FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/specialization: MSc. Petroleum Engineering/Drilling and Wells	The spring semester, 2023 Open / ~~Restricted~~
Author: Muhammad Suleman	
Supervisor at UiS: Prof. Dan Sui **External supervisor(s):** Juan Camilo (ProWellPlan AS)	
Thesis title: Autonomous Well Trajectory Design considering the Limitation of Torque and Drag Forces	
Credits (ECTS): 30	
Keywords: Trajectory design, Automation, well planning, Bezier curve, torque and drag limitations, optimization.	Pages: 80 Stavanger, 15 June 2023

Muhammad Suleman

Autonomous Trajectory Design Considering the Limitation of Torque and Drag Forces

Master Thesis Project for the degree of
MSc in Petroleum Engineering

Stavanger, June 2023

University of Stavanger
Faculty of Science and Technology
Department of Energy and Petroleum Engineering

Abstract

In the oil and gas industry, designing well trajectories is an important part of drilling operations that affect well construction, completion, and production. But the current trajectory planning process works in isolation and does not take many engineering constraints into account, which leads to inefficiency, manual iterations, and less-than-ideal results. This study aims to solve the problem by making an automated system for designing 3D trajectories that uses engineering calculations and focuses on torque & drag analysis. The objectives of this research include the development of algorithms to automate and optimize trajectory design, the integration of torque and drag calculations to avoid drill string damage through buckling or over torque, and the evaluation of the system's performance through case studies. The research also explores the kick-off point optimization and trajectory optimization between target points to enhance well placement and planning efficiency. The significance of this research lies in its potential to revolutionize well planning processes and minimize the cost associated with planning complex wells. It will help the industry by making trajectory planning easier, saving time and money, and minimizing the risks of drilling operations.

Acknowledgments

I would like to begin by expressing gratitude to Allah, the Most Merciful, for granting me the strength, guidance, and blessings that have enabled me to complete my research.

I also wish to extend my heartfelt thanks to my parents and family who have been my constant source of love, support, and encouragement throughout my academic journey. Their unwavering faith in my abilities has been a driving force behind my success, and I am eternally grateful for their love and guidance.

I would like to thank Professor Dan Sui, my thesis supervisor, for her invaluable guidance, support, and expertise throughout the entire process. Her insightful feedback, constructive criticism, and encouragement have been instrumental in shaping my research and academic growth.

I am also grateful to the ProWellPlan AS team, whose support made this research possible. Their contribution and partnership have been critical in the successful completion of this thesis.

Lastly, I would like to extend my gratitude to the University of Stavanger, for providing me with an exceptional learning environment and for facilitating my academic growth. The knowledge and skills I have gained during my studies will undoubtedly have a profound impact on my future endeavors

List of Abbreviations

AI	Artificial Intelligence
DL	Dog Leg
DLS	Dog-leg severity
E	Easting
ERD	Extended Reach Drilling
ERW	Extended Reach Wells
GA	Genetic Algorithm
KOP	Kick-off point
MD	Measured Depth
ML	Machine Learning
MOP	Multi-objective Optimization Problem
N	Northing
PSO	Particle Swarm Optimization
PWP	ProWellPlan
ROP	Rate of Penetration
RPM	Revolutions per minute
RSS	Rotary Steerable system
T&D	Torque and Drag
TOB	Torque on Bit
TVD	True Vertical Depth
UTM	Universal Transverse Mercator
V	Vertical Depth
WOB	Weight on Bit

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the oil and gas industry, drilling operations play a pivotal role in the extraction of hydrocarbon resources from the subsurface. An essential and primary aspect of planning drilling operations is trajectory design, which involves determining the optimal path and placement of the wellbore. The trajectory design has a significant impact on the success and efficiency of drilling projects. It influences well construction, completion, and production phases, directly affecting factors such as wellbore stability, cost-effectiveness, and resource recovery.

Trajectory design is a complex task that requires careful consideration of various geological, engineering, and operational factors. It particularly plays a crucial role in the success of horizontal drilling operations, especially in extended-reach wells where excessive torque and drag can have critical limitations during drilling [1].

The current trajectory design process has extensive room for improvement using automation, optimization, and machine learning algorithms. Advancing trajectory design practices using these methods are of paramount importance to industry.

1.2 Problem Statement

The design of well trajectories plays a crucial role in well planning, impacting the overall well design and drilling program. However, traditional approaches to trajectory design have been fragmented and isolated, leading to inefficiencies and time-consuming processes. Well trajectories were typically designed separately by the directional service companies, well planners, and geologists using different applications and tools. Surface and subsurface coordinates, along with operator-

provided constraints, formed the basis of trajectory design. The lack of integration and validation of engineering factors, such as geomechanics, torque and drag, and bottom hole assembly (BHA) tendency, further complicated the process. As a result, trajectory design has involved multiple iterations, discussions, and data sharing to address constraints and validation issues identified by other workflows. This approach of disconnected workflows hindered the efficiency and effectiveness of trajectory design, leading to suboptimal outcomes and increased time investment for engineers [2].

Therefore, there is a pressing need for an innovative solution that automates the trajectory design process, incorporates engineering calculations, and streamlines the optimization of drilling trajectories while considering critical constraints such as torque and drag forces. By addressing these challenges, the industry can enhance operational outcomes, minimize risks, and improve overall drilling performance.

1.3 Research Objectives and Scope

The primary objective of this research is to develop an automated 3D trajectory design system that considers engineering calculations, specifically focusing on torque and drag analysis. A new autonomous approach will be proposed where torque and drag calculations are incorporated into the process of trajectory design. The system aims to streamline the trajectory design process and enhance operational outcomes during drilling operations.

The research objectives include:

1. Developing algorithms and methodologies to automate trajectory design, incorporating subsurface target points and engineering constraints.
2. Integrating torque and drag calculations into the trajectory design process to ensure the drill string's ability to withstand forces exerted during drilling.
3. Evaluating the performance and effectiveness of the automated trajectory design system through case studies.
4. Assessing the potential for optimization of kick-off point (KOP) and target points, to improve wellbore placement and drilling efficiency.

The scope of this research focuses on the automation of trajectory design and the integration of torque and drag calculations. While the other engineering calculations, such as hydraulics and casing design, can be integrated into the system in the future, they are beyond the current scope of this research. The research will contribute to advancing trajectory design practices and laying the foundation for further enhancements in well-planning processes.

1.4 Research Significance

The research on automated 3D trajectory design considering engineering calculations, particularly torque and drag analysis, holds significant importance in the field of drilling operations. By developing an automated system that integrates engineering calculations into trajectory design, this research has the potential to revolutionize well planning processes and improve drilling efficiency.

The significance of this study lies in several key aspects. Firstly, the automated trajectory design system can reduce the time and resources required for trajectory planning by eliminating the manual optimization process and enabling engineers to generate well plans more efficiently. Secondly, by incorporating torque and drag analysis, the system ensures that the trajectory design accounts for critical operational constraints, thereby reducing the risk of drill string failures and improving overall drilling performance. Additionally, the research outcomes will provide valuable insights into the benefits of automated trajectory design and serve as a foundation for further enhancements and integration of additional engineering calculations in future studies.

The outcomes of this research can benefit the oil and gas industry by enhancing well-planning processes, minimizing operational risks, and ultimately optimizing drilling operations for improved productivity and cost-effectiveness.

1.5 Thesis Structure

The literature review in Chapter 2 explores the existing knowledge and research related to trajectory design in drilling operations. It provides an overview of traditional trajectory design approaches. The chapter also reviews the importance of engineering calculations in trajectory design, focusing on torque and drag analysis. By examining

relevant literature and research, this chapter establishes the foundation for the development of an automated trajectory design system.

In Chapter 3, the methodology employed for developing the automated 3D trajectory design system is described in detail. It outlines the steps involved in algorithm development and the integration of torque and design model. It provides insights into the workflow, showcasing the seamless integration of torque and drag model, into the trajectory design process. The chapter also discusses the methods utilized to ensure accurate trajectory design. By providing a comprehensive explanation of the methodology, this chapter establishes the basis for the implementation of the automated system. It also describes the code and tools used for developing and implementing the system.

Chapter 4 of this thesis presents the implementation of the autonomous trajectory design system through two case studies. The first case study focuses on extended reach well design, showcasing the system's ability to generate optimized trajectories that reach distant target points while adhering to specified constraints. The second case study addresses the optimization of the kick-off point. The results from both case studies demonstrate the system's effectiveness in generating optimized trajectories to achieve desired objectives. The findings and results of the automated system will be discussed here.

