring this is done by calculating the time difference between the first boom finishing and the last boom finishing. An example of a sequence done with a non-optimal end synchronization is displayed in figure 4.12. The Gantt diagram shows that when boom 2 finishes boom 1 still have 1 hole left and boom 3 still have about 4 holes left. It would be more optimal if some of the holes drilled in the end by boom 3 could be reallocated to boom 1 or boom 2.

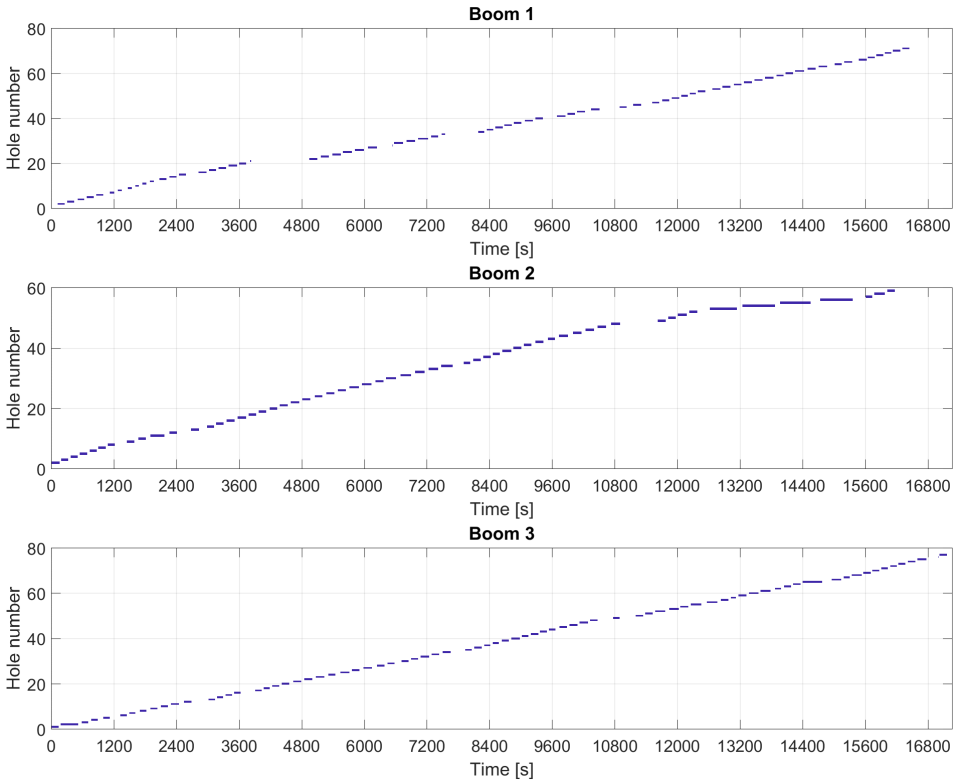


Figure 4.12: Gantt diagram of a sequence with a non-optimal end synchronization

Gathering the time between first boom finishing and last boom finishing for a set of sequences from Kjørholttunellene provides the histogram showed in figure 4.13. The average time between first boom finishing to the last boom finishing is 1511 seconds or 25.18 minutes. This is an unexpectedly high value. 1511 seconds is the same as the drilling time of 12.8 holes using the average drill time calculated from figure 4.9. There is a high potential to improve this value by making sure the holes are evenly allocated to the booms throughout the sequence.

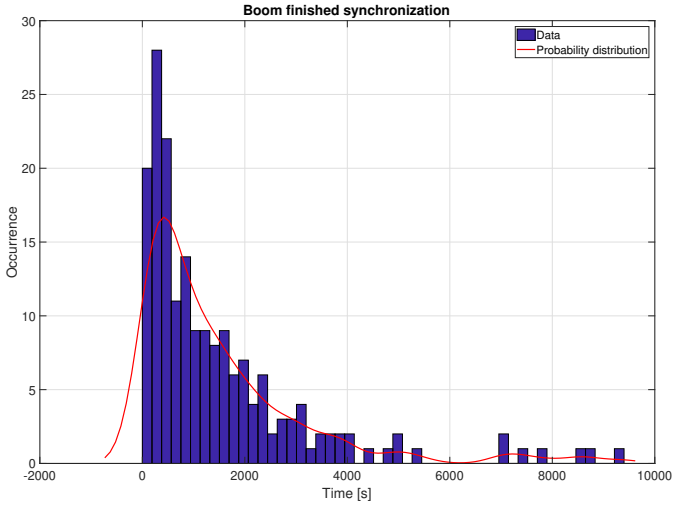


Figure 4.13: Time difference between the first boom finishing and the last boom finishing.

Chapter 5

Optimizing the drill sequence

The aim of this section is to provide an algorithm for finding the optimal way to drill a set of holes with a three boom drilling rig, as the rig described in subsection 1.4.2. Figure 5.1 displays a flowchart of the main steps of the algorithm. First, the drill plan is loaded, followed by allocation of the holes to the three different booms. Next, the optimal with respect to collision avoidance is calculated. This provides a sequence for each boom. Each boom then starts drilling the next hole in their sequence. When a hole finishes a reallocation of the remaining holes is calculated before calculating the new optimal solution and drilling the next hole. This dynamically updates the solution to consider unknown drilling times and assures that the booms always drills in the most effective way.

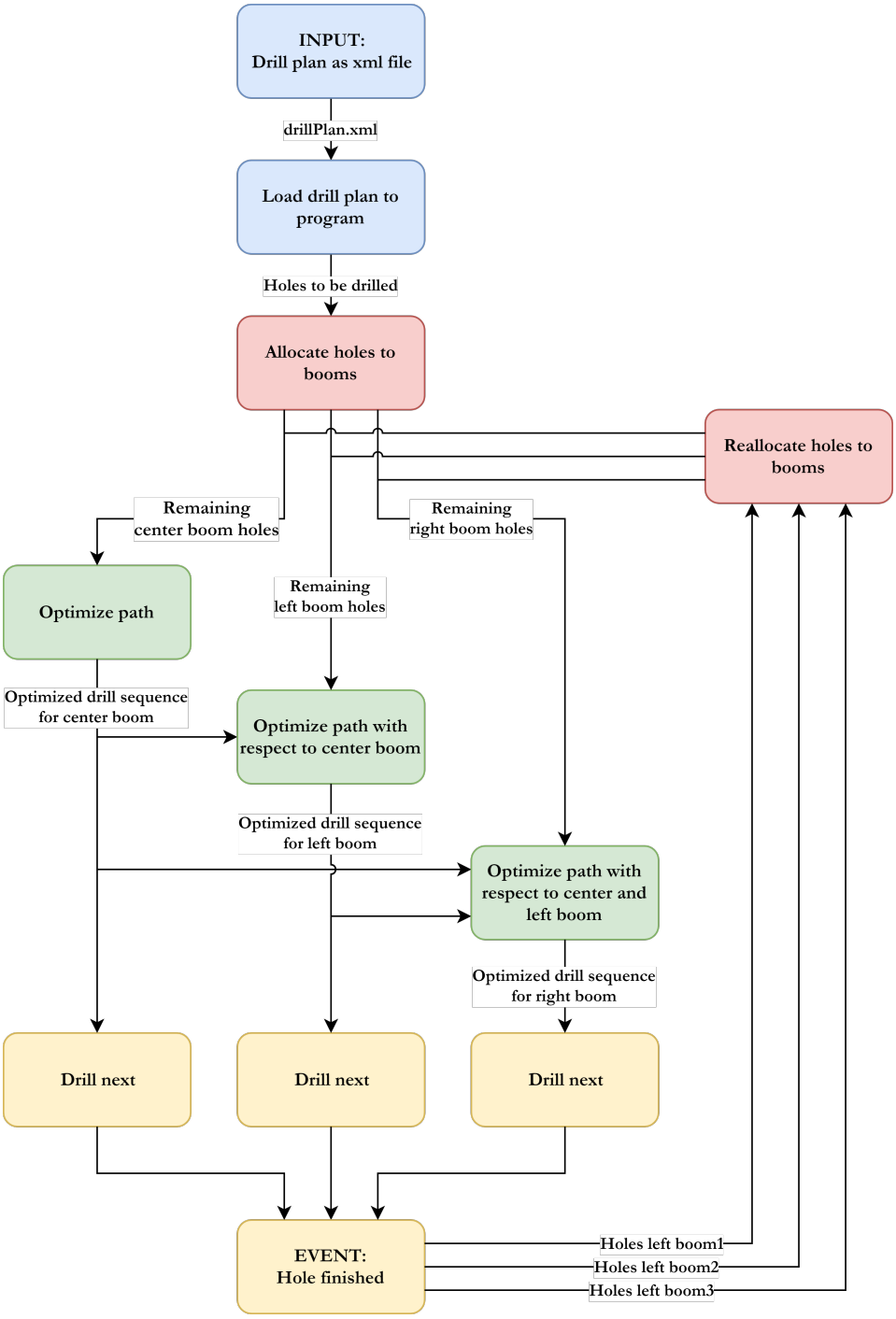


Figure 5.1: Workflow of main algorithm

First, the holes need to be allocated to each boom, and then a traveling salesman problem can be solved for each of the sets of holes.

5.1 Allocation of holes to booms

The traveling salesman problem could theoretically be solved for all possible hole allocations. This would result in a number of options described in equation 5.1.

$$nOptions = \begin{pmatrix} nHoles \\ nBooms \end{pmatrix} \tag{5.1}$$

The number of booms is 3. With number of holes as 174 as in the drill plan in figure 3.1 the number of options become 862924. Solving a traveling salesman problem that many times would not be practical. The runtime of the program would be too large.

With that in mind, it is more practical to derive a tactic based on experience. This will give an approximately optimal solution to the problem. The area in which each boom usually works is discussed in section 4.2. Based on the observations from figure 4.6 a algorithm for hole allocation can be created. The first goal is to separate the holes evenly with a vertical line, as in figure 4.6. Then by defining the center of the circle i figure 4.6 at the point where the vertical line crosses the y-axis. Then the closest point to the center is selected from every other side until the holes are evenly allocated. The method is described more in detail in algorithm 5.1.

```
input  : A list of n holes parameterized with (x_i, y_i) called: holes
output : Three list of holes: leftBoom, rightBoom, centerBoom
Sort holes with respect to x;
centerPoint = (holes(n/2),0);
leftBoom = holes(1 : (n/2));
rightBoom = holes(((n/2) + 1) : end);
while Not evenly allocated do
    if length(leftBoom) >= length(rightBoom) then
        Calculate hole from leftBoom closest to centerPoint;
        Append the closest hole to centerBoom;
        Remove the closest hole from leftBoom;
    else
        Calculate hole from rightBoom closest to the centerPoint;
        Append the closest hole to centerBoom;
        Remove the closest hole from rightBoom;
    end
end
```

Algorithm 5.1: Blast hole allocation algorithm

The first goal of the algorithm is to divide the set of holes into two equal parts using a vertical line. This is done by sorting the list of holes with respect to x and then picking the middle value. If the rig has only two booms, the task is done. For a three-boom rig, some holes need to be allocated to the center boom. Along the vertical line at y=0, a "centerPoint" is defined. This point provides a center for all of the holes allocated to the center boom. In the while loop of the algorithm one hole a time is allocated to the center boom. The closest hole from the boom with the most holes is picked. Holes are allocated to the center boom until holes per boom are approximately the same. Approximately the same means that the holes are not always divisible by three. If not, either one or two of the booms need to drill one more hole. The result of this algorithm applied to the drill plan for figure 3.1 is presented in figure 5.2.

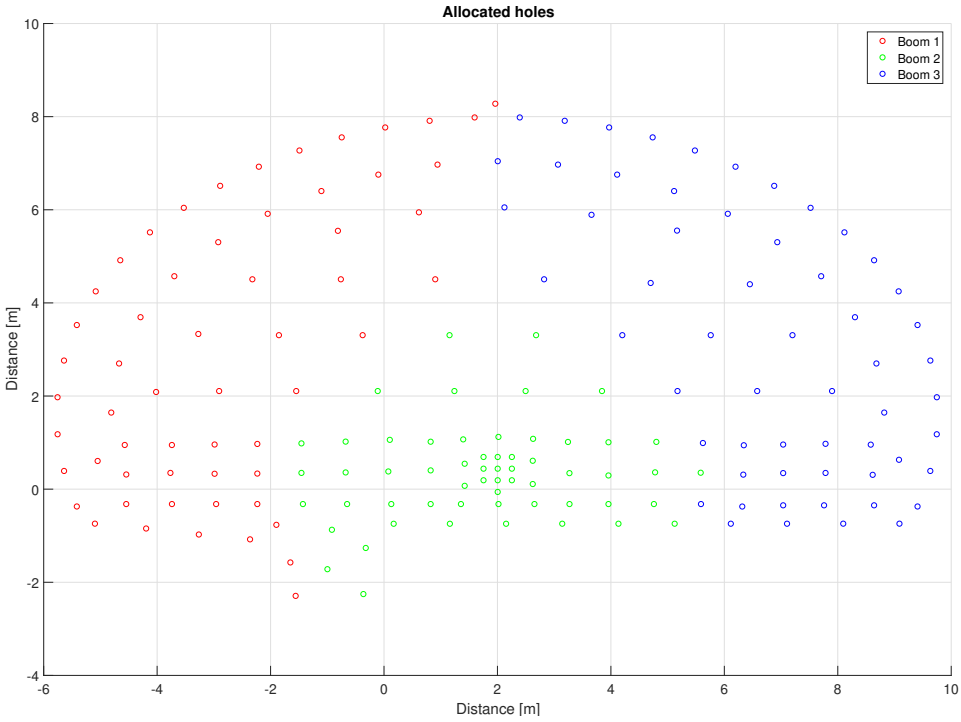


Figure 5.2: Holes allocated by algorithm 5.1

Figure 5.2 has a strong resemblance to figure 4.6 when it comes to booms at each hole. This was the goal of the algorithm. The experience learned from previously drilled sequences has been put to use to allocate holes to each boom. Additional plots for different drill plans are provided in appendix A. The algorithm is invariant to the size of the drill plan or the placement of the holes.

For allocating the bolt holes in the drill plan algorithm 5.2 is used. The set of holes is divided into three equal parts, one to the left, one in the center and one to the right.

input : A list of n holes parameterized with (x_i, y_i)
output: Three list of holes: leftBoom, rightBoom, centerBoom
Sort holes with respect to x;
divisor=$(n - n\%3)/3$;
left = holes(1:divisor,:);
center = holes(divisor+1:2*divisor,:);
right = holes(2*divisor+1:end,:);

Algorithm 5.2: Bolt hole allocation algorithm

Plotting both the allocated blast holes and bolt holes in the same 3D space provides figure 5.3.

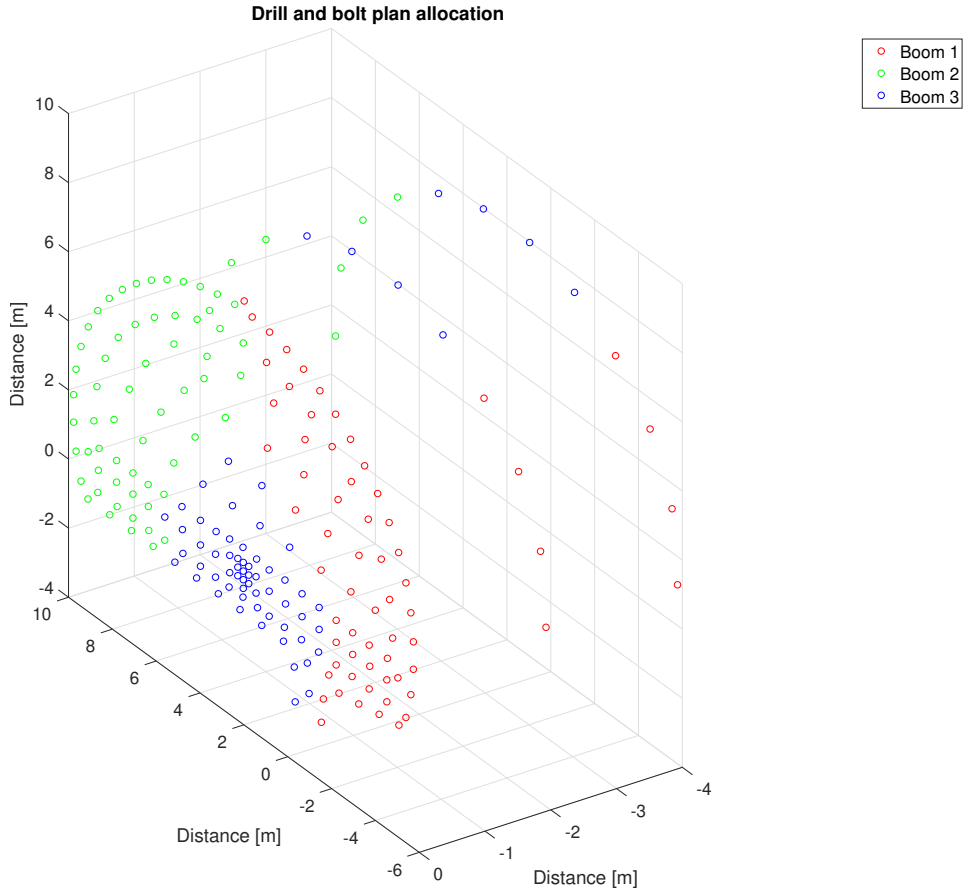


Figure 5.3: Holes allocated by algorithm 5.1 and algorithm 5.2

5.2 Shortest path

Now that each boom has a set of holes to drill, it is relevant to consider the sequence for each individual boom. The goal is to find the shortest path that spans all holes for each boom. The traveling salesman problem as described in section 2.1 can be applied to this problem. Each hole is a city and the goal is to visit each hole once to drill it. A solution to the traveling salesman problem in this case would result in a way to drill all holes with the least amount of movement for the booms. Minimizing the movement of the booms will reduce movement time and as a consequence reduce the time used to drill a sequence.