Chapter 5 summarizes the practical implementation of the automated 3D trajectory design system in various areas of well planning and drilling operations. It highlights the system's ability to automate well planning, ensure trajectory correctness during drilling operations using RSS, and optimize key trajectory points such as the kick-off point (KOP), surface points, and subsurface points. The chapter discusses the effectiveness and limitations of the system and offers recommendations for future enhancements and integration of additional engineering calculations. By presenting a clear conclusion and future research directions, this chapter concludes the thesis.

Chapter 2

Literature Review

This chapter provides an in-depth analysis of existing research and studies relevant to trajectory design. It explores the current state of the field, highlighting key concepts and methodologies that contribute to the understanding of automated trajectory design systems. Methods and approaches from the reviewed literature are critically analyzed and synthesized to establish a foundation for the research presented in the thesis.

2.1 Basics of Trajectory Design

In well-trajectory design, understanding the fundamental concepts and terms is crucial before we go into further details. A well path is composed of segments that connect fixed points along the trajectory, starting from the well-head and progressing toward the target. Each segment is associated with coordinates in a 3D dimensional system, including north, east, and true vertical depth (TVD), allowing for precise positioning of the wellbore.

The well path is planned segment by segment, with each step causing changes in the north, east, and TVD coordinates. By carefully coordinating and planning the well path, a detailed plan of the well is formed. The subsurface team often provides target coordinates, which serve as reference points for drilling engineers to plan the well from the well-head to reach the designated targets [3].

The well trajectory design considers various factors such as inclination, azimuth, and dogleg severity which determine the overall shape and position of the wellbore.

2.1.1 Inclination

Inclination refers to the angle at which the wellbore deviates from the vertical axis. It plays a crucial role in determining the trajectory of the well path. It is measured from

zero degree which is vertical, up to 90 degrees representing a horizontal shape. Figure 2.1 shows the inclination angle of a wellbore as **I**.

2.1.2 Azimuth

Azimuth is another critical parameter that helps define the direction of the wellbore. It refers to the horizontal angle measured in degrees clockwise from a reference direction, typically North. The borehole trajectory can be visualized within a circular representation of 360 degrees. When observing the well from above, the borehole can traverse in any direction around this circle, with the north direction corresponding to 0 or 360 degrees, the east direction to 90 degrees, and so on. In Figure 2.1, the angle represented by the Direction angle is the azimuth of the wellbore.

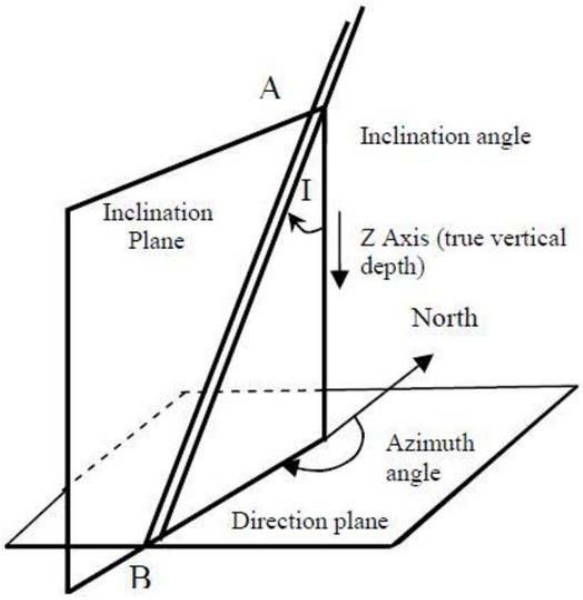

Figure 2.1 Inclination and Azimuth of Wellbore [3]

2.1.3 Dog Leg and Dog Leg Severity

In well-trajectory design, the concept of dog leg and dogleg severity is important to understand. The term "dog leg" (DL) refers to the change in wellbore direction. On the other hand, the "dog leg severity" (DLS) quantifies the magnitude or severity of the change in wellbore direction over a specific interval, typically measured in degrees per 100 feet or 30 meters.

The DL and DLS concepts can be explained using Figure 2.1 where **I** is inclination. If we consider two inclinations representing $\mathbf{I}_1$ and $\mathbf{I}_2$, then DL is given in (1)

$$DL = I_2 - I_1 \quad (1)$$

Dog leg severity is a measure of the rate of change of direction within a specific interval of the wellbore. It is calculated by dividing the change in direction (in degrees) by the length of the interval (in feet or meters). The resulting value represents the average degree change per unit of length and indicates the smoothness or abruptness of the wellbore curvature. DLS in deg/30 meters is given in (2) where DL is defined in (1) and CL is Curve Length between two survey points for which DLS is calculated.

$$DLS = \frac{DL}{CL} \times 30 \quad (2)$$

Another equation for DLS developed by Lubinski is given in (3) where **L** represents length, **I** represent Inclination and **A** represents Azimuth.

$$DLS = \frac{2}{L_2 - L_1} * sin^{-1} \sqrt{sin^2\left(\frac{I_2 - I_1}{2}\right) + sin^2\left(\frac{A_2 - A_1}{2}\right) * sin(I_1) sin(I_2)} \quad (3)$$

Higher dog leg severities indicate more pronounced changes in wellbore direction within a given interval, while lower values indicate smoother transitions. Managing dog leg severity is crucial to ensure the integrity and stability of the wellbore, optimize drilling efficiency, and minimize operational challenges such as increased torque and drag, casing wear, and equipment limitations.

2.2 Overview of Trajectory Design Methods

Wellbore trajectory design involves determining the optimal path and geometry of the wellbore to reach the desired target zones efficiently while considering various technical constraints. Over the years, numerous techniques and methodologies have been developed to address the challenges associated with wellbore trajectory design.

Specifically, this section focuses on three prominent methods: the Minimum Curvature method, the radius of curvature method, and the relatively newer method of the Bezier Splines method. These methods will be briefly explored and summarized.

2.2.1 Minimum Curvature Method

One commonly used technique or in other words, the industry standard is the minimum curvature method for survey calculation in directional drilling [4]. It approximates the wellbore path as a series of straight lines and circular arcs [5]. This method offers simplicity in calculation and has been widely adopted in practice.

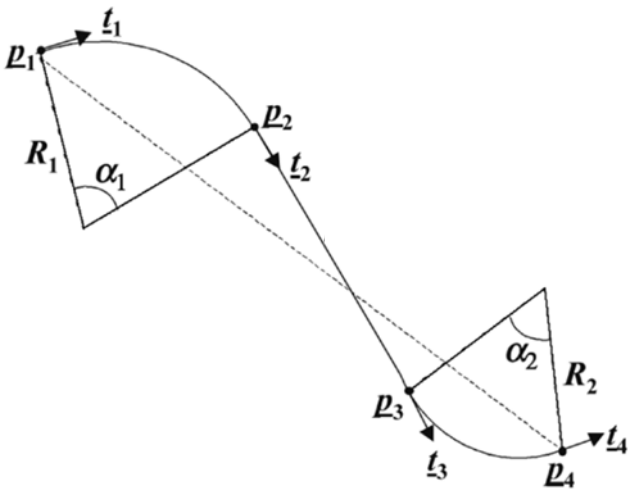

Figure 2.2. Trajectory Through 4 Targets Using Minimum Curvature Method [5]

Figure 2.2 shows this method through points $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$, and $\mathbf{p_4}$. **t** represents tangents, $\mathbf{R_1}$ and $\mathbf{R_2}$ are the radii of curvatures with the angle of curvatures represented as α_1 and α_2.

2.2.2 Radius of Curvature Method

In the radius of curvature method, the wellbore is assumed to be curved in either or both vertical and horizontal projections. Each segment of measured depth is defined by data obtained at both ends of the segment. It involves an approximation of the well path, where it is assumed that the path can be represented as a circular arc in both the horizontal and vertical planes [6].

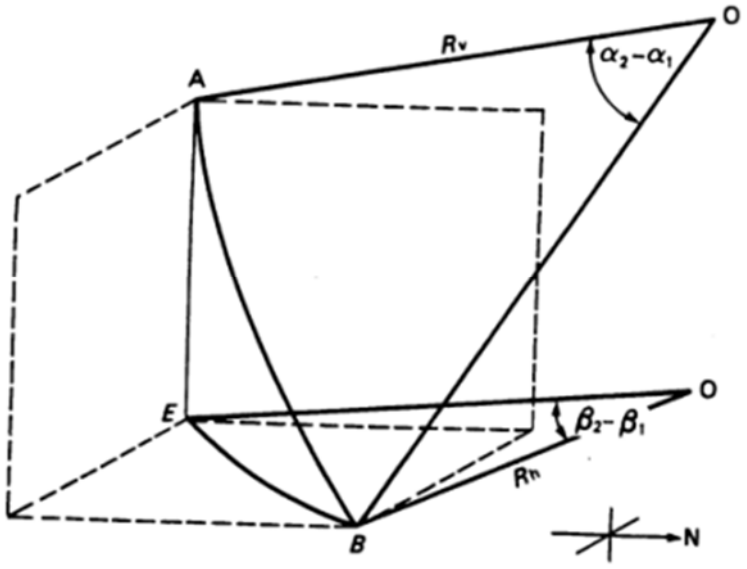

Figure 2.3 Radius of Curvature Method [6]

Figure 2.3 shows the circular arc between points A and B, having radii R_v and R_h in vertical and horizontal planes respectively. α and β represent the inclination and azimuth for a continuous arc connecting the two points A and B. In this approach, the well path can be described as a circular arc in the vertical plane, wrapping around a right cylinder.

2.2.3 Bezier Spline Method

Liu et al. in 1991 [7] discussed the use of spline curve methods for coordinate calculations in wellbore trajectories. It is a relatively new approach to complex three-dimensional trajectory design. These mathematical curves provide more flexibility and a superior approach in terms of coding compared to other methods. One of the remarkable advantages of this method is its ability to describe the trajectory using a single expression that encompasses all three space coordinates. This eliminates the need to work with separate coordinate functions, simplifying the representation of the trajectory and enhancing its efficiency [8].

The reasoning for the selection of this method over the minimum curvature method is given in Section **Error! Reference source not found.**2.2.4. The remaining details of the Bezier spline method and how it is implemented are discussed in Chapter 3.

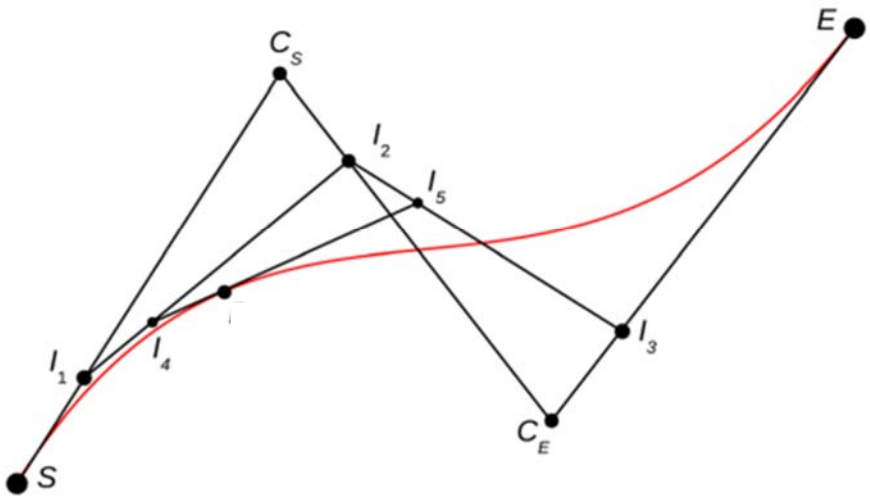

Figure 2.4 A Third-Order Bezier Curve [8]

In Figure 2.4, a single third-order Bezier curve is shown between point S and point E. C_S and C_E are control points for this red curve and I_1, I_2, I_3, and I_4 represent intermediate points.

2.2.4 Comparison of Trajectory Design Methods

Well trajectories have significant impacts on other design considerations such as drill string design, casing design, torque and drag estimation, and wellbore pressure calculations. Incorrect designs can lead to problems like wellbore instability, loss of circulation, and drill string failure, which are costly. Initially, vertical wells were drilled, but with the move towards drilling complex and designer wells, more complex calculations became necessary. Survey calculations were introduced to monitor the inclination and azimuth of wells as they bend and turn simultaneously.

The minimum curvature method assumes a circular arc in an inclined plane, which results in constant curvature between survey points. This constant curvature model of the minimum curvature method introduces discontinuity at the survey course intervals. This type of constant curvature model due to discontinuity may result in the omission of some contact forces which will result in the underprediction of the hook load or the stresses in the drill string components or casing or drill pipe wear calculations.

On the other hand, the radius of curvature method assumes the well path as a circular arc in both vertical and horizontal planes. It is more suitable for rotary mode drilling. But for recent drilling with RSS, when the wellbore twists and turns, this method results in discontinuity in the engineering calculations also.

In contrast, the Bézier method (spline method) provides a continuous curvature profile along the wellbore trajectory. By using spline curves, the well path can be smoothly interpolated between survey points, eliminating the discontinuities found in other methods [9].

2.3 Challenges in Trajectory Design Process

Well planning involves designing various aspects of well construction, including trajectory, wellbore geometry, fluids, casing, drill string, and cement, all of which are interconnected to optimize well performance. The trajectory design determines the path of the wellbore from the surface location to the target reservoir and influences the overall well depth. It serves as a crucial reference for other drilling programs components, such as casing and drill string loads, drill bit and steering tool design, and well production outcomes. Thus, designing an optimal trajectory is vital for the overall success of the well.

Traditionally, trajectory design has been conducted in isolation by geologists or directional drilling vendors, aiming to connect the surface location with the target reservoir. Certain constraints, such as kickoff point depth, maximum allowable dogleg severity, final inclination, and azimuth approaching the reservoir, are considered during the design process. However, the design process primarily relies on manual methods, involving the selection and connection of hold and curve sections from the surface to the total depth. The outcome of this process heavily relies on the competence and experience of the planner, which may not always result in the most optimized design.

Once the engineers complete the trajectory design, it is shared with other planning team members via email or manual data transfer for further analysis and validation. If

engineering validations, such as anticipated total pressure, BHA tendency, or maximum torque, indicate discrepancies, it may necessitate revising the trajectory design. This feedback loop requires communication with the trajectory planner to make further adjustments, such as shortening the total depth, minimizing dogleg severity at shallower depths, or reducing the maximum inclination in the tangent section. Unfortunately, this iterative process often involves utilizing multiple software modules or applications, leading to time-consuming efforts [2].

2.3.1 Use of Landmark Software

Halliburton Landmark's Compass software is a widely used tool in the field of trajectory design for drilling operations. Compass provides capabilities that enable engineers to plan wellbore trajectories using the minimum curvature method [10].

It is important to note that the trajectory design is not directly connected to the engineering calculations like torque and drag in tools like Compass. For such calculations and analysis, another tool named Well Plan is used. These two aspects of well planning and drilling operations typically work in isolation, each focusing on its specific set of considerations.

The trajectory design phase in Compass primarily focuses on generating a well path that connects the specified survey points measured in measured depth, inclination, and azimuth. This process involves mathematical calculations based on the minimum curvature method to obtain the remaining parameters related to the trajectory. The objective of this method is to create a continuous trajectory through survey points.