Matlab code is implemented to solve the traveling salesman problem in the case of drilling holes in the drill plan. The solution is coded as the specially designed function in code snippet 5.1 which solves the problem using integer programming as discussed in section 2.1.

```
1       function [startPoint, endPoint] =
    ↪  getShortestPath(holes,startHole,endHold)
```

Code snippet 5.1

The input of the function is a list of holes where each point has a x and y value. In addition, the index of a start hole and an end hole is provided. The purpose of the function is to define and solve problem statement 2.3. The integer linear programming problem is repeated in 5.2

$$\min \quad \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \tag{5.2a}$$

$$\text{s.t.} \quad 0 \leq x_{ij} \leq 1 \qquad i, j = 1, ..., n \tag{5.2b}$$

$$u_i \in \mathbb{Z} \qquad i = 1, ..., n \tag{5.2c}$$

$$\sum_{i=1}^{n} \sum_{j=i}^{n} x_{ij} = n - 1 \tag{5.2d}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, ..., n \tag{5.2e}$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, ..., n \tag{5.2f}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n \tag{5.2g}$$

The first task is to define the decision variable x as defined in equation 2.2 in the theory section. The decision variable is a vector with length equal to all the possible segments between the holes, with a length of `nCr(nHoles,2)`. To define all the different segments the matrix idxs is defined in code snippet 5.2.

```
nHoles = length(holes)
idxs = nchoosek(1:nHoles,2);
```

Code snippet 5.2

The code in 5.2 provides a matrix of all possible segments between the holes. The resulting idxs matrix is represented mathematically in equation 5.3. The first segment is from hole 1 to hole 2, the second segment is from hole 1 to hole 3 and so on. Each row represents a segment. The decision vector x will then correspond to which of these segments are used for a solution of the problem and as a result x will have the same length as the segment list.

$$idxs = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ \vdots & \vdots \\ 1 & nHoles \\ 2 & 3 \\ 2 & 4 \\ \vdots & \vdots \\ nHoles - 1 & nHoles \end{bmatrix} \tag{5.3}$$

The distance vector c has the same length as x and contains the distance between all the different holes, or the length of the segments defined in idxs. The code for calculating all the segment distances is presented in code snippet 5.3. c is then a vector with the same length as idxs with corresponding segment lengths.

```
h_x = holes(:,1);
h_y = holes(:,2);
c = hypothp_x(idxs(:,1)) - h_x(idxs(:,2)),h_y(idxs(:,1)) -
↪  h_y(idxs(:,2)));
```

Code snippet 5.3

This completes the variables used for the objective function presented in equation 5.2a. Constraint 5.2b and 5.2c is defined in code snippet 5.4. All decision variables is integers and this constraint is represented by a vector intcon with one for each decision variable that is an integer. All decision variables are integers, so intcon is a vector of only ones. The lower boundary is zero and an upper boundary is one for each decision variable x.

```
lendist = length(c);
intcon = 1:lendist;
lowerBoundray = zeros(lendist,1);
upperBoundary = ones(lendist,1);
```

Code snippet 5.4

Next is the equality constraints in equation 5.2d, 5.2e and 5.2f. A matrix of coefficients Aeq

and a vector of coefficients beq has to be calculated and has to include all three equality constraints.

The first equality constraint 5.2d, constrains the problem to only contain $n-1$ segments. This is for a path returning to the start point. For a path not returnin to the start point the value has to be $n-2$. The code for implementing this is presented in code snippet 5.5. This sets the first row of Aeq to a row where each element is one. When a row where each element is one is multiplied by the decision vector the result will be the sum of all decision variables in the decision vector equal to one. This sum has to be exactly the value of the first row of beq $n-2$. This results in just enough segments to reach all holes without making a return to the start point.

```
Aeq = spones(1:length(idxs));
beq = nHoles-2;
```

Code snippet 5.5

Constraint 5.2e and 5.2f is merged into one in the implementation. Each hole has to be arrived at once, and departed from once. That translates to each hole needs to be included in exactly two different segments. This is valid for all holes that are not start and end holes. For the start and end hole, there needs to be exactly 1 segment including the hole. The code for implementing constraint 5.2e and 5.2f is presented in code snippet 5.6.

```
Aeq = [Aeq;spalloc(nHoles,length(idxs),nHoles*(nHoles-1))];
for i = 1:nHoles
    whichIdxs = (idxs == i);
    whichIdxs = sparse(sum(whichIdxs,2));
    Aeq(i+1,:) = whichIdxs';
end
beq = [beq; 2*ones(nHoles,1)];
beq(startHole+1) = 1;
beq(endHole+1) = 1;
```

Code snippet 5.6

This adds a row to Aeq for each hole. All values in this row corresponding to a segment where the hole is included is set to one. The corresponding addition to the beq vector is the number 2. This lets exactly two segments and that includes that hole to be selected. A row for each hole makes sure that each hole has only two segments connected to it.

Aeq ends up being a matrix with size $[1 + nHoles, nCr(nHoles, 2)]$, and beq ends up as a vector of size $[1 + nHoles, 1]$. This concludes the equality constraints. Now a go at the optimal solution with all constraints so far is performed in code snippet 5.7.

```
1       opts = optimoptions('intlinprog', 'Display', 'off',
        ↪  'Heuristics', 'round-diving', 'IPPreprocess', 'none');
2       [x,costopt,exitflag,output] = intlinprog(c, intcon, [], [],
        ↪  Aeq, beq, lowerBoundray, ub, opts);
3
4       tours = detectSubtours(x,idxs); %Function designed to return
        ↪  all subtours
5       numTours = length(tours);
```

Code snippet 5.7

This provides an optimal solution without including the subtour constraint. If there is more than one subtour, these need to be eliminated in the optimal solution. This brings constraint 5.2g, or the no subtour constraint. Implementing this constraint can be done dynamically by introducing new inequality constraints for each result of the optimization that has more than one tour.

```
1   A = spalloc(0,lendist,0);
2   b = [];
3   while numTours > 1
4       b = [b;zeros(numtours,1)];
5       A = [A;spalloc(numtours,lendist,nHoles)];
6       for i = 1:numtours
7           rowIdx = size(A,1)+1;
8           subTourIdx = tours{i};
9           variations = nchoosek(1:length(subTourIdx),2);
10          for j = 1:length(variations)
11              whichVar = (sum(idxs==subTourIdx(variations(j,1)),2)) &
                ↪  (sum(idxs==subTourIdx(variations(j,2)),2));
12              A(rowIdx,whichVar) = 1;
13          end
14          b(rowIdx) = length(subTourIdx)-1;
15      end
```

```
16
17      [x,costopt,exitflag,output] = intlinprog(c, intcon, A, b, Aeq,
        ↪  beq, lowerBoundray, upperBoundary, opts);
18
19      tours = detectSubtours(x,idxs); %Function designed to return
        ↪  all subtours
20      numTours = length(tours);
21  end
```

Code snippet 5.8

One way to implement the no sub-tour constraint is in code snippet 5.8. After the program tried to optimize with no inequality constraints, it checks how many tours there are. If there is only one tour, the optimization is successful and the algorithm completes. If there are more than one tours it collects all variables associated with the current subtours and then adds an inequality constraint to stop that sub-tour from existing in the optimal solution. The code runs recursively until there is only one tour. When that happens, the code in code snippet 5.9 runs. The code snippet provides a list of start points and a list of endpoints. Together they represent a list of segments in correct sequential order that are returned from the function.

```
1   theTour = cell2mat(tours(1,1));
2   startPoint = zeros(length(theTour)-1,2);
3   endPoint = startPoint;
4   for i=1:(length(theTour)-1)
5       startPoint(i,1) = holes(theTour(i),1);
6       startPoint(i,2) = holes(theTour(i),2);
7       endPoint(i,1) = holes(theTour(i+1),1);
8       endPoint(i,2) = holes(theTour(i+1),2);
9   end
10  return startPoint, endPoint;
```

Code snippet 5.9

Using this function three times, one for each boom, on the allocated drill plan in figure 5.2 results in figure 5.4. The figure shows the optimal path for each boom.

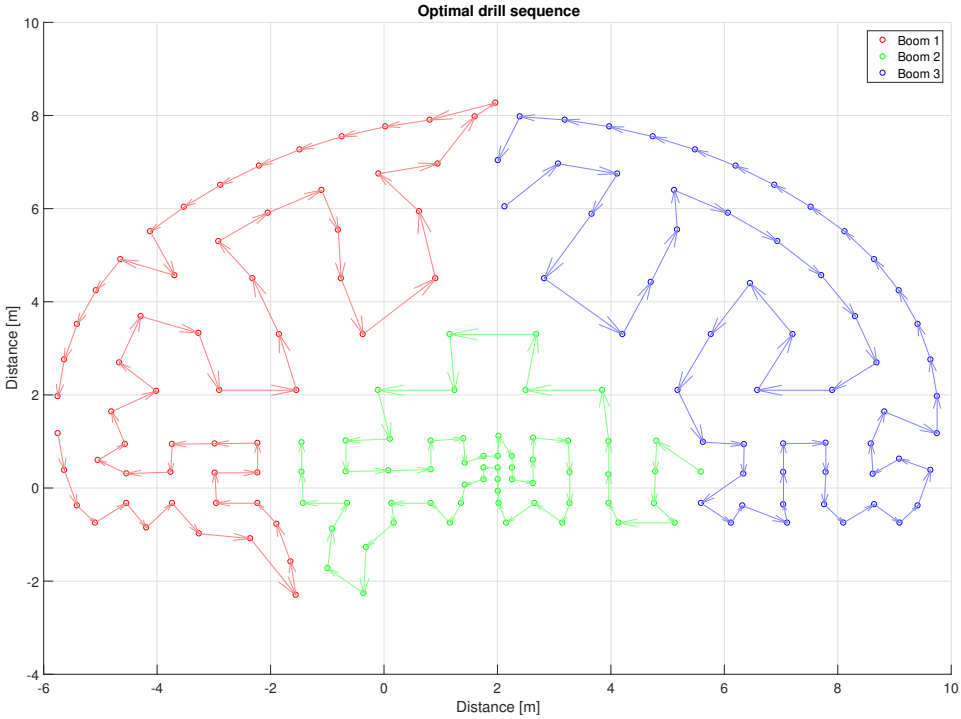


Figure 5.4: Optimal sequence for each boom

The best way to select start and end holes for the path would be to try all combinations of start and end holes. That would result in `nPr(nHoles,2)` options for each boom. In figure 5.4 each booms has 58 holes to drill. With nHoles=58 number of options is 3306. The calculation time for that many traveling salesman problems is not practically possible. It would take too much time to solve that many integer programming problems of this size. Running the optimization 50 times with random start and endpoints show that the selection of start holes and end holes for the optimal path does not make a big difference. The results from running the algorithm 50 times are shown in figure 5.5. The difference in average moved distance for each hole is only 2-3 cm. This means that the start and end hole could be randomly picked without affecting the optimal solution significantly.

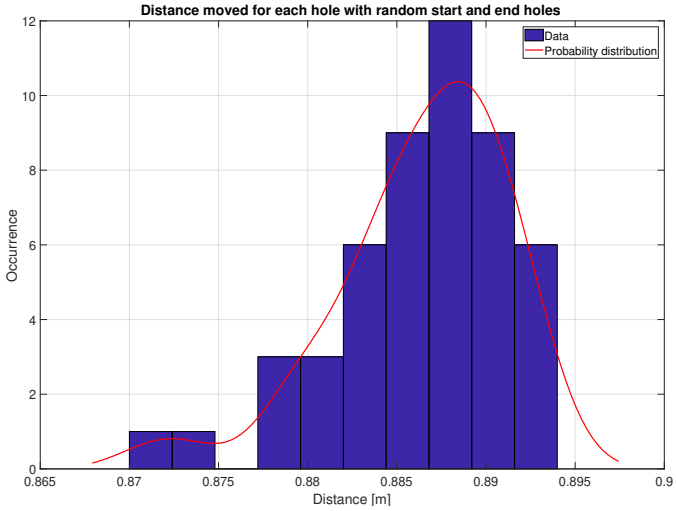


Figure 5.5: Histogram of distance moved for each boom with random start an end holes.

The optimal path for the two dimensional drill plan containing only blast holes is solved. When including the bolt holes in the roof of the tunnel the problem goes from two dimensional to three dimensional. The traveling salesman problem is still valid to use in a three dimensional space. The only change to the algorithm is changing the distance calculation from code snippet 5.3. The new code snippet is shown in code snippet 5.10. The third dimension has to be included in the calculation. Other than that, the algorithm stays almost the same. The dimension of the decision vector and all the other parameters for the optimization will have the same dimensions.

```
h_x = holes(:,1);
h_y = holes(:,2);
h_z = holes(:,3);
c = sqrt((h_x(idxs(:,1)) - h_x(idxs(:,2))).^2+(h_y(idxs(:,1)) -
↪  h_y(idxs(:,2))).^2+(h_z(idxs(:,1)) - h_z(idxs(:,2))).^2);
```

Code snippet 5.10

In figure 5.6 the optimal sequence for drilling all holes including bolt holes is showed in three dimensions

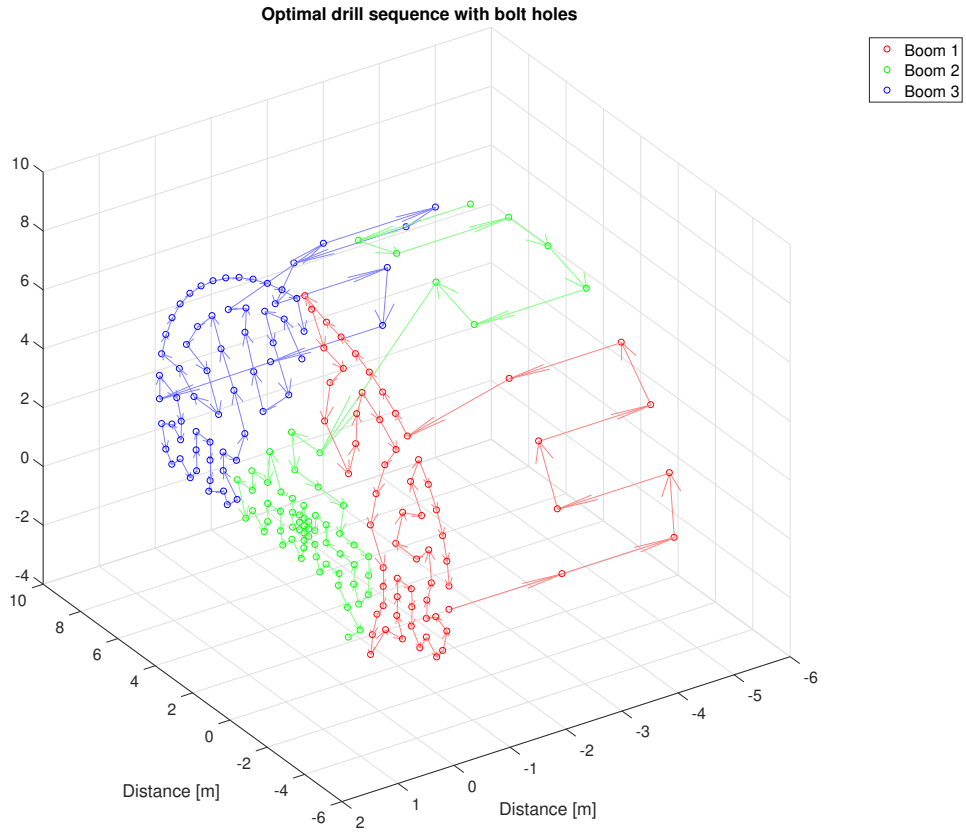


Figure 5.6: Optimal sequence for each boom with all holes in three dimensions.

Additional optimal paths for different drill plans without bolting is provided in appendix B. These plots show that the method can be used on drill plans of different size and composition.