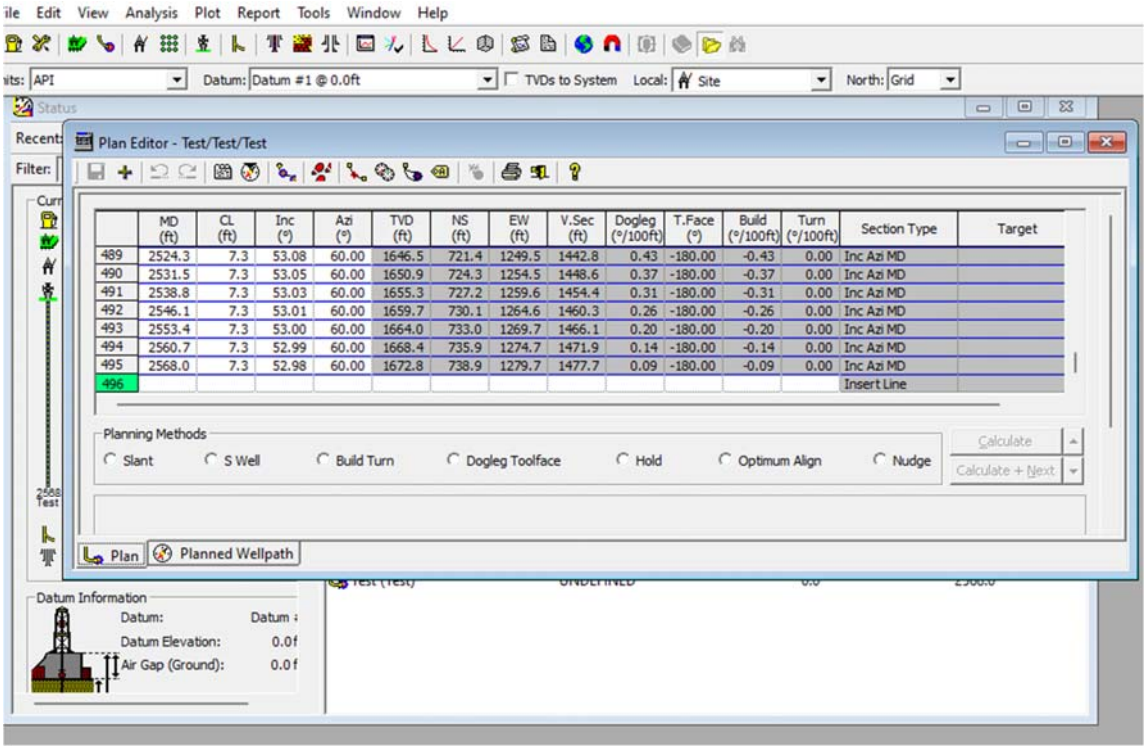

Figure 2.5 Screenshot of Compass Software by Landmark

To address these challenges, there is a growing need for automated and integrated trajectory design approaches that streamline the iterative process and enable real-time engineering validations. Such solutions would significantly reduce manual effort, enhance design optimization, and improve overall efficiency in well-planning workflows. By leveraging advanced computational algorithms and integrating engineering calculations, these automated systems can revolutionize trajectory design and enable more accurate and efficient well construction.

2.3.2 Effect of Trajectory on Torque and Drag Forces

The trajectory of a wellbore has a significant effect on the torque and drag forces encountered during drilling operations. Torque refers to the rotational force required to turn the drill string, while drag refers to the axial resistance encountered as the drill string moves through the wellbore.

For example, in ERWs, managing torque and drag is a significant challenge. As the lateral length increases, torque and drag can become substantial, limiting the achievable reach, and affecting drilling efficiency. Excessive torque can lead to

equipment failures and drilling issues, while high drag can hinder the drilling progress. Mitigating torque and drag requires advanced modeling techniques, proper wellbore positioning, and the use of specialized drilling tools and technologies designed to minimize these forces.

Here's how the trajectory impacts the torque and drag forces:

1. Wellbore Inclination:
   The inclination angle of the wellbore, which is the angle at which the wellbore deviates from vertical, plays a crucial role in torque and drag forces. As the inclination angle increases, the torque required to turn the drill string also increases. This is because the weight of the drill string component acts perpendicular to the wellbore inclination, creating a larger moment arm and thus increasing the torque.

2. Dogleg Severity:
   Doglegs occur when the wellbore changes direction abruptly, resulting in a significant change in inclination and azimuth. Higher dogleg severity, characterized by sharper changes in direction, can significantly impact torque and drag forces. In such sections, the drill string experiences additional bending stresses, which contribute to increased torque and drag. These stresses can lead to higher frictional forces between the drill string and the wellbore walls.

3. Wellbore Curvature:
   The curvature of the wellbore, including its radius of curvature, also affects torque and drag forces. In curved sections, the drill string experiences bending and torsional forces, which contribute to increased torque. The tighter the radius of curvature, the greater the torque required to navigate the wellbore. Similarly, drag forces can increase in curved sections due to the increased contact area between the drill string and the wellbore walls.

4. Wellbore Friction:
   Friction between the drill string and the wellbore walls is a major contributor to drag forces. The trajectory of the wellbore influences the contact area between

the drill string and the wellbore, affecting the amount of friction encountered. Changes in inclination, azimuth, or curvature can lead to variations in the contact area, resulting in fluctuating drag forces along the wellbore trajectory.

5. Cuttings Accumulation:
   During drilling, cuttings are generated and circulated to the surface. In deviated or horizontal sections, cuttings tend to settle and accumulate on the low side of the wellbore. This accumulation can create additional resistance against the drill string, leading to increased drag forces. Proper well trajectory design, including sufficient inclination and appropriate cleaning techniques, is essential to minimize cuttings accumulation and mitigate its impact on torque and drag forces.

It is crucial to consider the trajectory's effect on the torque and drag forces during well planning and drilling operations. By optimizing the wellbore trajectory and considering factors such as inclination, dogleg severity, curvature, friction, and cuttings management, drilling engineers can mitigate excessive torque and drag forces, improving drilling efficiency, reducing wear and tear on drilling equipment, and minimizing the risk of operational issues.

2.4 Optimization Techniques

Optimization is a fundamental process in various fields that aims to find the best possible solution from a set of feasible options. It plays a crucial role in decision-making, resource allocation, system design, and problem-solving across diverse domains such as engineering, economics, logistics, and computer science.

Optimization techniques encompass a wide range of methods and algorithms that are designed to systematically search for and identify optimal solutions within a given problem space. These techniques enable us to maximize desired objectives, minimize costs, satisfy constraints, or strike a balance between conflicting factors.

The choice of an appropriate optimization technique depends on the nature of the problem, the complexity of the search space, and the specific requirements of the application. Various optimization techniques have been developed over the years, each offering unique advantages and approaches to tackling different types of problems.

One widely used category of optimization techniques is evolutionary algorithms, which draw inspiration from the principles of natural selection and genetic inheritance. Evolutionary algorithms, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), employ populations of candidate solutions that evolve over generations through the application of genetic operators like mutation, crossover, and selection. These algorithms exhibit robustness, adaptability, and the ability to handle complex search spaces [11] [12].

2.4.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a popular optimization technique inspired by the collective behavior of bird flocks or fish schools. It is a population-based, stochastic optimization algorithm that can be used to solve a wide range of optimization problems, including trajectory design in various fields such as robotics, engineering, and computer science.

The concept of PSO is based on the idea of simulating the social behavior of a group of particles, known as a swarm, searching for the optimal solution in a multidimensional search space. Each particle in the swarm represents a potential solution and moves through the search space by adjusting its position and velocity based on its own experience and the collective knowledge of the swarm.

At each iteration, particles evaluate their fitness value, which represents how well their current position satisfies the optimization objective. The particles then update their velocities and positions based on their individual experience, the best solution they have encountered so far (personal best), and the best solution found by any particle in the swarm (global best).

The update process involves balancing exploration (searching for new solutions) and exploitation (exploiting promising regions) by adjusting the particle's velocity and position. By sharing information and learning from each other's experiences, particles can quickly converge to better solutions over iterations [13] [14].

2.4.2 Genetic Algorithm Optimization

Genetic algorithm is a powerful optimization technique that can be applied to various problem domains, including trajectory optimization in drilling operations. They are inspired by the process of natural selection and mimic the principles of evolution to search for the optimal solution within a given problem space. In the context of trajectory design, genetic algorithms offer an efficient approach to finding the best trajectory that satisfies the specified constraints [15].

The genetic algorithm optimization process involves several key steps. Firstly, an initial population of potential solutions, known as individuals or chromosomes, is randomly generated. Each individual represents the location of a single or more than one point between target points in local coordinates. The population is evaluated based on a fitness function, which measures the quality or performance of each individual.

The fitness function is designed to capture the objective of the optimization problem. In the case of trajectory design, it accounts for factors of MD, Torque, and Drag forces combined. The fitness function quantifies the fitness or suitability of each individual within the population.

The next step is the application of genetic operators to create new generations of individuals. These genetic operators include selection, crossover, and mutation. Selection involves choosing individuals from the current population based on their fitness. Individuals with higher fitness are more likely to be selected for the next generation, simulating the concept of survival of the fittest.

Crossover involves combining genetic information from selected individuals to create offspring. This process mimics genetic recombination in nature, where traits from parents are passed on to their offspring. By exchanging and recombining genetic

material, the algorithm explores different combinations of trajectory characteristics and can potentially discover better solutions.

Mutation introduces random changes or modifications to the genetic information of individuals. This helps to maintain diversity within the population and prevents premature convergence to suboptimal solutions. Mutation allows for the exploration of new regions in the search space and can lead to the discovery of previously unexplored trajectories.

The process of selection, crossover, and mutation is repeated iteratively over multiple generations. With each generation, the population evolves, and the fitness of individuals typically improves. The algorithm continues until a stopping criterion is met, which could be a maximum number of generations, the convergence of fitness values, or the attainment of a satisfactory solution [16] [17].

Chapter 3

Methodology and System Architecture

This chapter presents the methodology and system architecture employed in the development of an autonomous trajectory design system. It outlines the workflow followed and provides an overview of the key components and processes involved.

The objective of this chapter is to describe the sequential steps involved in the workflow, from the input stage where user-defined information is provided to the output stage where a final trajectory is generated that satisfies the specified constraints. It also highlights the use of optimization techniques during the trajectory generation process through multiple subsurface target points.

The mathematical models used in the trajectory design process with calculations of azimuth, inclination, DLS, torque, and drag forces are described in detail. Moreover, the details of optimization algorithms are described for readers. The inclusion of these mathematical calculations and algorithmic formulas provides a comprehensive understanding of the underlying principles and methodologies employed in the autonomous trajectory design system. By examining these calculations, readers can gain insight into the technical aspects of trajectory generation and optimization.

3.1 Trajectory Design Workflow

The brief workflow for the developed system is presented in Figure 3.1. It begins with the input stage, where the user provides the necessary information. This includes specifying the geological target points along with optimization points and defining the maximum Dog Leg Severity and the pipe/hole properties required for torque and drag calculations. The coordinates of points that the user selects as optimization points will be evaluated, changed, and optimized using genetic algorithm.

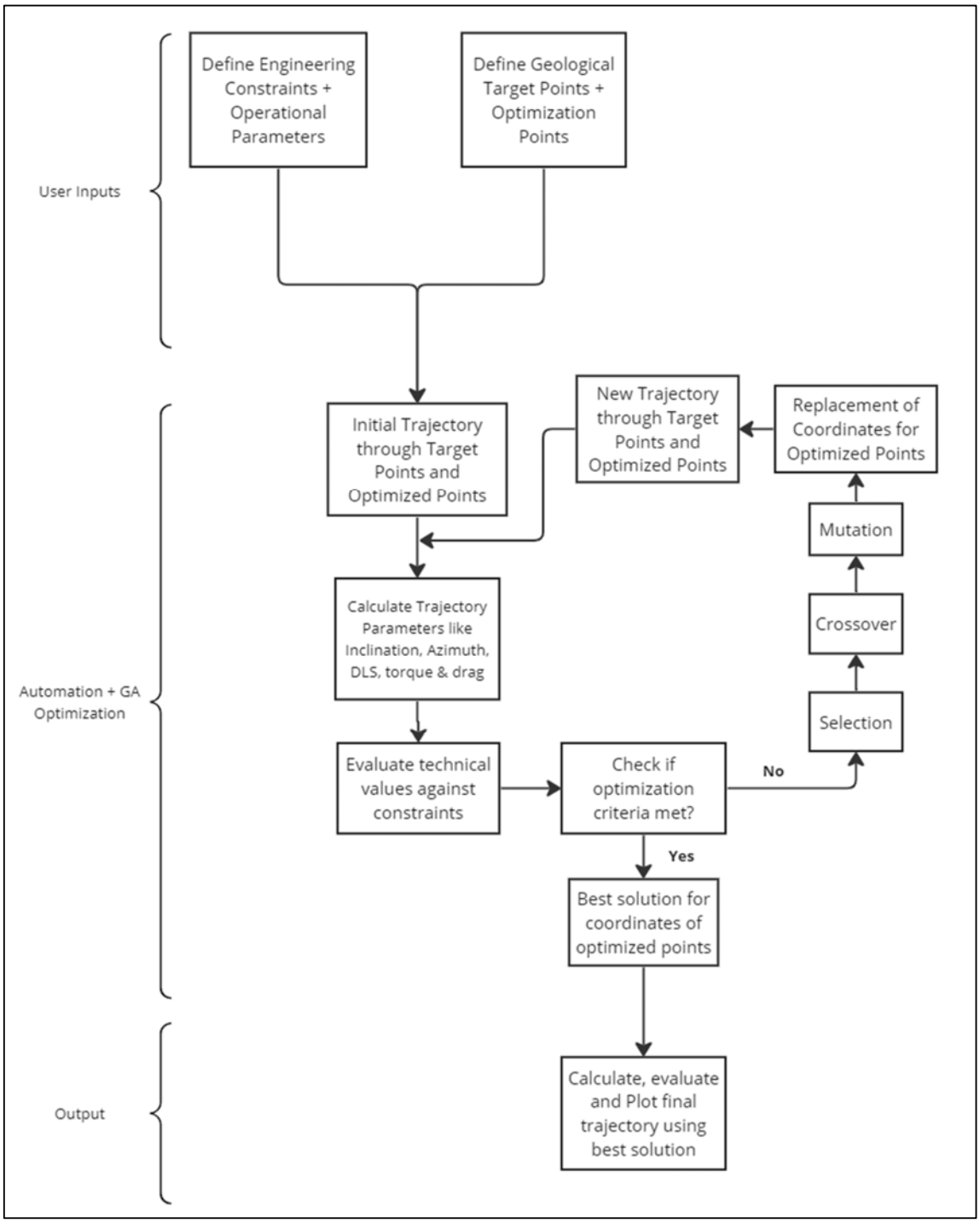

Figure 3.1 Optimized Trajectory Design System

In the automation and optimization phase, an autonomous trajectory generation algorithm is implemented. This algorithm automatically generates an initial trajectory to calculate various trajectory parameters out of which MD, Torque, Drag, and DLS are used in an optimization algorithm named GA (Genetic Algorithm). GA then

evaluates the objective function and constraints to calculate new coordinates for the optimization points. Here multi-objective optimization is used where MD, Torque, and Drag are combined in one equation, and the objective of GA is to minimize it. Moreover, constraints for maximum DLS and maximum torque as makeup torque is also defined for GA to follow.

After multiple iterations, GA gives the best coordinates for the optimized point, using which trajectory is generated and plotted. To optimize the overall trajectory, the optimization algorithm can modify the path between the multiple target points if we give optimization points between all target points. It is also possible for users to optimize the target point itself if required. This algorithm ensures that the constraints, such as the maximum DLS and torque and drag limitations, are met while generating the final optimized trajectory.

The output of the thesis workflow is the final best trajectory that satisfies the user-defined constraints. A use case for this system is implemented and analyzed using the extended reach well example in Chapter 40. It is presented as the result of this research, showcasing the ability of the autonomous trajectory design system to generate trajectories that adhere to the specified limitations.

Overall, the algorithm proposed in this thesis encompasses the generation of trajectories based on user inputs, their optimization considering torque and drag constraints, and the delivery of a final trajectory that meets the desired objectives.

3.2 System Inputs

The inputs of the system consist of following parameters:

1. Given target points
2. Number of optimization points
3. Configuration and Operational data
4. Engineering Constraints

3.2.1 Given Target Points

The first step of the trajectory design system is defining the geological target points. They are defined in a local coordinate system, using Northings (N), Eastings (E), and True Vertical Depths (TVD) in the form of arrays of [N, E, V]

The local coordinate system provides a consistent reference framework for representing the target points and can be easily converted to other systems like UTM. By using Northing, Easting, and vertical position, each target point is specified relative to a reference point of wellhead location. Wellhead location is defined as [0, 0, 0] corresponding to [N, E, V]

To illustrate the concept, consider one point which is defined as:

$$[N, E, V] = [0, 0, 500] \text{ (meters)}$$

Here, the Northing is 0 meters, the Easting is 0 meters, and the TVD is 500 meters. This indicates that the first target point is located 500 meters below the wellhead position.

Moving on to the second target point, let's say it is [100, 100, 700], we observe that the Northing and Easting values are 100 each, suggesting that the point moved 100 meters towards north and 100 meters towards east respectively. However, the TVD value is 700, indicating that the point is positioned 700 meters below the reference elevation. As we progress to define other target points through the array, each subsequent target point represents a specific location in 3D space through which trajectory should be generated.

Here, it is important to note that the negative sign with Northing and Easting represents opposite directions towards south and west respectively.

3.2.2 Number of Optimization Points

Along with defining the target points, users can define the **multiple optimization points** using unknown coordinates, like using arrays of zeros or any other digits as initial guess.

So, it is required to identify each coordinate as either given (target point) or require optimization (optimized point). There is no restriction on number of consecutive optimization or target points. User has freedom to define whatever point as optimization point along the trajectory. This is demonstrated in an example shown in Table 3.1 where points 2, 4 and 5 are marked as the ones that require optimization, while other points are marked as given target points.

Table 3.1 Structure of Target and Optimization Points

Point #	Coordinates (m)	Type
0	[0, 0, 0]	Given
1	[0, 0, 500]	Given
2	[X_1, X_2, X_3]	Require Optimization
3	[100, 100, 700]	Given
4	[X_4, X_5, X_6]	Require Optimization
5	[X_7, X_8, X_9]	Require Optimization
6	[520, 370, 1200]	Given

The optimization algorithm will automatically calculate the coordinates for the points that require optimization considering the limitations and constraints defined by user.