5.3 Optimized solution with respect to collision avoidance

Calculating the optimized solution for each boom provides the system with the knowledge of where each boom will be in the future. It would be considered bad planning to let multiple booms collide because the holes they are drilling is too close to each other. With this in mind, a way to avoid collisions is implemented in the solution of the problem.

Which boom to calculate the optimal path for first matters because the first boom cannot collide with anything when no other paths are calculated. The center boom has a boom on each side and is, therefore, it is the hardest to avoid colliding with the other booms. It is natural to let it calculate its optimal solution first. Then the left and right boom can calculate its optimal solution with respect to the solution chosen for the center boom. It is also necessary to choose which of the left and right boom get to calculate its solution first. This does not matter because they work quite similar areas, only mirrored. This implementation has chosen the left boom before the right boom.

Consider a function detectCollision() that aims to calculate wherever the current solution will collide with any of the other sequences calculated. It returns true if the path contains a collision and false if not. If the function returns true, then a constraint preventing this solution should be defined.

In section 2.2 a method for calculating collision for robotic manipulators is explained. For the purpose of calculating if the booms are colliding in static position with no velocity equation 2.6 reduces to 5.4.

$$d_{nm}(\tau) - S < r_n + r_m \tag{5.4}$$

For calculation of collision between the three booms an algorithm can be formulated

input : Origins of all link centers for all booms parameterized in a common list P_n with corresponding r_n and a safety margin S
output : Collision: true or false
collision = false;
foreach *Component* $n \in P_n$ **do**
  foreach *Component* $m \neq n, n+1, n-1 \in P_n$ **do**
    Calculate μ_n and μ_m from equation 2.9;
    Determine configuration based on decision tree in figure 2.3;
    Calculate with correct equation from 2.7;
    if $d_{nm}(\tau) - S < r_n + r_m$ **then**
      return *true*;
    else
      return *false*;
    end
  end
end

Algorithm 5.3: detectCollision()

Implementing a simulation of the mechanics of the booms is not necessary to show that the optimal solution can be calculated with collision avoidance. Consider a sequence where there is no collision avoidance as in figure 5.7. Each hole is drilled at the time-steps numbered in the figure. Defining a simple collision detector as in algorithm 5.4 makes sure that each hole at each time-step will avoid being closer than a minimum distance to other holes at the same time-step.

input : The path of the boom to consider: thisBoom, the path of up to two other booms: otherBoom1 and otherBoom2, and a minimum distance: minDist
output : Collision: true or false
collision = false;
foreach *hole i in thisBoom* **do**
  if *distance(thisBoom(i), otherBoom1(i)) < minDist* **then**
    return *true*
  end
  if *distance(thisBoom(i), otherBoom2(i)) < minDist* **then**
    return *true*
  end
end
return *false*

Algorithm 5.4: detectCollision()

When a collision is detected, a new constraint has to be added to the inequality constraints. In code snippet 5.11 shows the code for adding this constraint. First, the current tour is saved as a matrix. The idxs matrix contains all the segments, as described in code snippet 5.2. All the segments corresponding to this tour is collected. The code creates a row where all segments used are set to one and the rest is set to zero. When multiplying with the decision vector with exactly these segments the result will be nHoles-1. Giving b the value nHoles-2 will prevent this solution from taking place since nHoles-1 is not less or equal to nHoles-2. All other solution will still be valid subject to this constraint, selecting only one different segment will make the constraint valid. The code is repeated until there is no collision. Each time the code hits a collision that exact solution will be prevented from existing by adding a new inequality constraint.

```
while collision
    collision = detectCollision()
    if collision
        theTour = cell2mat(tours(1,1));
        collisionTourIdxs = zeros(length(theTour)-1,2);
        for i=1:(length(theTour)-1)
            if theTour(i) < theTour(i+1)
                collisionTourIdxs(i,1) = theTour(i);
                collisionTourIdxs(i,2) = theTour(i+1);
            else
                collisionTourIdxs(i,1) = theTour(i+1);
                collisionTourIdxs(i,2) = theTour(i);
            end
        end

        constraintIdxs =
        ↪  sparse((ismember(idxs,collisionTourIdxs,'rows'))');
        A = [A;spalloc(1,lendist,nHoles)];
        b = [b;zeros(1,1)];
        A(end,:) = constraintIdxs;
        b(end) = nHoles-2;
    end
end
```

Code snippet 5.11

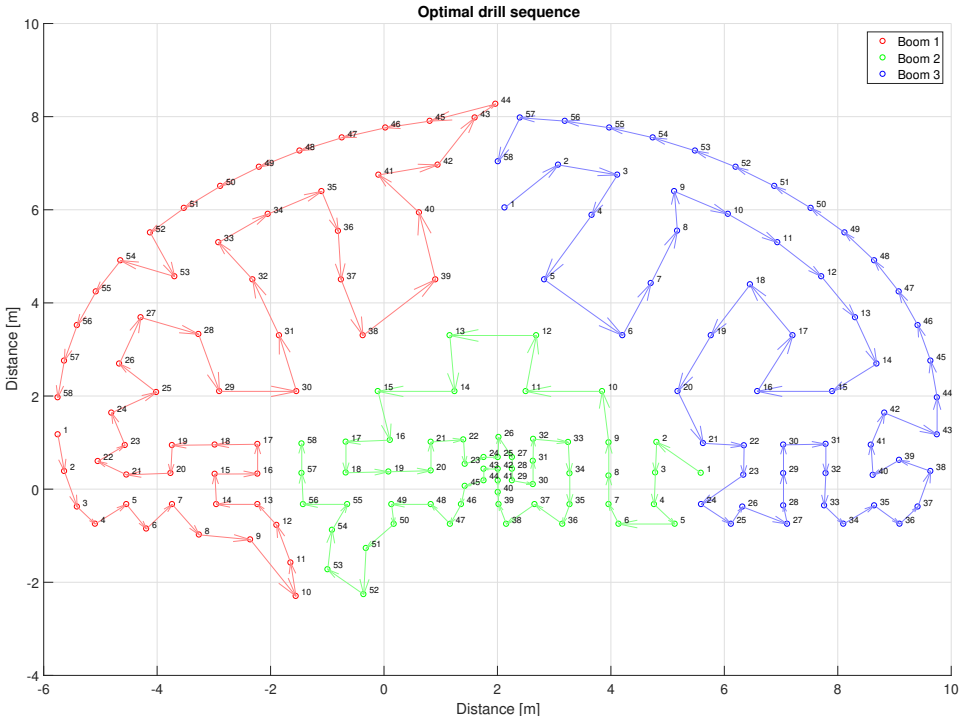


Figure 5.7: Drill times

In figure 5.7 the holes at time-step 17 for boom 1 and boom 2 are quite close to each other. By setting the minimum distance to two meters, these holes will cause a collision. By updating an inequality constraint to restrict all solutions with a collision within a minimum distance of two meters gives an optimal path as shown in figure 5.8. The path for boom 3 stays the same because there are no holes at the same time-step that are too close to each other. The marked area in figure 5.8 has a different path than in figure 5.7. The optimal path changes because of a constraint avoiding a collision. In figure 5.9 the minimum distance is set to three meters. This changes the path for boom 1 drastically to avoid a collision.

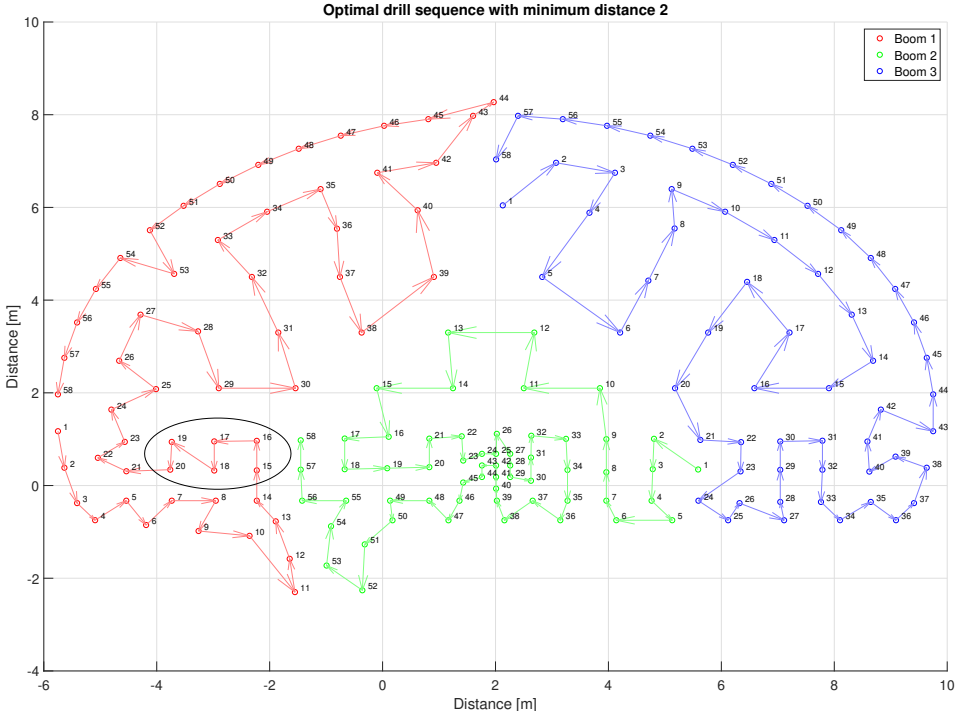


Figure 5.8: Drill times

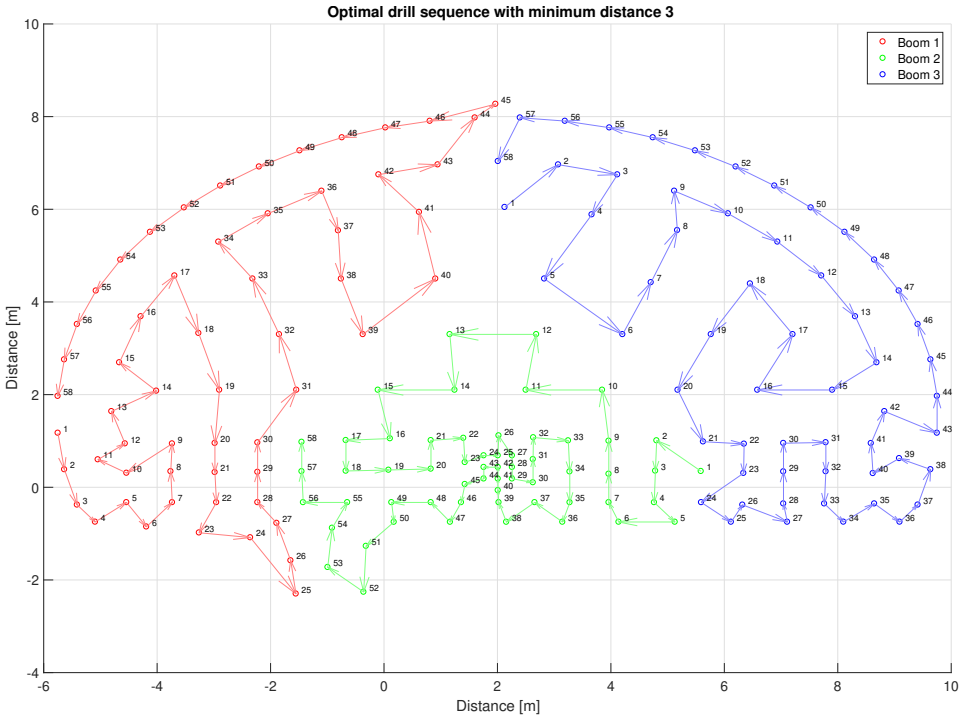


Figure 5.9: Drill times

5.4 Dynamic update of optimized solution

To determine if a dynamical update of the optimal solution is necessary, the drill time for all holes should be considered. If drill times vary significantly it would be optimal to reallocate holes from one boom to another in the event of uneven progress.

5.4.1 Reallocation of holes

A sub-goal of an optimal drilling is for all booms to be active as long as possible. This translates to all booms finishing at approximately the same time. For all booms to finish at the same time they need to drill approximately the same number of holes. The use of approximately refers to that the number of holes not always is divisible on the three booms. In section 4.3 and figure 4.9 the time for drilling a single hole is discussed. The drilling time has a significant variance, and cannot be predicted before drilling. When

some holes take longer or shorter time to drill the holes left to drill could eventually be allocated unevenly between the booms. As a consequence, there is a need for reallocation. The reallocation algorithm implemented is described in algorithm 5.5.

The purpose of the algorithm is to even out the numbers of holes remaining for each boom. If the booms have approximately the same number of holes, or the number of holes is within an error of one, there is nothing to do. The holes are as evenly distributed as they can be. If there are an uneven allocation of the remaining holes the algorithm takes a hole from the boom with the most holes and moves it to the boom with the least holes until the distribution is even. The hole closest to any of the holes in the destination boom is selected. Moving a hole from the left boom to the right boom is done through the center boom because the right and the left boom share few close holes. It would be inefficient to move a hole directly from the right boom to the left boom and vice versa.

```
input  : Three list of holes: leftBoom, rightBoom, centerBoom
output : Three list of holes: leftBoom, rightBoom, centerBoom
l = length(leftBoom);
r = length(rightBoom);
c = length(centerBoom);
if l, r and c is approximately the same then
    return
end
while l, r and c is approximately not the same do
    if l<=r AND l<=r then
        if c>=r then
            leftBoom <= The closest hole to leftBoom from centerBoom;
        else
            centerBoom <= The closest hole to centerBoom from rightBoom;
            leftBoom <= The closest hole to leftBoom from centerBoom;
        end
    else if r<=l AND r<=c then
        if c>=l then
            rightBoom <= The closest hole to rightBoom from centerBoom;
        else
            centerBoom <= The closest hole to centerBoom from leftBoom;
            rightBoom <= The closest hole to rightBoom from centerBoom;
        end
    else if c<=r AND c<=l then
        if r>=l then
            centerBoom <= The closest hole to centerBoom from rightBoom;
        else
            centerBoom <= The closest hole to centerBoom from leftBoom;
        end
    end
    l = length(leftBoom);
    r = length(rightBoom);
    c = length(centerBoom);
end
return
```

Algorithm 5.5: Hole reAllocation algorithm

An example of the use of the algorithm 5.5 is provided in figure 5.10 and figure 5.11. Figure 5.10 has an uneven allocation of holes to the three booms. There are 61 red holes, 55 green, and 58 blue. Running algorithm 5.5 provides figure 5.11 with 58 holes of each for each boom. The holes marked with the pink circle are the holes moved from the center boom to the left boom.

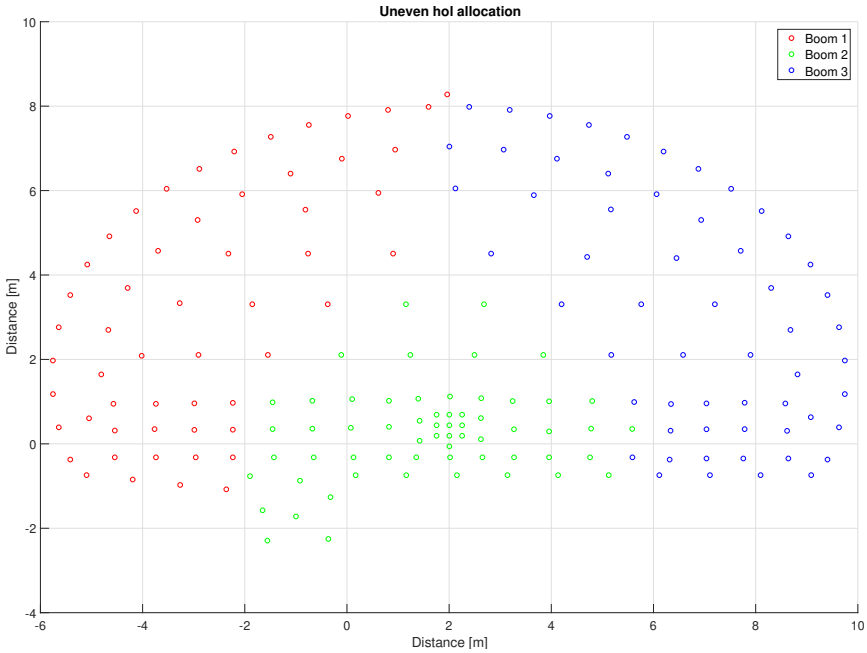


Figure 5.10: An uneven allocation of holes

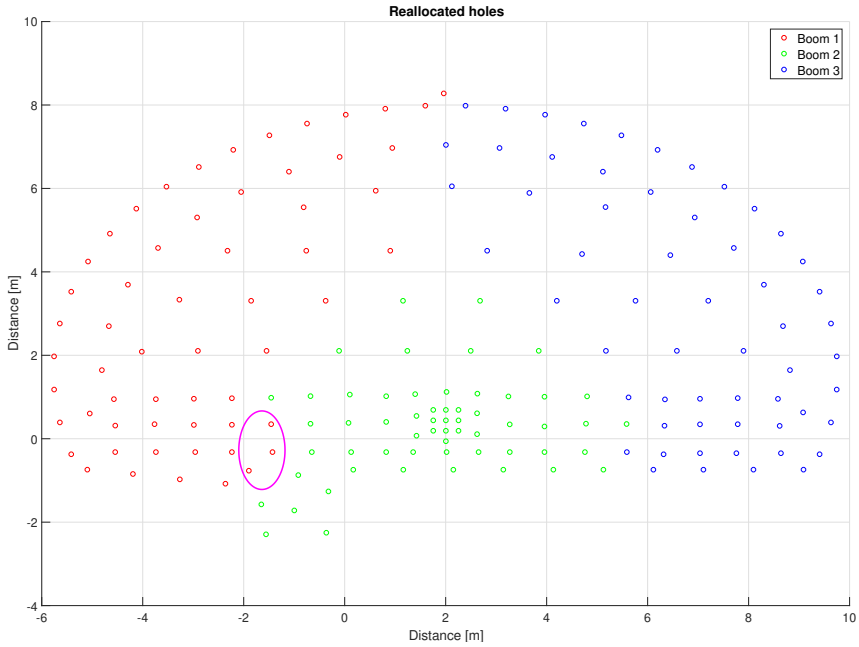


Figure 5.11: After running the reallocation algorithm. Reallocated holes marked with pink circle

This provides a tool for reallocation of holes during drilling. If one boom is slower than the others are, holes could be reallocated to another boom. After the reallocation, the new optimal path needs to be found for each boom.

5.4.2 Recalculating the optimal solution

When the remaining holes have been reallocated, a new optimal path needs to be found. This is done by using the same algorithm described in section 5.2. The start hole is selected as the current hole and the end hole are selected as the last hole added to the boom. The selection of the start and end holes was discussed in section 5.2. It is natural to select the start hole where the boom is at the current time, but it does not matter much where the end hole is selected.

In figure 5.12 a display of a drill sequence in progress is presented. The black holes are holes that are drilled and the colored holes are holes not yet drilled. The remaining holes are unevenly allocated in this figure. Running the remaining holes through the reallocation algorithm and calculating the new optimal path with respect to collision

avoidance provides figure 5.13. The right boom has two more remaining holes than the left boom. By reallocating one hole from the right boom to the center boom and one hole from the center boom to the left boom the remaining holes even out. The holes reallocated is marked with pink circles in figure 5.13. A new optimal path is calculated for each boom since all booms have had a change in allocated holes. The difference in the paths in figure 5.12 and figure 5.13 can observed by studying the figures. The dynamic update makes sure that the optimal solution is always relevant.

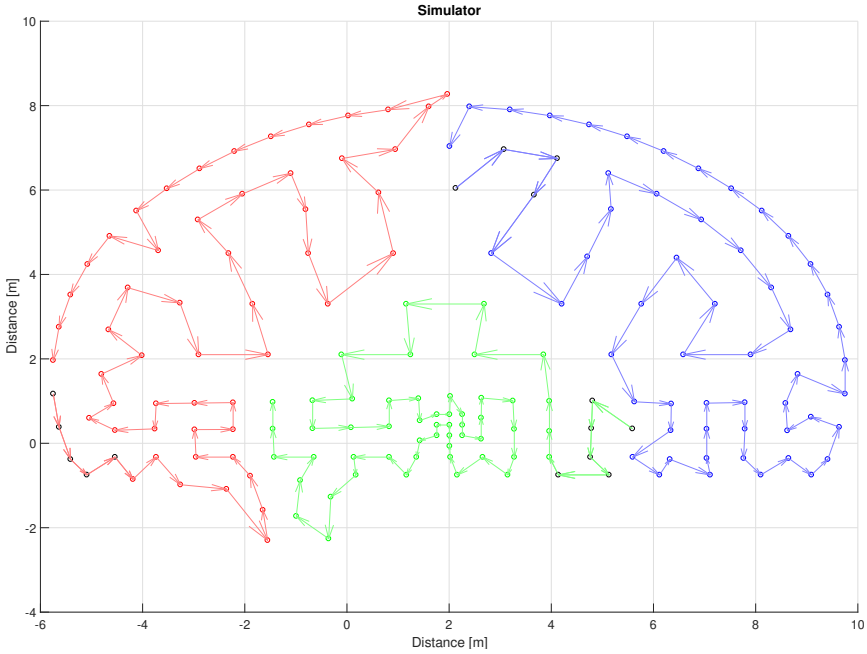


Figure 5.12: Before reallocation and new path, collision avoidance is set to minimum 1 meter

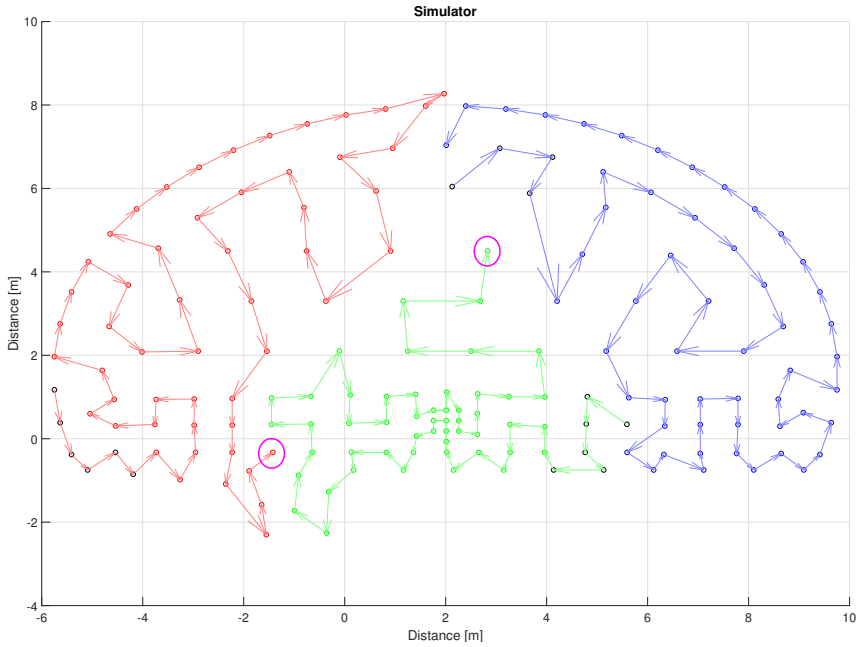


Figure 5.13: After reallocation and new path, collision avoidance is set to minimum 1 meter. Reallocated holes marked with pink circles

Chapter 6

Results and discussion

To verify the results of the algorithm a simulator has been implemented. The simulator lets each hole have a randomized drill time that matches the variable drill time discussed in section 4.3. The simulator lets each boom drill holes along the optimal path. The algorithm calculates the new optimal path dynamically. The path obtained by simulating the drilling will be different every time it is run due to the randomized drill time. A result of a simulation is shown in figure 6.1

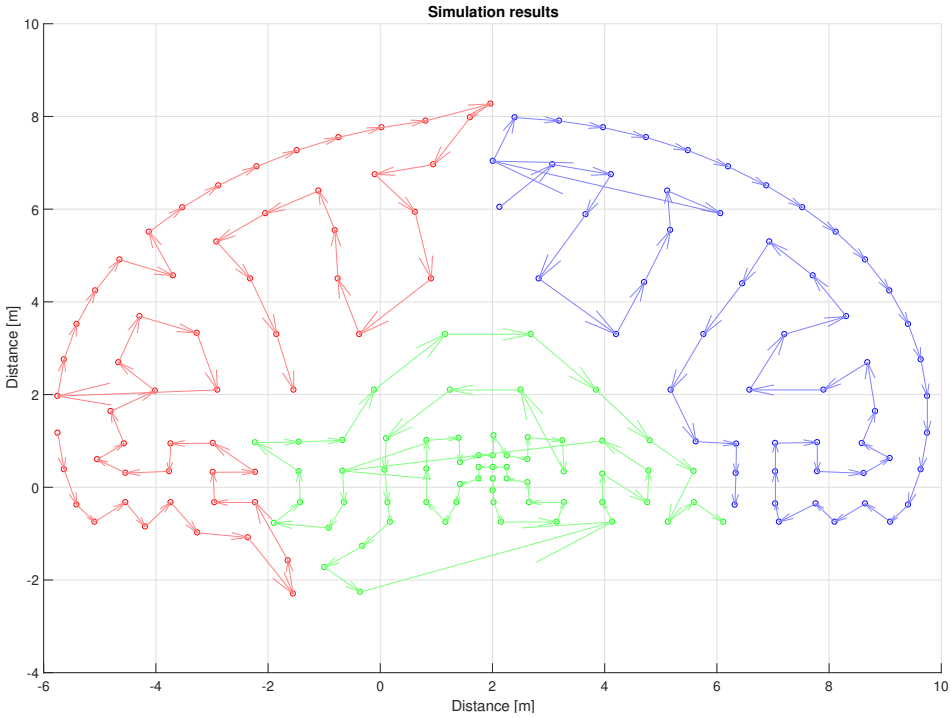


Figure 6.1: Simulation result

As the simulator runs, each hole is assigned a random drill time. For every hole finished, the program checks if there are any new optimal allocation and path. A video of a running simulation is attached to this thesis. In the video, the dynamic update of the algorithm shows clearly. At multiple timesteps, it is necessary to reallocate the holes and recalculate the optimal path.

Additional simulation for different drill plans have been provided in appendix C

6.1 Efficiency compared to current practice

In section 4.4 the efficiency of the drilling at current basis is discussed. This can be compared with the efficiency of the algorithm presented in this thesis. For current practice, the average distance moved to drill a hole is 1.22 meters. The calculated distance moved for the initial optimized path in figure 5.4 is 0.80 meters. That is only 65.57% of the current distance moved to drill a hole. In other words, the transition is reduced by 34.43%. This is

the initial optimal sequence. For this sequence to be true the drill time of each hole has to be the same. Currently the transitions is 30.88% of the total drill time. This results in 10.63% of the total drill time being saved.

To get a better measurement of the efficiency of the algorithm it is better to run the simulator a number of times to provide a distribution of distance moved including the dynamic update. This gives values that are including the variable drill time, as is the case in the real world. Running the simulator 200 times gives a histogram as shown in figure 6.2. The average distance moved to drill a hole is here 0.91 meters and the variance is 0.45 metes squared. The variance calculated from the drill logs is 1.32 meters squared. This shows that a computer-aided optimization algorithm as proposed in this thesis does not only reduce the time usage but also provides an algorithm that is more stable for all sequences than an operator choosing the sequence. These results provide a 25.41% save in distance moved and a possible 7.85% save in total drilling time. This is a huge efficiency improvement without introducing additional expenses for entrepreneurs.

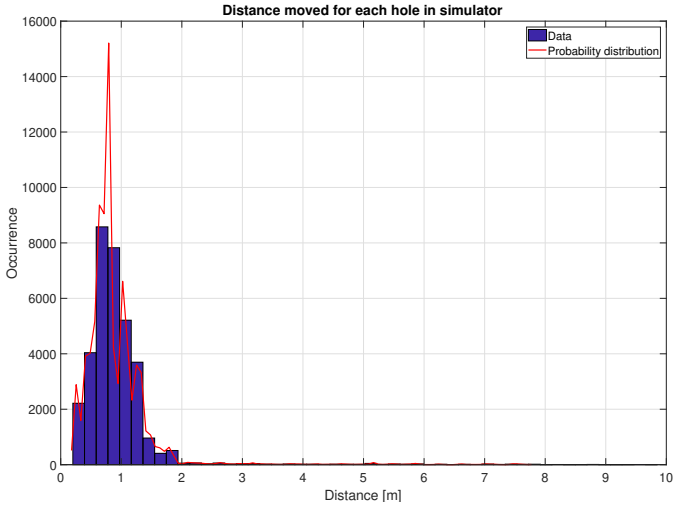


Figure 6.2: Distance moved for each hole in simulator

The finish synchronization discussed in section 4.4 can be measured in the simulator as well. With random drill times set to match the drill times collected in section 4.3 the end synchronization result for 200 runs of the simulator is presented in figure 6.3. The average deviation in finish time is about 190 seconds or 2.5 minutes. That is a big improvement to the average of 25.18 minutes found in the drill logs.

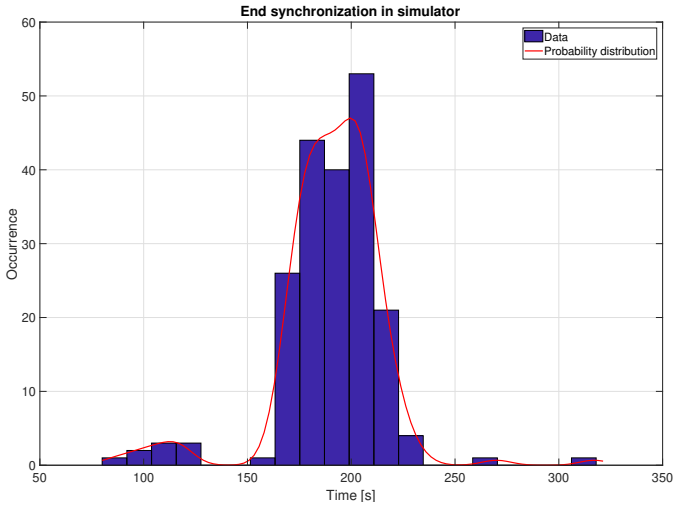


Figure 6.3: Time between first boom finishing and last boom finishing in simulator. Simulator run 200 times.

6.2 Automatic drilling

This contribution to automatically generating and updating a sequence for drilling is also a step towards making automatic drilling easier to use. When using this algorithm there is no need for manually inputting a sequence in the office before drilling. The algorithm is also flexible in the way that it updates dynamically so that the operator will not have to make changes in the sequence. The changes are done automatically.

It is not necessary to use automatic drilling to use the results of this thesis, the optimal path can just as well be presented in a graphic user interface to the operator. Then the operator can follow the optimal path for each boom while the computer automatically calculates new optimal paths.

6.3 Flexibility of algorithm

The algorithm recalculates the path for each hole drilled. The operator can drill a hole not recommended as the next hole if necessary. The algorithm will recalculate the new optimal path that starts at that hole and includes the rest of the holes. Having this flexibility will make the algorithm easier to use in practice. The operator does not have to push extra

buttons to turn the algorithm off and on.

Chapter 7

Conclusions and future work

This thesis have considered a method for generating sequences for drilling in tunnels. The analysis of the drill logs from Kjørholttunellene have resulted in an algorithm to allocate holes between the boom as well as a baseline for comparing the efficiency of the method presented in this thesis.

The calculation of the sequence is done by using a traveling salesman formulation, and solving it with integer programming. Constraints have been fitted to calculate an optimal solution with only one complete path for each boom with a possibility to select a start and a endpoint. Further a method for constraining the optimal solution to avoid collisions i discussed, implemented and tested in a simulator. The optimal path subject to collision avoidance minimizes the transition time from one hole to another.

The allocation of holes is dynamically updated to consider the variable drill time or other disturbances. A new optimal path is calculated when the allocation changes. This makes sure that the algorithm is flexible for disturbances, and makes sure that the allocation stays even so that all booms are max utilized.

The results from the method presented have been tested in a simulator. The results from the simulator provides a save in transition time of about 25 percentage, and a save in total drill time of about 8 percentage.

Future work would be field testing the method. With field testing better results than simulations can be obtained. Future work will also be adding the geometry of the booms to the collision detector.

References

Bosscher, P. and Hedman, D. [2009]. Real-time collision avoidance algorithm for robotic manipulators, *2009 IEEE International Conference on Technologies for Practical Robot Applications*, pp. 113–122.

Dantzig, G. B. [2016]. *Linear Programming and Extensions.*, Princeton Landmarks in Mathematics and Physics, Princeton University Press.

Edvartsen, T. M. [n.d.]. Databoring - hva hendte? Unpublished.