3.2.3 Configuration Data and Operational Data

To effectively perform trajectory design and optimization, the system requires certain necessary configuration data and operational data. These configuration data provide essential inputs that define the physical properties of the drilling system. Among the crucial configuration data are parameters related to the drill pipe, including its inner diameter (ID), outer diameter (OD), and density. While operational data includes weight on bit, torque on bit and mud weights. These parameters are essential for accurately calculating torque and drag forces exerted on the drill string during drilling

operations. All the parameters that needed to be defined for the system are listed in Table 3.2

3.2.4 Engineering Constraints

In addition to specifying geological target points and optimization points, the autonomous trajectory design system incorporates engineering constraints to ensure desired trajectory design. These constraints are essential in guiding the trajectory generation process and optimizing the drilling path accordingly.

One of the primary engineering constraints considered is the maximum Dog Leg Severity. It measures the rate of change of wellbore inclination and azimuth and controlling it within predefined limits is essential to reduce the risks of drill string failures, operational failures, and tool failures. By defining a maximum DLS value, the trajectory design system ensures that the final generated path does not exceed this limit at any point in the wellbore path. If GA does not find any path that satisfies the defined maximum DLS within the maximum iterations defined, it gives the closest value to the maximum DLS defined.

Furthermore, torque and drag calculations are integrated for assessing the mechanical forces acting on the drill string during drilling operations. Torque refers to the rotational force required to turn the drill string, while drag is the resistance encountered by the drill string as it moves through the wellbore. Excessive torque and drag can lead to increased energy consumption, equipment wear and tear, and potential drilling issues. Therefore, the trajectory design system considers the torque and drag limitations by considering factors such as drill string properties, wellbore friction factors, mud properties, and drilling parameters. Parameters that needed to be defined are listed in Table 3.2

By defining these constraints, the autonomous trajectory design system ensures that the generated trajectory is also compliant with the operational limitations. The integration of these constraints into the trajectory optimization algorithm allows for the iterative modification of the path, aiming to achieve an optimal trajectory that meets the user-defined constraints.

Table 3.2 Description of Input Parameters

Parameter	Description	Unit
DLS_max	Maximum DLS to be used as a constraint for GA optimization.	deg/30 meters
nt	Used for controlling the resolution of generated trajectory. It controls the distance between t values for the Bezier curve which varies from 0 to 1.	-
Pipe od	Pipe outer Diameter	inches
Pipe id	Pipe Inner Diameter	inches
Pipe length	Calculated by pipe bottom (default = Total MD) and pipe top (default = 0)	meters
Pipe makeup_torque	Makeup torque value for pipe	kN * meter
odAnn	Diameter of Hole or casing	inches
wob	Weight on bit (it effects the axial forces and have impact in calculation of drag forces)	kN
tbit	Torque on bit (It impacts the torque values of the drill string)	kN * meter
fric	Friction factor (can be an array of multiple friction factors against MD for sections).	-
densities	Mud density and pipe density (default pipe density = 7.8)	sg
max_iter	Maximum Iterations for the optimization problem	-

3.3 System Automation and Optimization

After target points along with constraints and operational parameters are given as input, the system moves to the next phase of automation and optimization. Automation and optimization are key components of this system, aimed at enhancing the efficiency and accuracy of trajectory design in drilling operations.

Automation involves the development of computational methods that can autonomously generate well trajectories based on user-defined inputs using multiple Bezier splines. By automating the trajectory design process, time-consuming manual iterations can be minimized allowing for faster trajectory generation.

Optimization, on the other hand, focuses on refining and improving the generated trajectories. GA algorithm iteratively modifies the path between target points to optimize the trajectory while ensuring that the specified constraints, such as maximum Dog Leg Severity and torque and drag limitations, are met.

The combination of automation and optimization not only streamlines the trajectory design process but also leads to the discovery of more optimal drilling paths.

3.3.1 Generating Trajectory using Bezier Curves

Trajectory design involves a step-by-step process that begins with receiving user-defined target points, maximum DLS (Dog Leg Severity), and torque/drag constraints as input.

Jie Cao and Dan Sui (2022) [18] described the detailed process to design trajectories by using multiple Bezier splines. The first step of trajectory design in three-dimensional space is the calculation of control points for cubic Bezier curves between target points. These control points are calculated with continuity conditions that enable the creation of a smooth and continuous trajectory.

A Bezier curve **C(t)** is a parametric curve based on Bernstein polynomials B(t). It is defined in (4).

$$\boldsymbol{C}(\boldsymbol{t}) = \sum_{i=0}^{n} B_{n,i}(t)\boldsymbol{P}_i$$

(4)

where,

$\boldsymbol{P_i}$ are the control points.

n is the degree of the curve. As we are using cubic Bezier curves, the degree of the curve will be three.

t is the curve function parameter, and it varies from 0 to 1.

$B_{n,i}$ is the th Bernstein polynomials of degree n

$$B_{n,i}(t) = \frac{n!}{i!\,(n-i)!} t^i (1-t)^{n-1}$$

(5)

It should be noted that the Bezier curve always interpolates the first and last control points but not necessarily the other middle control points [18].

The cubic Bezier curve in three dimensions is mainly used in this research and it is expressed in equation (6).

$$\boldsymbol{C}(\boldsymbol{t}) = (\mathbf{1}-t)^{\mathbf{3}}\mathbf{P}_0 + \mathbf{3}(\mathbf{1}-t)^{\mathbf{2}}t\mathbf{P}_1 + \mathbf{3}(1-t)t^{\mathbf{2}}\mathbf{P}_2 + t^{\mathbf{3}}\mathbf{P}_3$$

(6)

where **P₀**, **P₁**, **P₂**, and **P₃** are the control points considering degree n = 3. The matrix of a cubic Bezier curve is defined in equation (7).

$$\mathrm{C(t)} = \begin{bmatrix} (1-t)^3 \\ 3(1-t)^2 t \\ 3(1-t)t^2 \\ t^3 \end{bmatrix}^T \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} = \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}^T \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}$$

(7)

C(t) in equation (7) is just one curve between $\mathbf{P_0}$ and $\mathbf{P_3}$ obtained by varying values of t from 0 to 1. Consider P_0 as the first target point and P_3 as the second target point. More target points can be connected using multiple curves by satisfying the continuity conditions. There are three continuity conditions C^0, $C^{1,}$ and C^2 described by Jie Cao and Dan Sui [18]. These are:

1. C^0 continuity condition ensures that the well path is continuous. However, it may result in discontinuous azimuth and inclination angles, which do not satisfy the requirements of a well-path design.

2. C^0 and C^1 continuity ensure the designed well path becomes not only continuous but also smooth. This means that the azimuth and inclination of the well path are continuous throughout.

3. C^0, C^1 and C^2 continuity ensures that in addition to continuous azimuth and inclination, the Dog Leg Severity remains continuous along the measured depth of the trajectory.

The choice of continuity conditions can be adjusted based on the specific requirements of the path design. Conventional well path designs typically do not necessitate the C^2 continuity condition, as C^0 and C^1 continuity is often sufficient to achieve the desired continuity and smoothness. In our algorithm, we used the C^2 continuity condition to generate trajectory after KOP.

3.3.2 Calculation of Trajectory Parameters

To determine the azimuth, inclination, DLS, and other parameters for each section of the trajectory, mathematical calculations are discussed here. These calculations consider the defined target points and calculated control points. Then trajectory is generated through those using control points before calculating trajectory parameters. By accurately calculating these parameters, the trajectory can be precisely guided to achieve the intended goals of optimization defined by the user.

Let's consider a well-path design using a cubic Bezier curve, which provides a smooth trajectory in three-dimensional space. The schematic representation of this well path is depicted in Figure 3.2 Example of 3D Trajectory Through Two Points with the Cartesian coordinate system of (N, E, V), or equivalently (x, y, z), where N represents the north direction, E represents the east direction, and V represents the vertical direction [18].