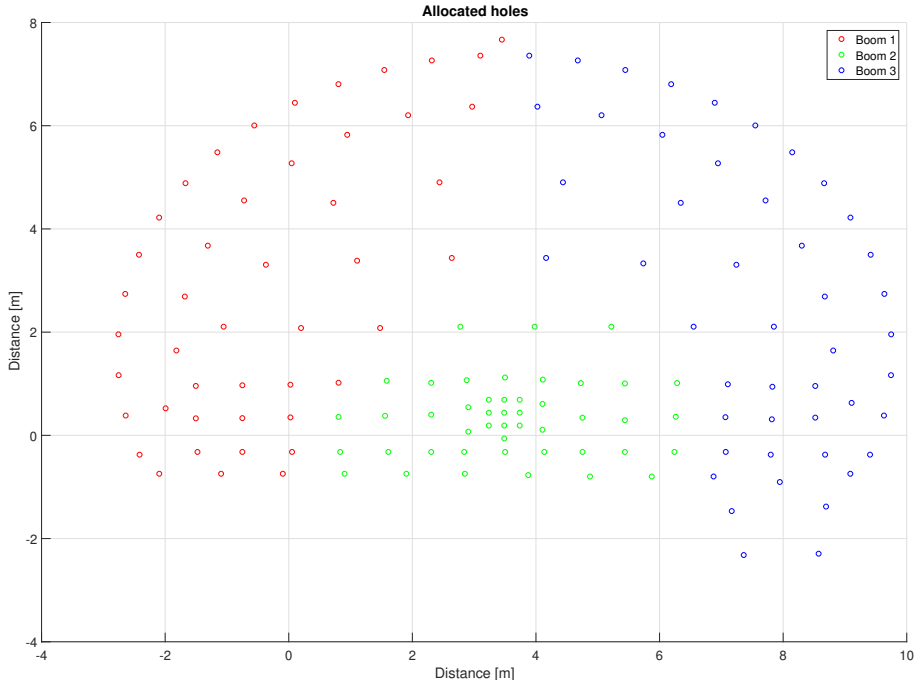
Eide, T. P. [2009]. Digital tunneling technology - challenges and possibilities, *FJELL-SPRENGNING / BERGMEKANIKK / GEOTEKNIKK* (7). http://www.rockmass.no.

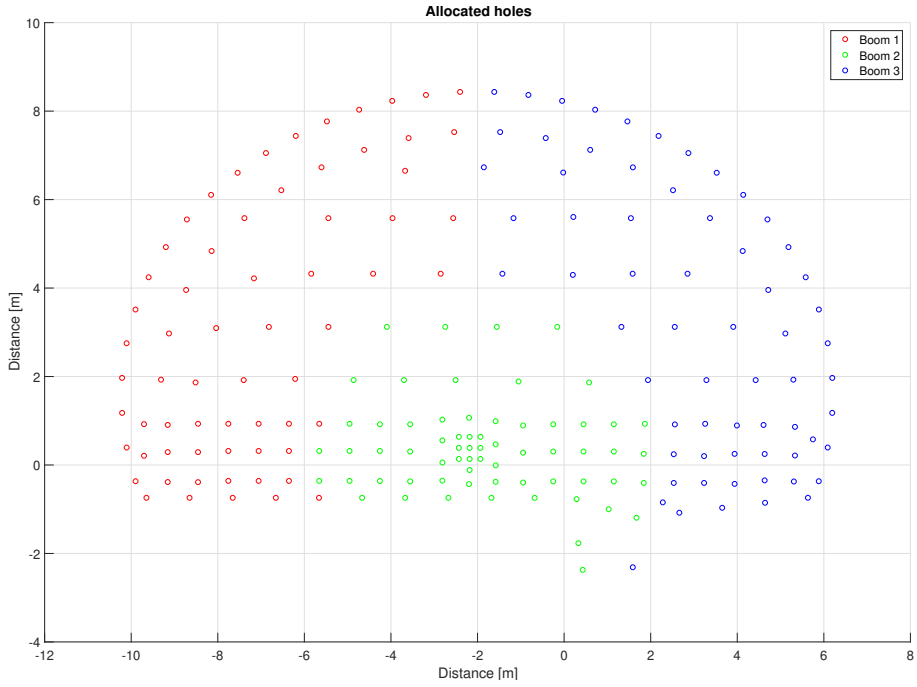
Ennen, P., Ewert, D., Schilberg, D. and S., J. [2014]. Efficient collision avoidance for industrial manipulators with overlapping workspaces, *Procedia CIRP* **20**(20): 62 – 66.

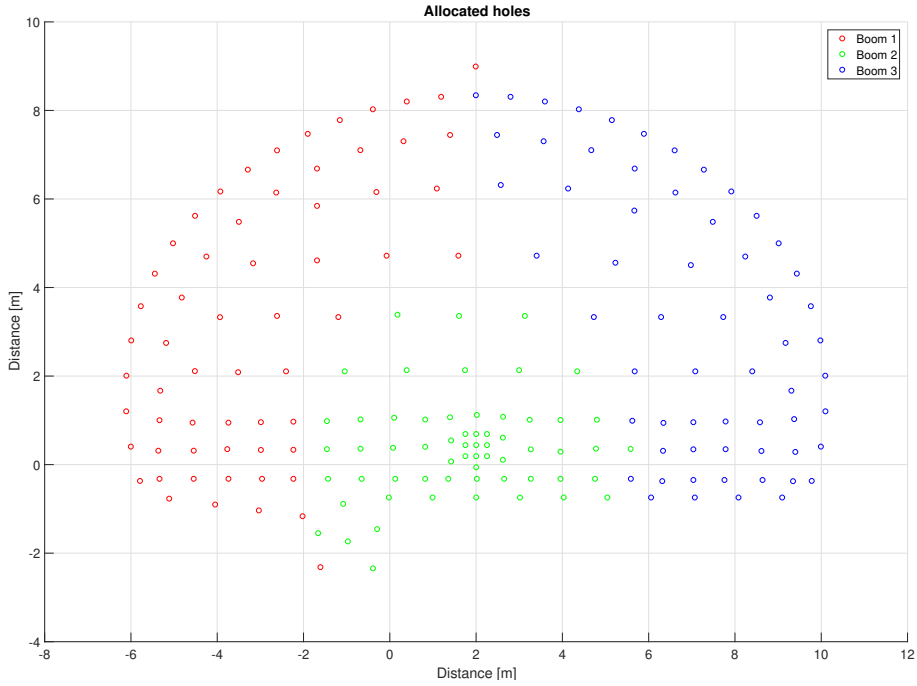
Helleseth, B. [2000]. *Driftsoppfølging skatestraumtunellen, heldata og dataflyt*, Master's thesis, NTNU - Department of Civil and Environmental Engineering.

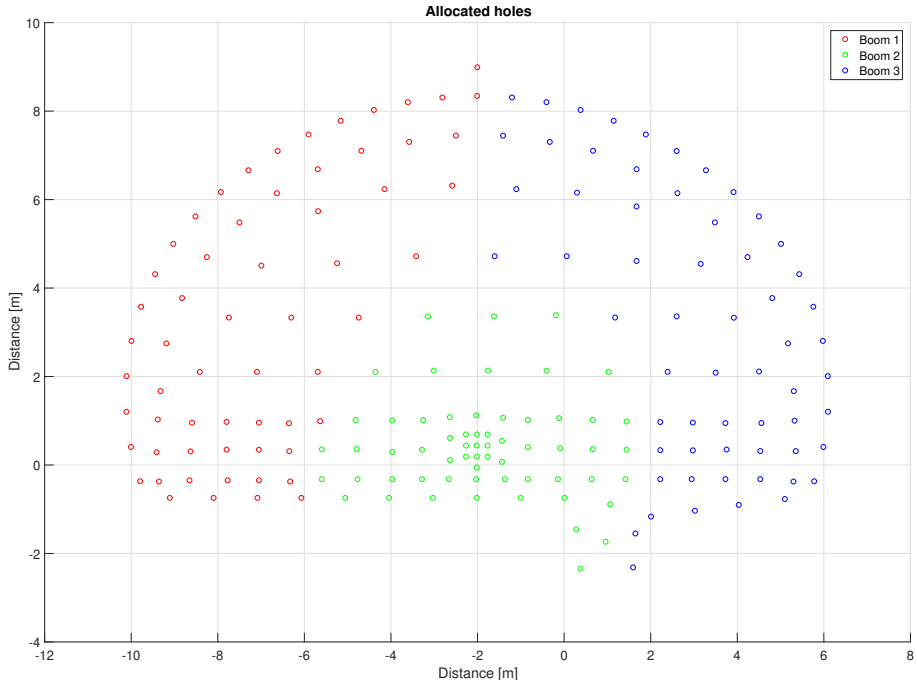
Nemhauser, G. L. and Wolsey, L. A. [1988]. *Integer and combinatorial optimization*, Wiley.

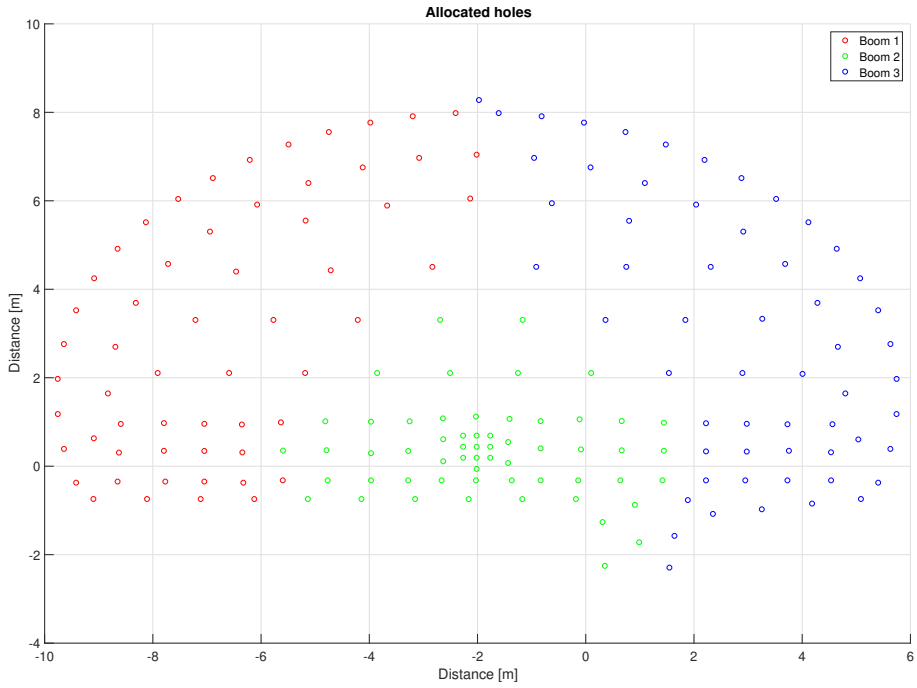
Appendix A: Additional boom allocation plots

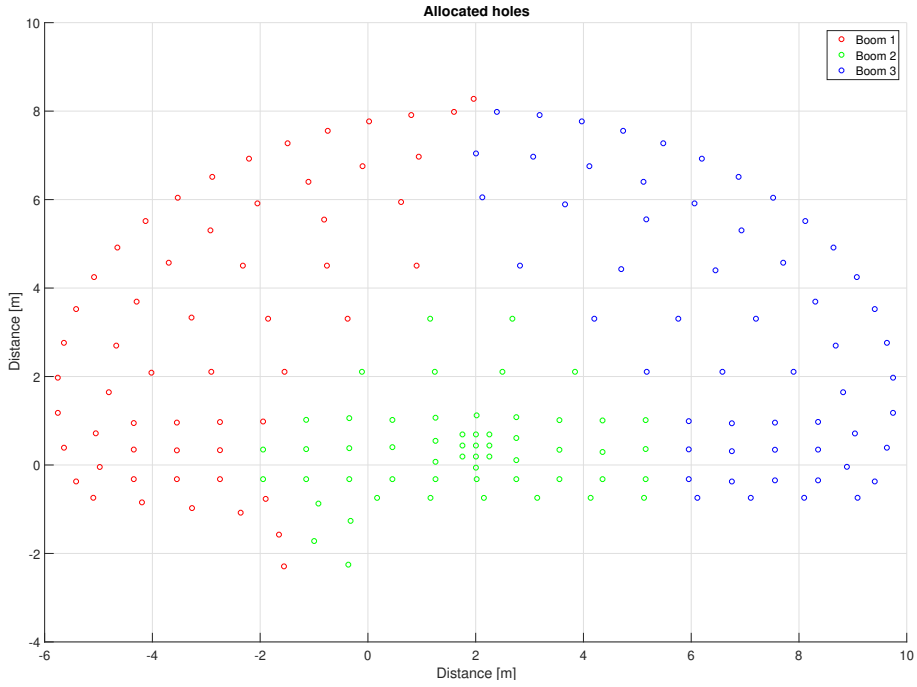


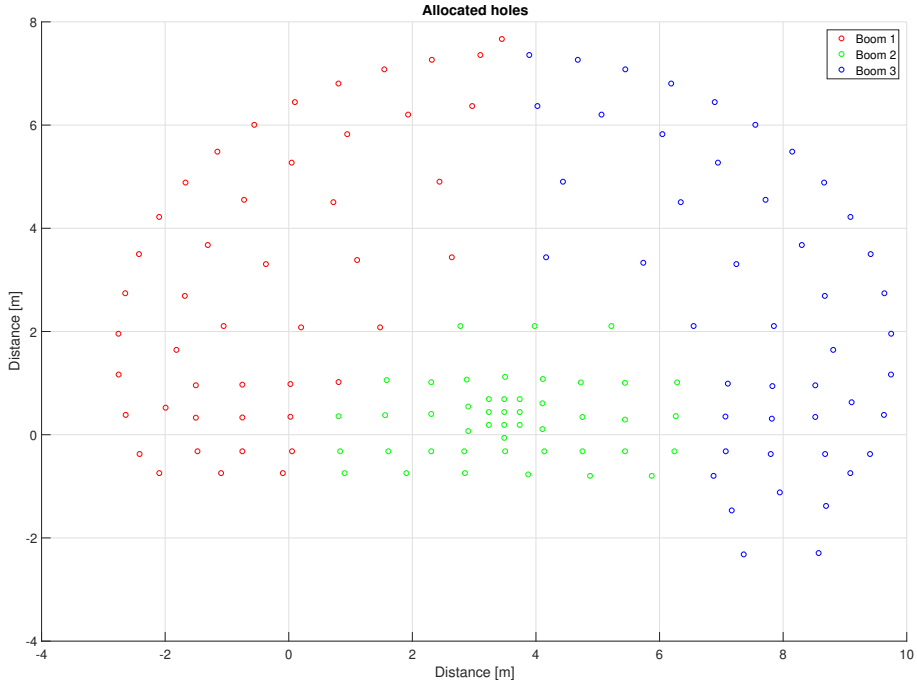

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-12
-10
-8
-6
-4
-2
0
2
4
6
8

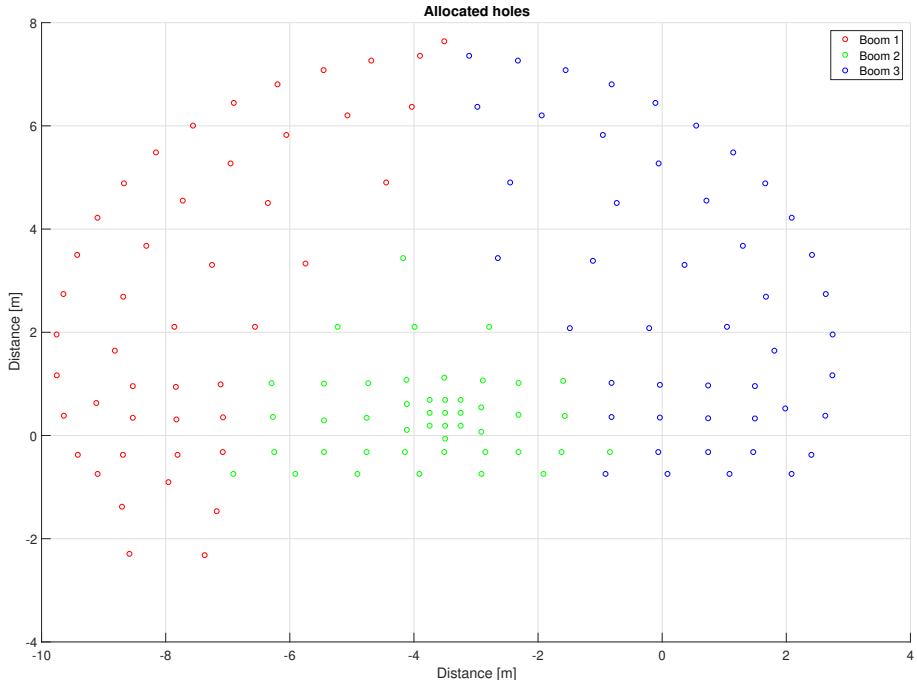

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-8
-6
-4
-2
0
2
4
6
8
10
12

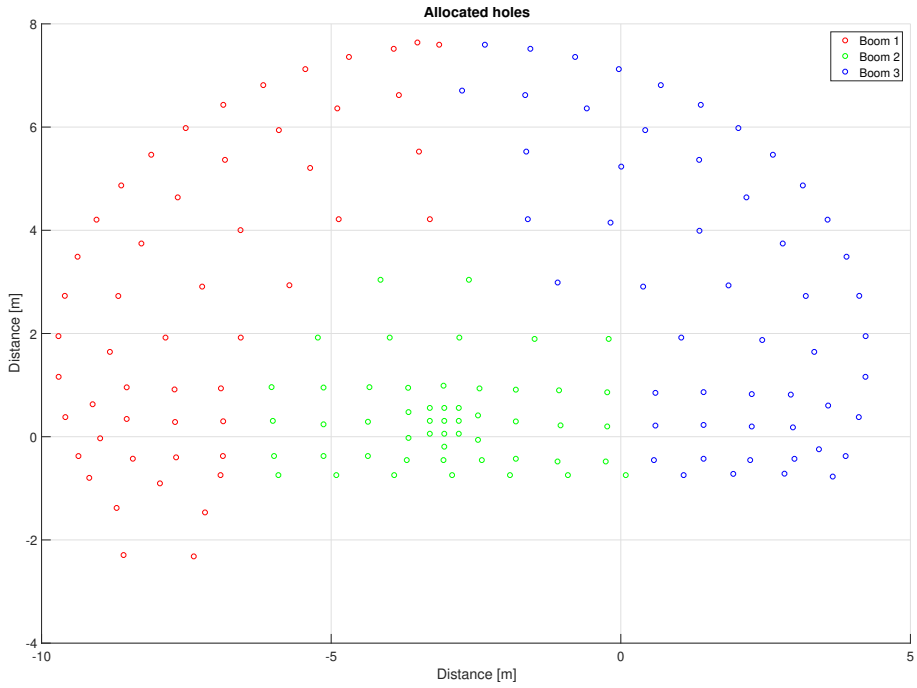

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-12
-10
-8
-6
-4
-2
0
2
4
6
8

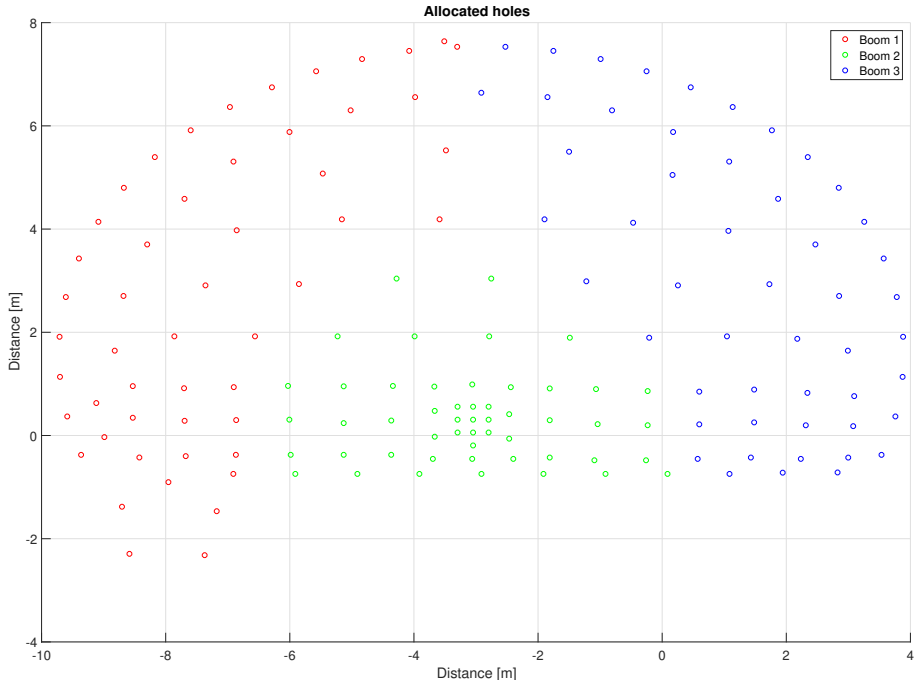

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4
6

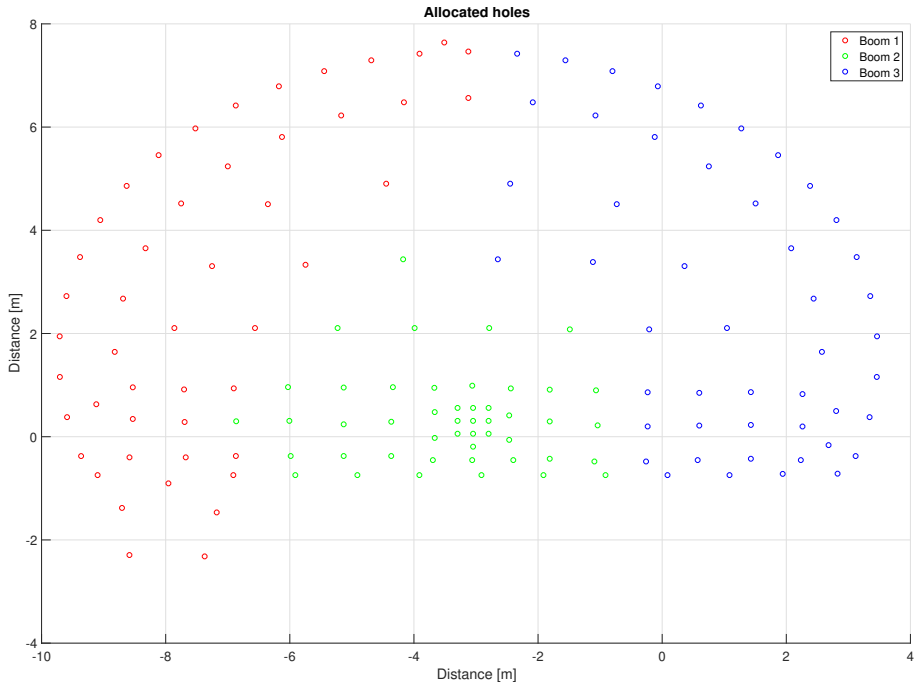

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-6
-4
-2
0
2
4
6
8
10


Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10

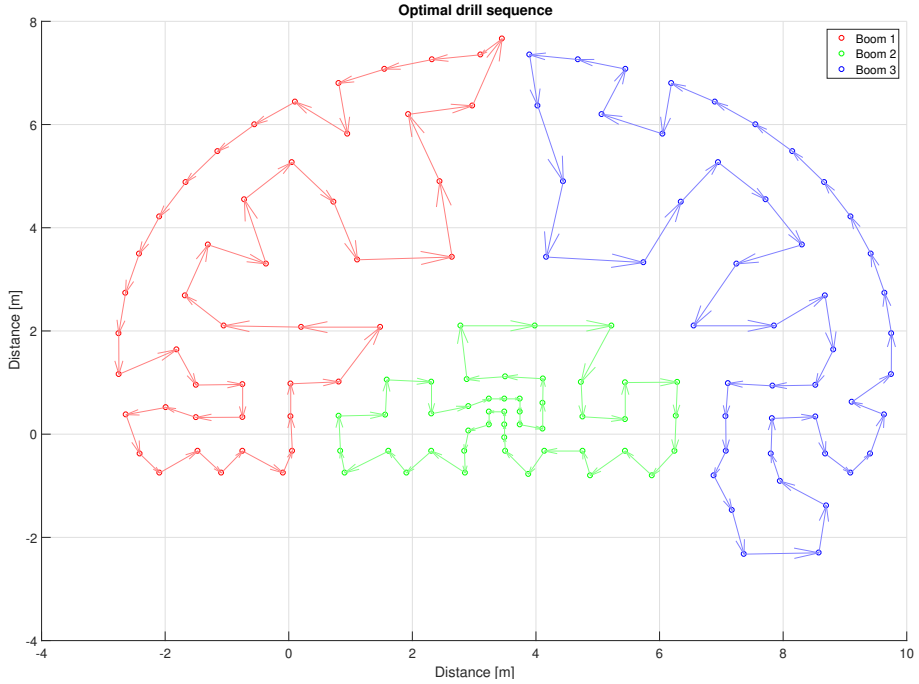

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4

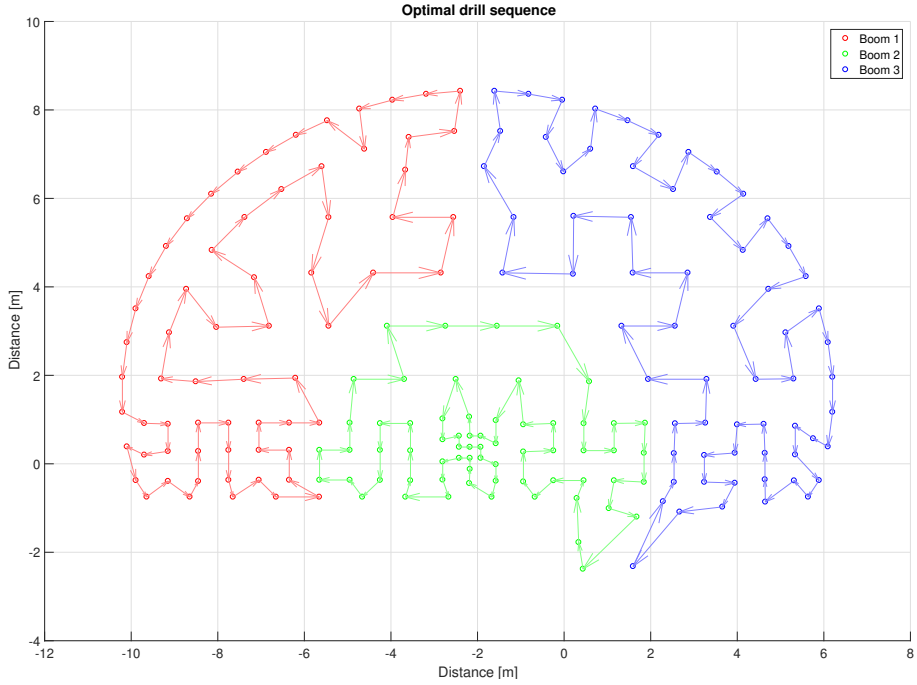

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-5
0
5

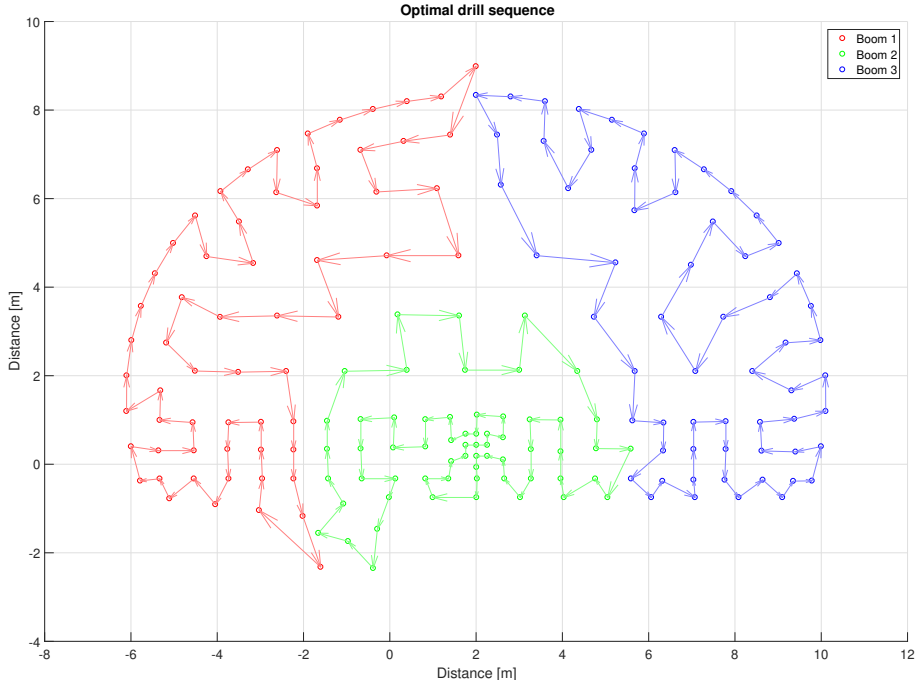

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4

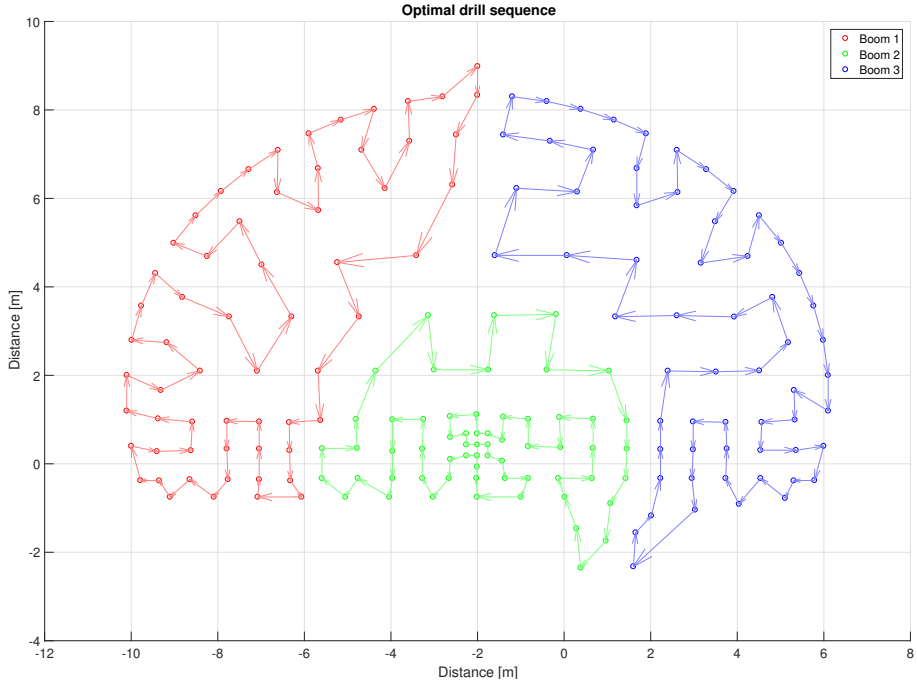

Allocated holes
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4

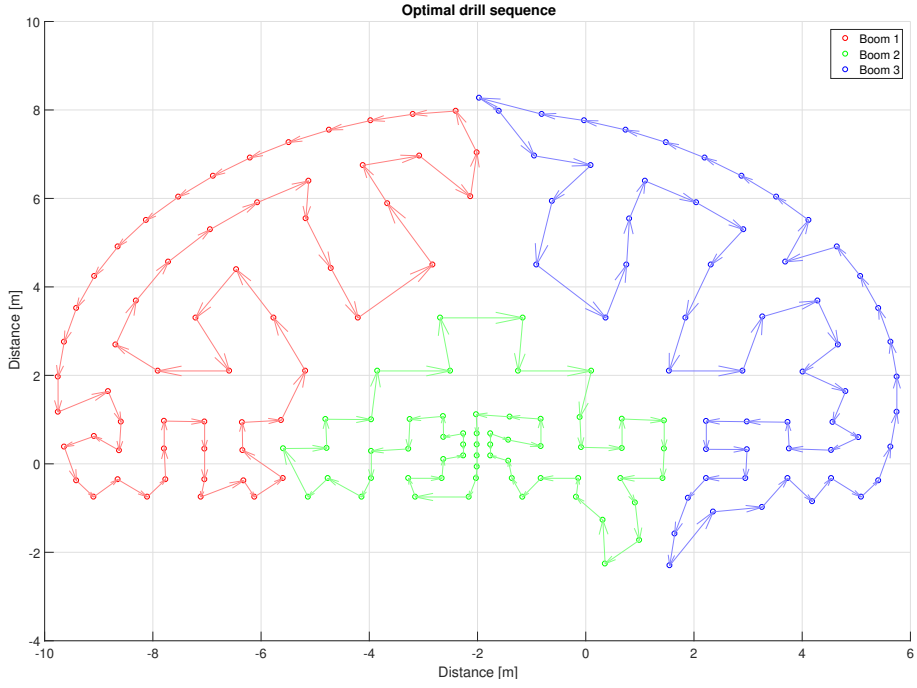
Appendix B: Additional optimal sequences