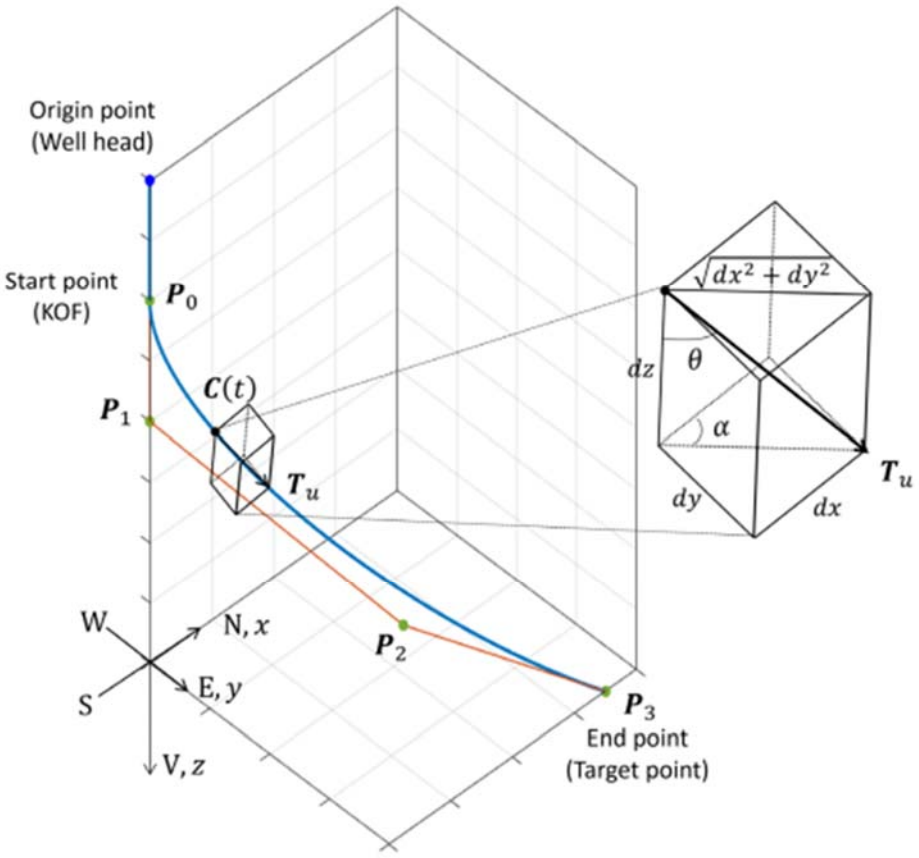

Figure 3.2 Example of 3D Trajectory Through Two Points [18]

At any arbitrary point C(t) along the well path, we can consider a small element of displacement represented by **dx**, **dy**, and **dz**. These elements correspond to infinitesimal changes in the north, east, and vertical directions, respectively. This small element allows us to analyze the local behavior of the well path at that specific point.

By examining the local properties of the well path, we can gain insights into its curvature, inclination, and changes in direction. These properties are essential in

understanding the behavior of the wellbore and ensuring its successful navigation with desired properties.

The characteristics of the well trajectory are determined through the utilization of the Bezier curve function **C(t)** and its derivatives. The tangent vectors of the Bezier curve are denoted as **T**, as depicted in Figure 3.2. Additionally, the unit tangent vector is represented as $\mathbf{T_u}$. By utilizing these vectors, we can derive trajectory-specific properties.

$$\boldsymbol{T} = \frac{dC(t)}{dt} = C'(t) \tag{8}$$

$$\mathbf{T}_u = \frac{\mathbf{T}}{\|\mathbf{T}\|} = \frac{C'(t)}{\|C'(t)\|} \tag{9}$$

and,

$$C'(t) = [C'_x(t), C'_y(t), C'_z(t)]^T \tag{10}$$

The unit tangent vector at any point along the curve can also be expressed in terms of its coordinates using the inclination (θ) and azimuth (α)

$$T_u = (sin\theta.cos\alpha\ , sin\theta.sin\alpha\ , cos\theta) \tag{11}$$

By combining equation (10) and (11), we can determine the **inclination** (θ) and **azimuth** (α) at a specific point along the curve.

$$tan\ \alpha = \frac{dy}{dx} \tag{12}$$

$$tan\,\theta \;=\; \frac{\sqrt{dx^2 \;+\; dy^2}}{dz}$$

(13)

where

$$dx \;=\; C_x'(t), \qquad dy \;=\; C_y'(t), \qquad and \qquad dz \;=\; C_z'(t)$$

(14)

The **Curvature of a Bezier Curve** in three dimensions is given by equation (15)

$$k(s) \;=\; \frac{\sqrt{(\mathrm{C}_z''C_y' \;-\; \mathrm{C}_y''C_z')^2 + (\mathrm{C}_x''C_z' \;-\; \mathrm{C}_z''C_x')^2 + (\mathrm{C}_y''C_x' \;-\; \mathrm{C}_x''C_y')^2}}{(C_x'^2 \;+\; C_y'^2 \;+\; C_z'^2)^{\frac{3}{2}}}$$

(15)

where,

s is the **arc length** on a Bezier curve and is given by equation (16)

$$s \;=\; \int_0^t \sqrt{\|\boldsymbol{C}'\|}\; d\tau \;=\; \int_0^t \sqrt{C_x'^2 \;+\; C_y'^2 \;+\; C_z'^2}\; d\tau$$

(16)

The curvature is directly determined by the first and second derivatives of the path curves, highlighting the importance of smoothness in influencing the curvature values. Similarly, the DLS is sensitive to the smoothness of the path, as it is determined based on the curvatures according to the relationship described in equation (17) in units of degrees/30 meters.

$$DLS \;=\; \frac{180 \times 30\; k(s)}{\pi}$$

(17)

Therefore, when designing the well path using the Bezier curve, the crucial element is the control points. The properties of the well path, such as inclination and azimuth, are

functions of the parameter t, corresponding to a specific measured depth along the trajectory.

3.3.3 Torque and Drag Model

To calculate torque and drag forces, a mathematical model and algorithms suggested by Johancsik, Friesen, & Dawson, 1984 [19] are employed which is a soft string model. This model considers the geometry and properties of the wellbore, drill string, and mud, along with the drilling parameters. They consider the interactions between the drill string and the wellbore, including the effects of contact friction and wellbore tortuosity. By simulating the drilling operation and incorporating these factors, torque and drag forces can be calculated and analyzed.

The calculation begins at the bottom of the drill string and proceeds upward, considering each short element of the drill string and its contribution to the overall axial and torsional load. The first step in the calculation is determining the normal force acting on a short, slightly curved element of the drill string. This is achieved by considering the weight of the element (W) and the two tension forces ($F_t + \Delta F_t$) exerted by the drill string. The net normal force (F_n) is obtained by taking the negative vector sum of the normal components from the weight and tension forces as shown in Figure 3.3.

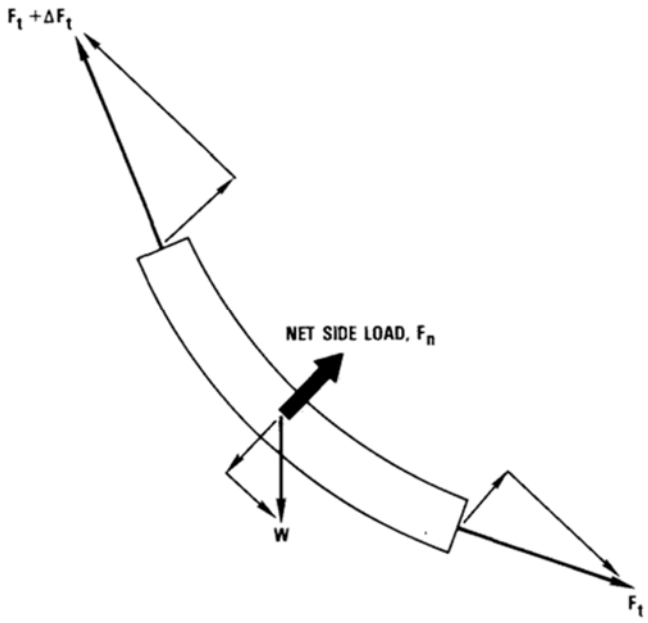

Figure 3.3 Force Balance on Drill String Element [19]

While the axis of the element is assumed to be an arc of a circle, it is important to note that this circle is not usually vertical. As a result, the net normal force is not typically in the vertical plane. However, for friction calculations, only the magnitude of the normal force is required, not its direction [19].

The magnitude of the normal force can be calculated using the equation (18).

$$F_n = [(F_t \, \Delta\alpha \, sin\theta)^2 + (F_t \, \Delta\theta + W sin\theta)^2]^{\frac{1}{2}} \quad (18)$$

Equation (18) considers the weight of the element **W**, the tension forces $\mathbf{F_t}$, the **inclination** (θ), and **azimuth** (α) to determine the normal force. It serves as a fundamental component for further calculations related to torque and drag forces.