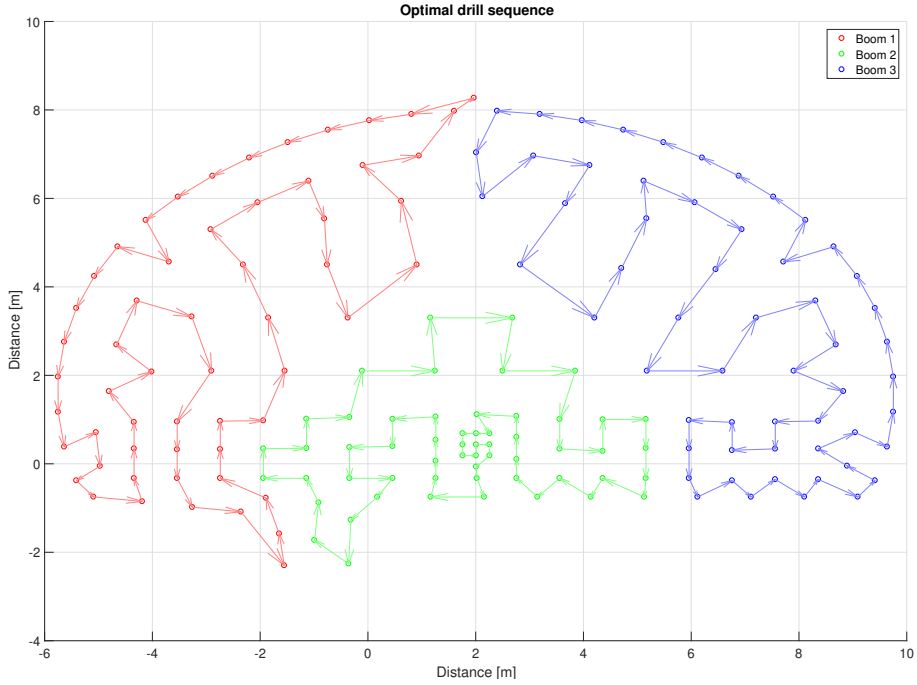


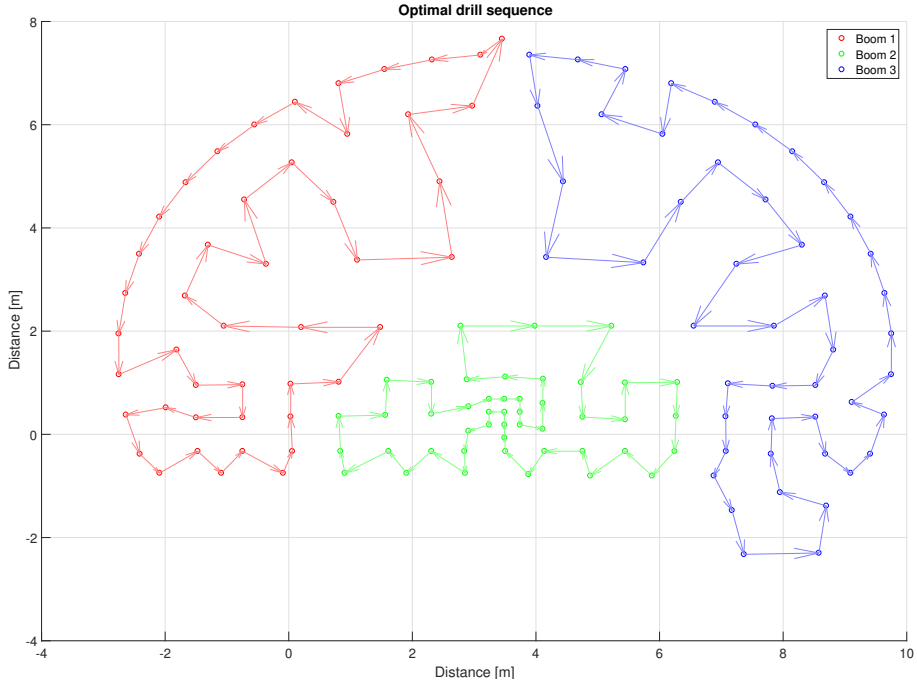

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]

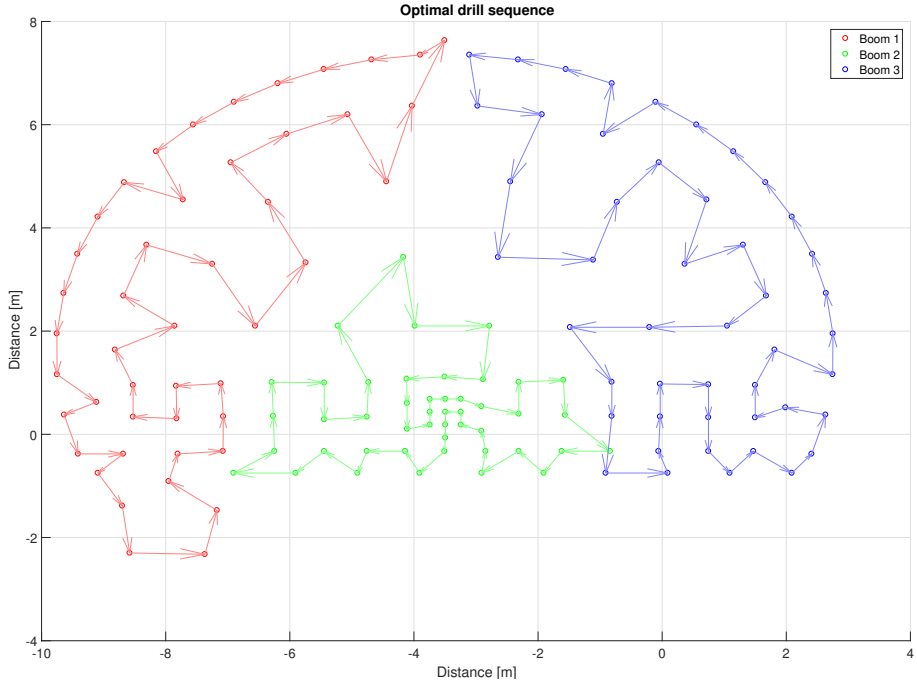

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]

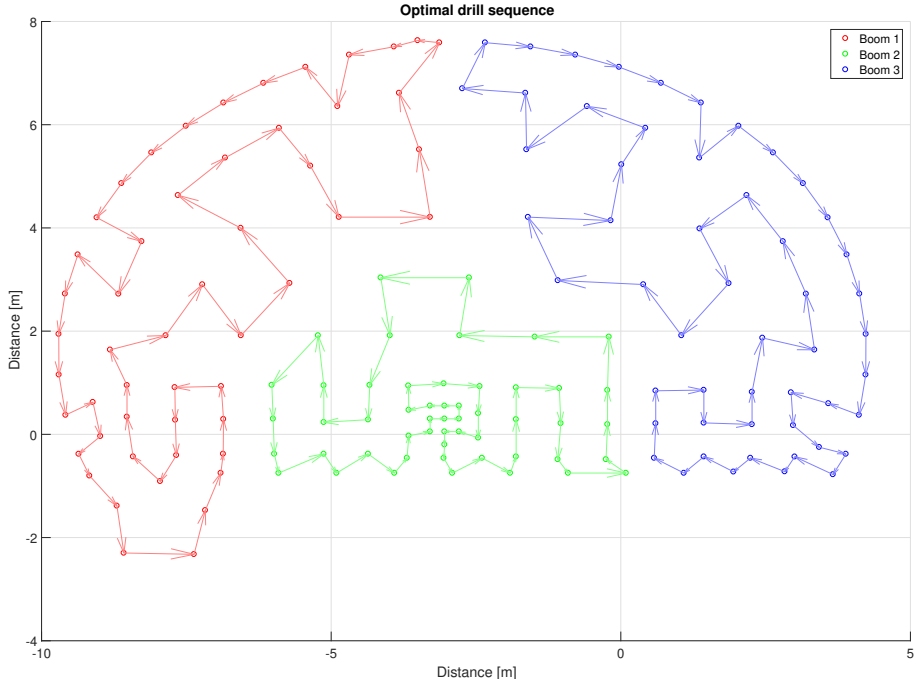

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]

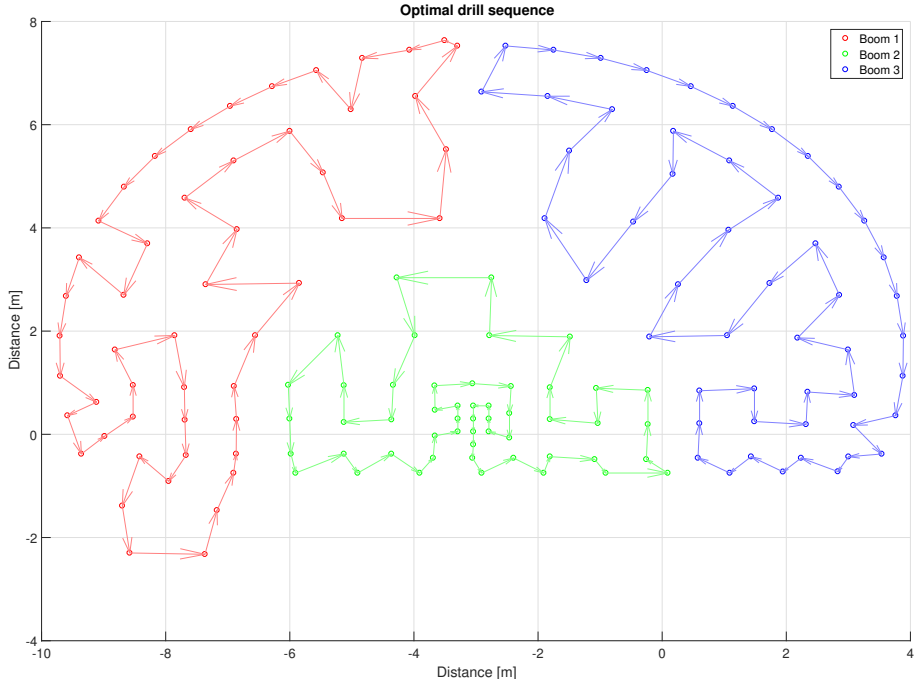

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]

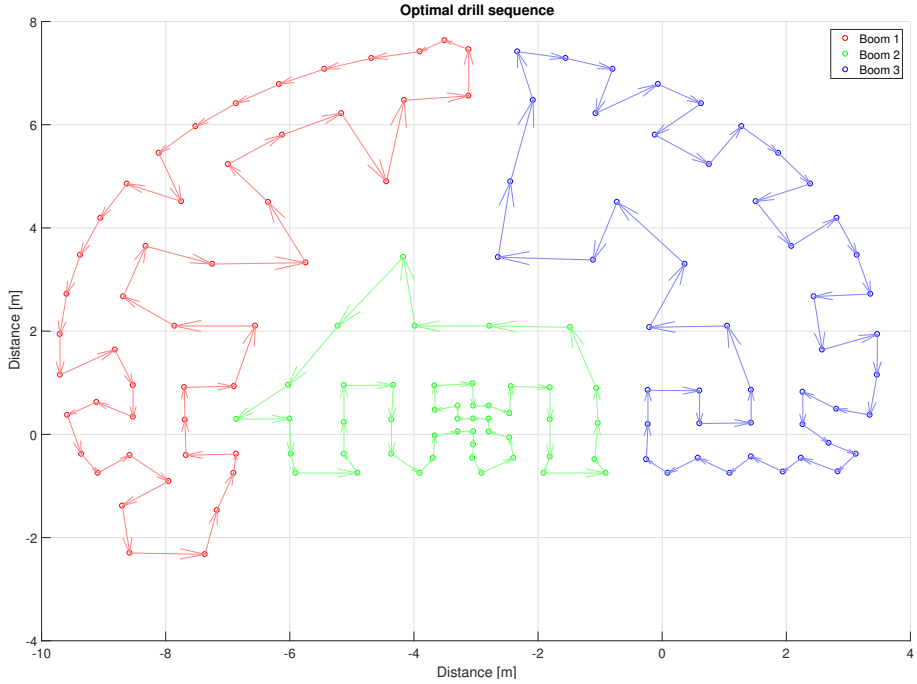

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]


Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]

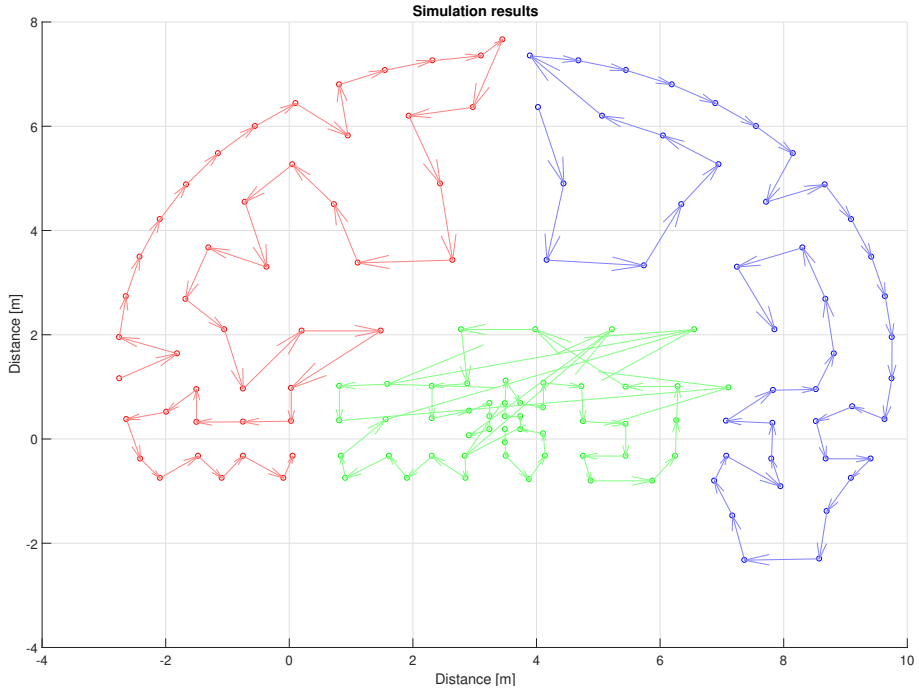

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4

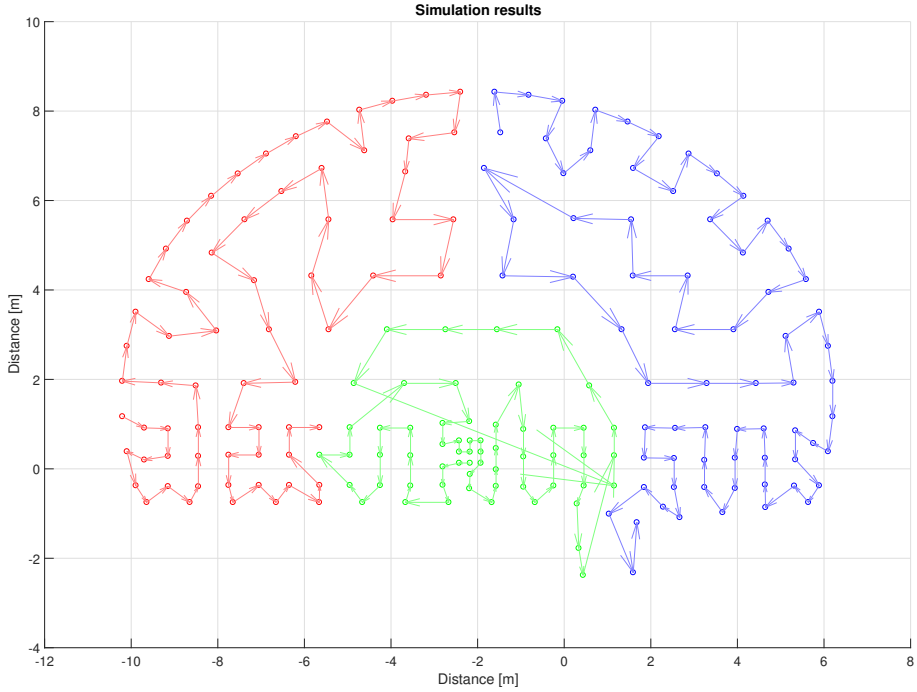

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-5
0
5

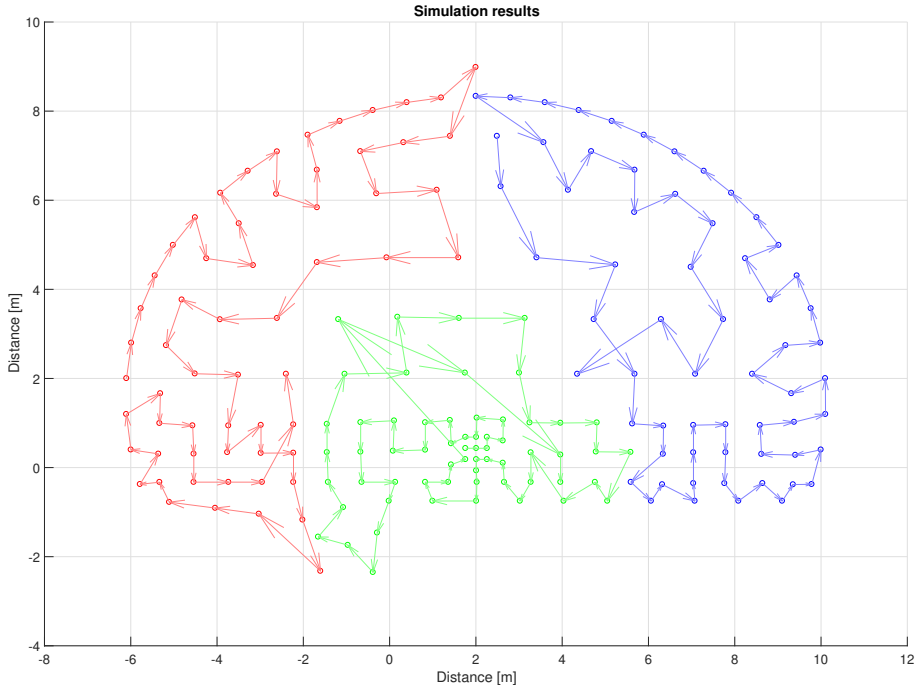

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4

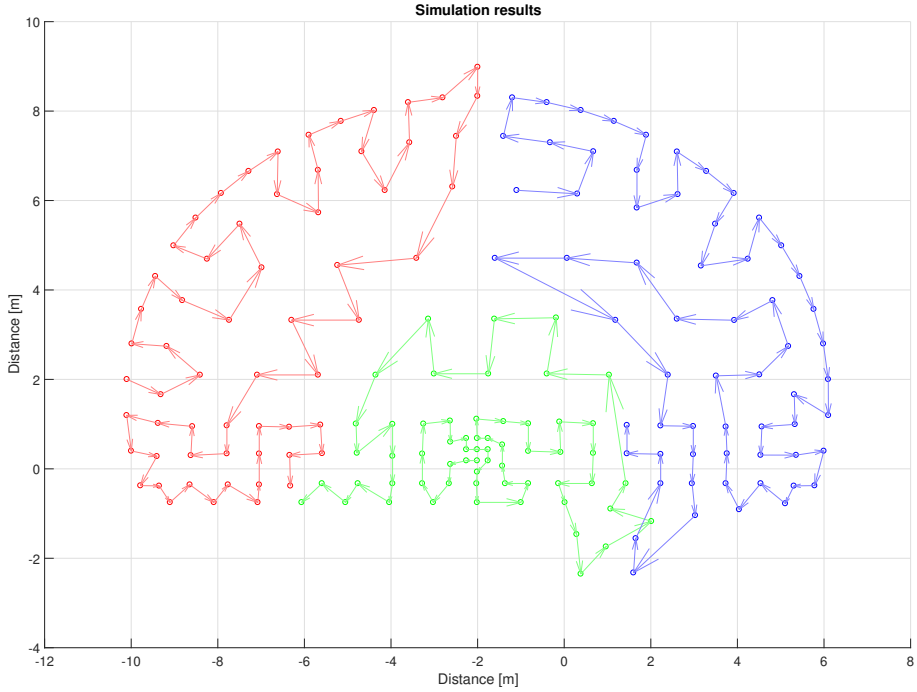

Optimal drill sequence
Boom 1
Boom 2
Boom 3
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4

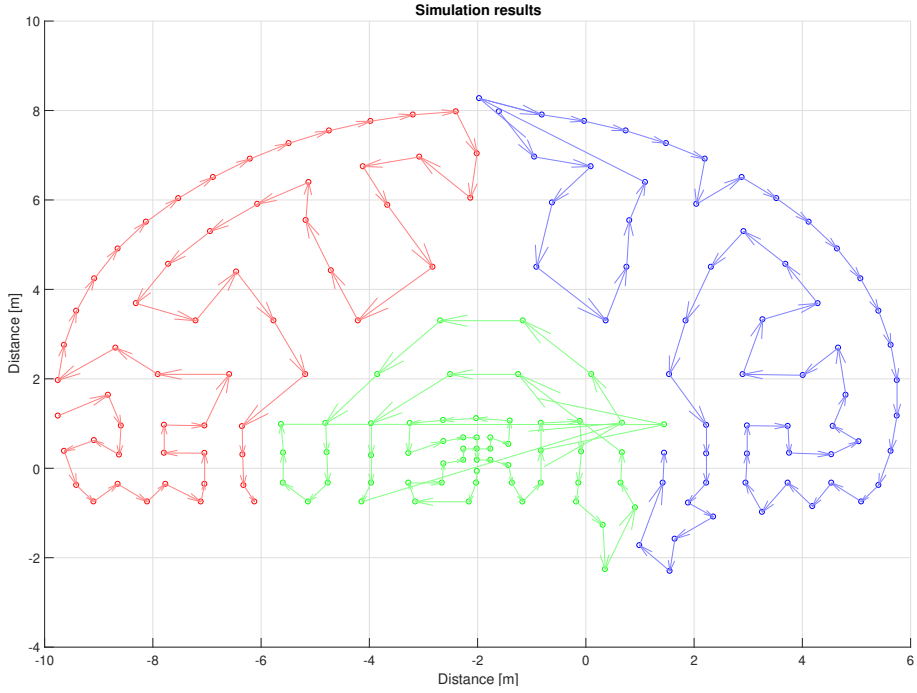
Appendix C: Additional simulation results

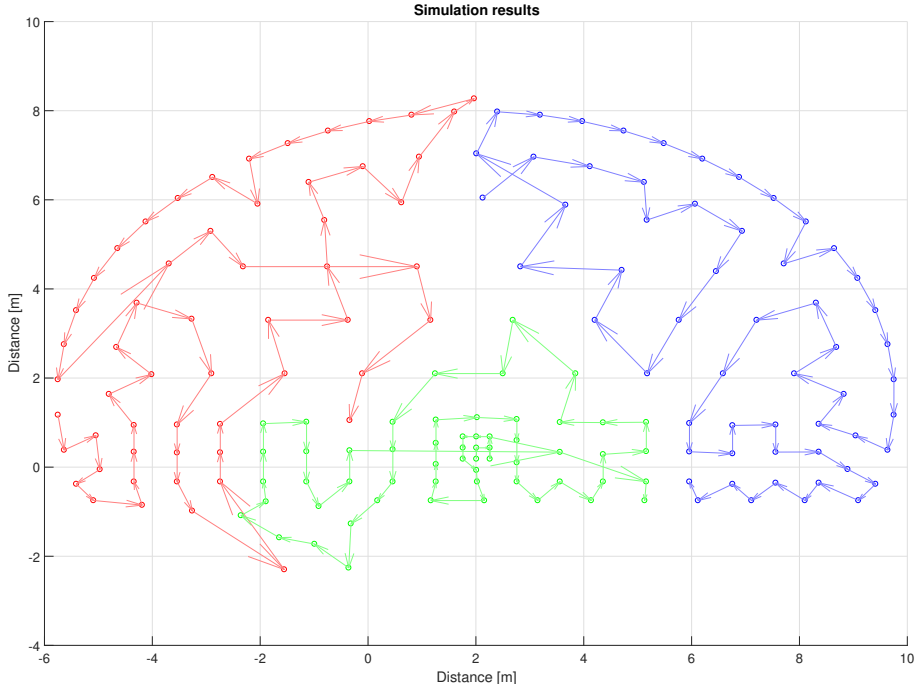


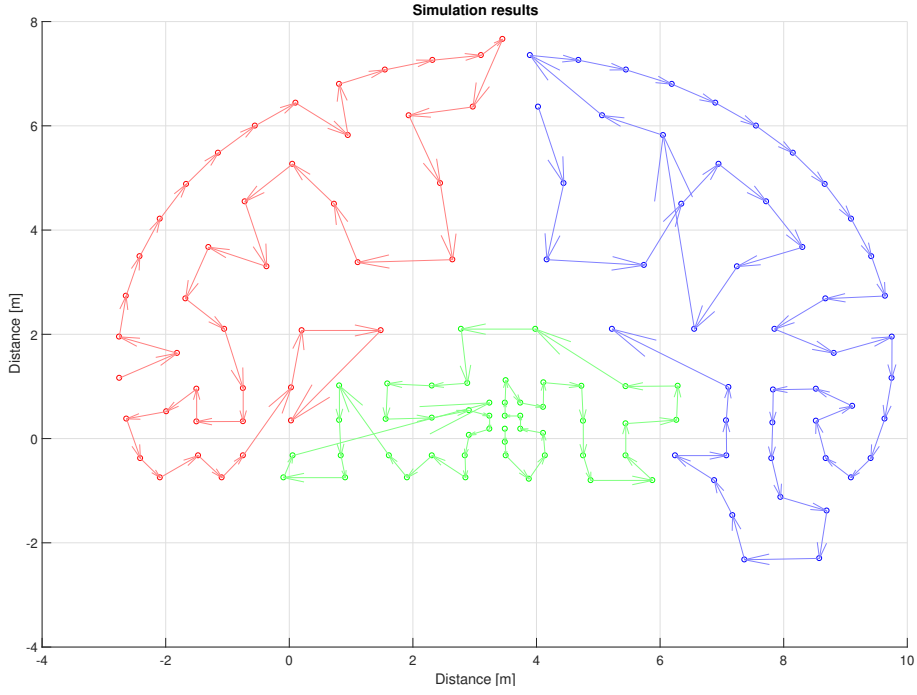

Simulation results
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-12
-10
-8
-6
-4
-2
0
2
4
6
8

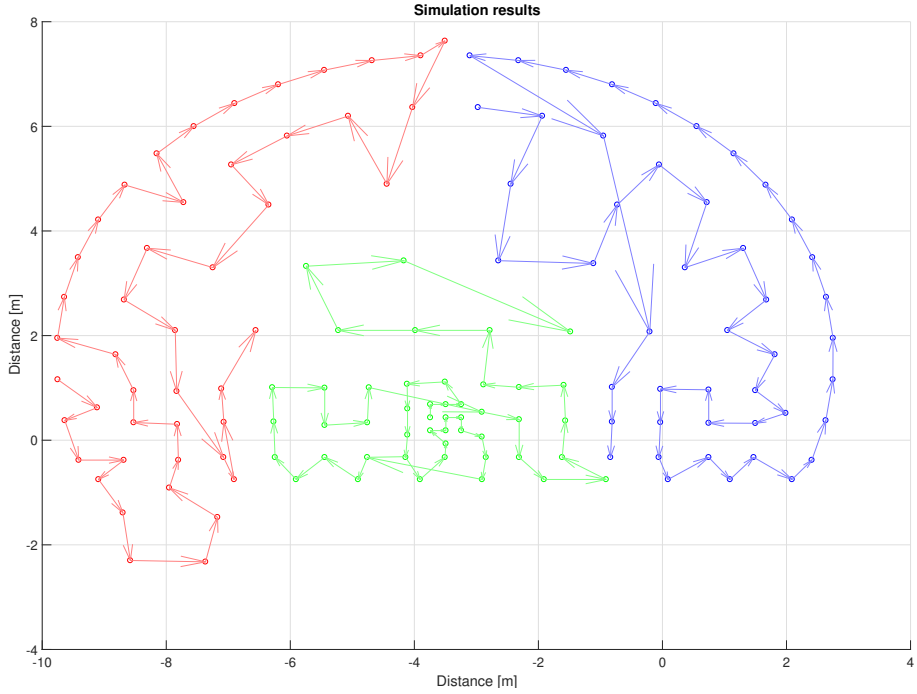

Simulation results
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-8
-6
-4
-2
0
2
4
6
8
10
12

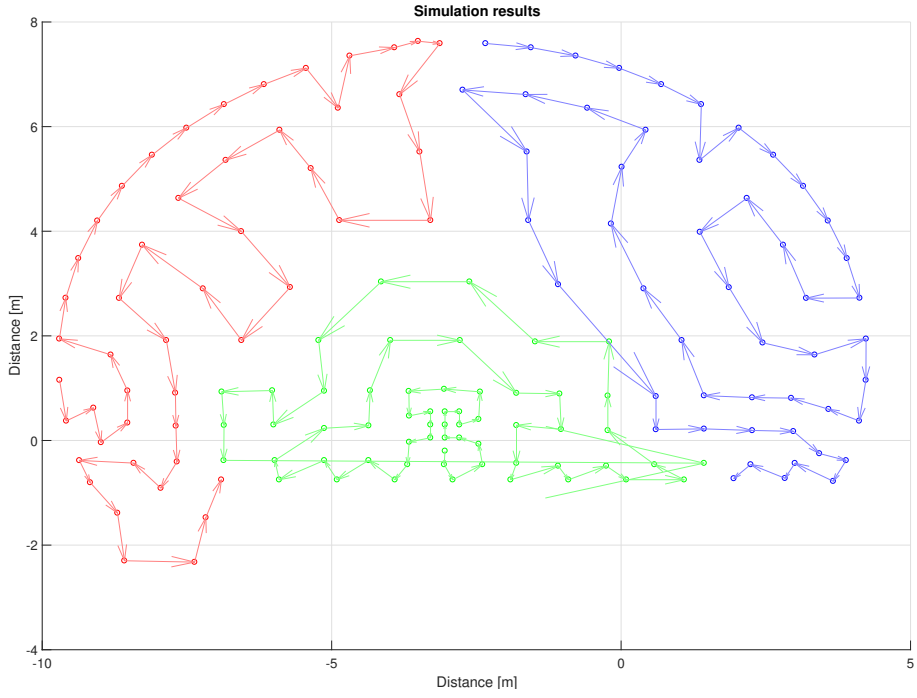

Simulation results
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-12
-10
-8
-6
-4
-2
0
2
4
6
8

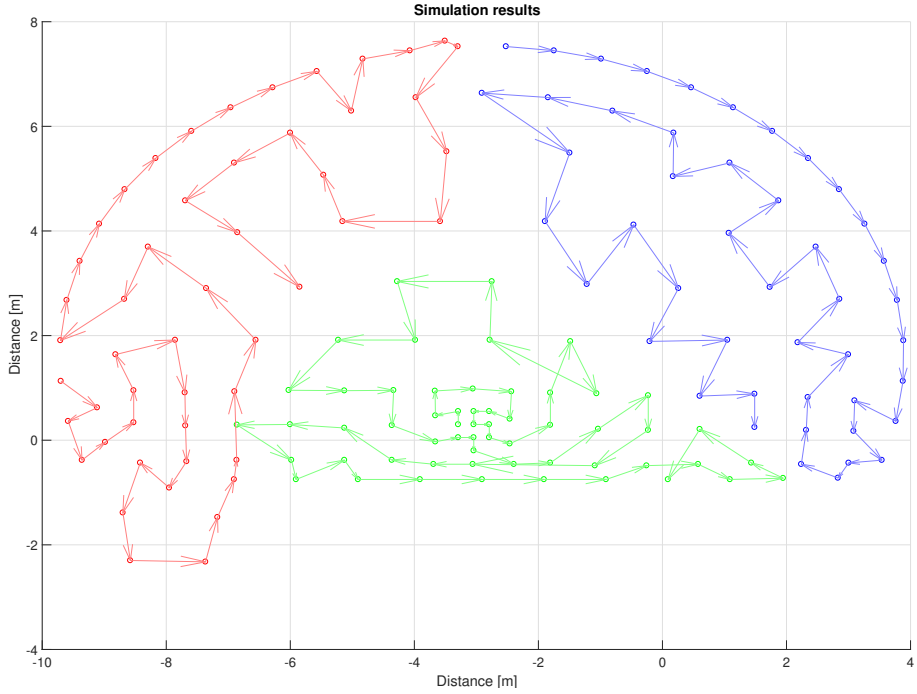

Simulation results
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4
6

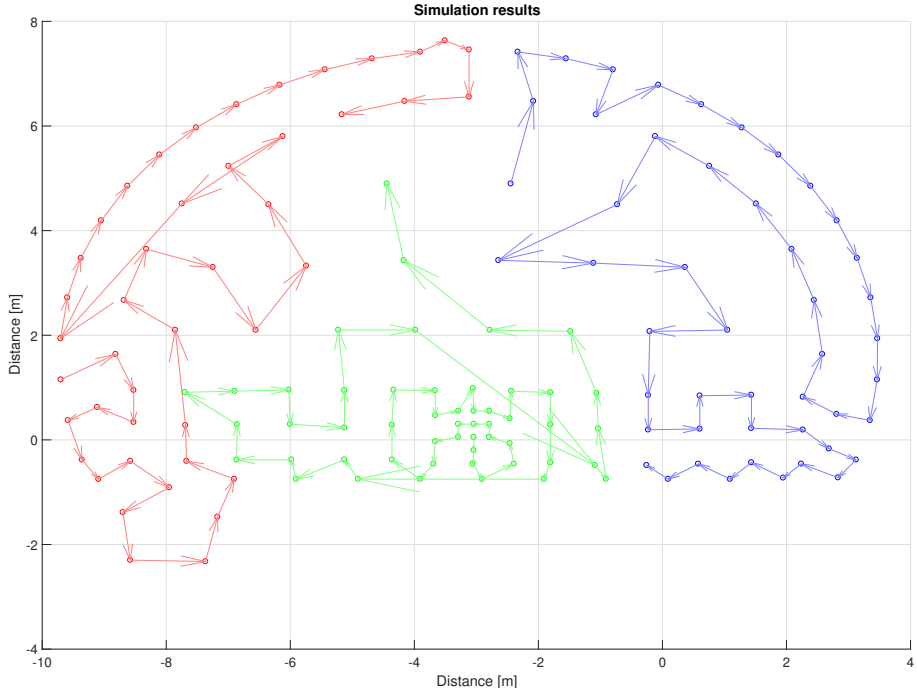

Simulation results
Distance [m]
Distance [m]
10
8
6
4
2
0
-2
-4
-6
-4
-2
0
2
4
6
8
10


Simulation results
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10


Simulation results
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4


Simulation results
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-5
0
5


Simulation results
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4


Simulation results
Distance [m]
Distance [m]
8
6
4
2
0
-2
-4
-10
-8
-6
-4
-2
0
2
4