The equation for the tension increase is given in equation (19)

$$\Delta F_t = W cos\theta \pm \mu F_n \quad (19)$$

Similarly, the torsional increase is given in equation (20)

$$\Delta M = \mu F_n r \quad (20)$$

Equation (19) for calculating torque and drag forces includes a plus or minus sign to account for the direction of pipe motion, whether it is upward or downward. The plus sign corresponds to upward motion, where friction adds to the axial load, while the minus sign corresponds to downward motion, where the opposite occurs. In practice, when presenting data, this sign is often associated with the friction coefficient μ, allowing for the identification of coefficients calculated from slack-off drag measurements. [20]

All these equations provide an exact representation when applied to infinitesimal elements of the drill string. However, when longer elements are considered, small errors are introduced due to the neglect of second-order terms.

By following this stepwise approach and incorporating the necessary mathematical models and algorithms, it becomes possible to calculate and analyze the torque and drag forces experienced during drilling operations.

3.3.4 Integration and Modification of Torque and Drag Model

To incorporate the calculation of torque and drag forces into the trajectory design process, Open-source Torque and Drag Model in python language Version0.1.1 available at [https://pypi.org/project/torque-drag] is utilized as a foundational framework. This model provides a solid foundation for torque and drag calculations, considering various factors such as wellbore geometry, drill string properties, mud characteristics, and drilling parameters.

However, to meet the specific requirements of the trajectory design methodology in this thesis, certain modifications are made to the existing torque and drag model. One notable addition is the consideration of makeup torque as a constraint. Makeup torque refers to the torque applied to the threaded connections during the assembly of the drill string. It provides the limitation of torque that can be applied to drill string during

the drilling operation. Exceeding makeup torque can damage the connections permanently.

By incorporating makeup torque limitations into the torque and drag model, the trajectory design process becomes more comprehensive. The modified model takes into account the maximum allowable makeup torque at the connection point along the drill string. This ensures that the generated trajectories adhere to the physical limitations imposed by makeup torque, thereby enhancing the practicality and reliability of the resulting well paths.

3.3.5 Optimization Algorithm for Trajectory Design

Optimization of generated trajectory can be accomplished by adding additional points in between target points through which the trajectory needs to be passed. The number and location of points play an important role in determining the overall trajectory.

Once the initial trajectory is designed, an iterative optimization process is implemented by varying the coordinates of optimization points. GA optimization is implemented to achieve the required position of the point according to user-defined constraints with the combined objective of minimizing MD, torque, and drag forces. The objective function is defined by combining these three parameters as per equation described in (21).

$$Combined\ cost\ function\ = \frac{1}{3} \times MD\ + \frac{1}{3} \times T\ + \frac{1}{3} \times F \quad (21)$$

where,

MD = Measured Depth, meters
T = Torque, kN.m
F = Drag Force, kN

Note that here all MD, T and F are given equal weightage. Here all these parameters are directly proportional to each other, so their weightage does not affect the overall

impact on result considerably. This means increasing MD also increases torque and drag forces in general. But in future cases, when more cost functions will be included, their weightages will also play a considerable role if they are inversely proportional to each other.

Other optimization parameters that needed to be defined are described in Table 3.3

Table 3.3 Description of Optimization Parameters

Optimization Parameter	Description
Objective function	A combined function including MD, Torque, and Drag is introduced in our system for optimization
Variables: n_dim	These are equal to the number of coordinates for optimization points (For Example, it is 3 variables in case of 1 optimization point corresponding to [N, E, V]
lb	Lower boundary for search space
ub	Upper boundary for search space
max_iter	Maximum number of iterations allowed during the optimization process
constrain_eq	Functions defining constraints. In this system, there are two constraints, DLS and Makeup Torque.

This process evaluates the initial trajectory based on criteria. Optimization algorithms are then employed to modify the trajectory iteratively. Small perturbations or adjustments are introduced to the position of the optimization point to generate new candidate trajectories. The optimization process aims to select the fittest trajectories

that meet the engineering constraints while discarding weaker alternatives. Multiple iterations refine the trajectory until the optimization objectives are met.

The final trajectory is generated once the optimization process is completed. This trajectory satisfies the user-defined constraints and objectives. Graphical representation of trajectory profile, azimuth, inclination, DLS, torque, and drag provides a clear visualization of the trajectory and facilitates analysis and evaluation.

3.4 System Outputs

The finalized trajectory using the best-optimized solution is plotted using a Python package named plotly [21]. The best-optimized solution is in the form of coordinates for optimized points which are shown as [X_1, X_2, X_3], [X_4, X_5, X_6] and [X_7, X_8, X_9] in Table 3.1.

Using the coordinates of these optimized points and target points, Six graphs are generated for detailed analysis and visualization. Three out of six graphs visualize the generated trajectory, and the remaining shows the trajectory parameters and torque and drag values for generated trajectory. List of these graphs is listed below:

1. TVD (meters) vs Easting (meters)
2. Northing (meters) vs Easting (meters)
3. 3D Trajectory Design
4. Azimuth (°), Inclination (°), DLS (°/30 meters) vs MD (meters)
5. Torque (kN. meter) and Makeup Torque vs MD (meter)
6. Drag Force (kN) vs MD (meter)

All these graphs will be shown with case studies in Chapter 4.

3.5 System Architecture

The system is coded in python programming language, and it contains a total of six files. The names of the files along with their short description are given in Table 3.4. Some of the coded files are also given in Appendix.

In the development of this Python code, several open-source Python packages were utilized, providing a solid foundation of functionality and capabilities. These packages served as powerful tools for implementing optimization algorithms, data manipulation, visualization, and other essential functionalities. The following list highlights some of the key Python packages that were employed:

1. NumPy: A fundamental package for scientific computing in Python, offering powerful numerical operations and array manipulation capabilities [22].
2. SciPy: A comprehensive library for scientific and technical computing, providing a wide range of optimization algorithms, statistical functions, and numerical tools [23].
3. Pandas: A versatile data manipulation and analysis library, enabling efficient data handling, transformation, and exploration [24].
4. Plotly: An interactive data visualization library that allows the creation of interactive plots, charts, and dashboards [25].
5. Well_profile: A Python package specifically designed for wellbore trajectory analysis and visualization, providing tools for trajectory calculation, plotting, and analysis.
6. Torque_drag: A Python library for torque and drag analysis in drilling operations, enabling the calculation of torque, drag, and related forces. This library was modified according to our requirements before implementation.
7. scikit-optimize (sko): A library for sequential model-based optimization, offering various optimization algorithms and tools for hyperparameter tuning and global optimization.

These open-source Python packages played a crucial role in the development of this code, offering reliable and efficient solutions to various computational tasks. By leveraging the capabilities of these packages, it was possible to focus on the specific implementation of optimization algorithms to solve the challenge at hand.

Table 3.4 Description of System Files Coded in Python.

File Name	Short Description
main.py	Main file where input parameters and optimization parameters are defined. All other functions are called through this file.
kop_optimization.py	This is a modified version of main.py especially for KOP optimization.
gn_bezier_fit.py	The function in this file generates control points based on multiple cubic Bezier spline equations.
pascal.py	A function that generates coefficients for the Bezier equation based on pascal's triangle.
bezier_eval3.py	This function evaluates the Bezier curve described by control points. It returns coordinates, inclination, azimuth, curvature, and MD of the complete trajectory.
evaluate_bezier_struct3.py	This function populates the structure of piecewise Bezier splines.
Optimization_problem.py	Here objective functions and cost functions are defined for optimization problems. The function for plotting the final solution is also defined here.

Chapter 4

Case Studies and Results

In this chapter, a case study is presented to demonstrate the application and effectiveness of the autonomous trajectory design system developed in this thesis. One of the primaries focus of this section is to showcase how our algorithm can be implemented in real-world scenarios with tangible benefits. We will examine how it can automate well-planning processes, optimizing the Kick-Off Point, surface points, and subsurface points. By automating these processes, we can enhance efficiency and overall operational success.

The first case study focuses on an extended reach well scenario, where the objective is to drill a wellbore that reaches a target point located at a significant horizontal distance from the drilling rig. The results obtained from the case study provide valuable insights into the capabilities of the autonomous trajectory design system in generating optimized well trajectories that adhere to user-defined constraints.

The second case study focuses on the optimization of KOP. The objective of this case study is to optimize the KOP location to achieve user-defined trajectory targets like DLS and torque limits while connecting drilling targets in an optimized way.

4.1 Case Study on Extended Reach Well

4.1.1 Extended Reach Wells

Extended reach wells are a specialized type of well designed to access remote or challenging reservoirs from a single drilling location. These wells are characterized by their extended lateral length, which presents unique challenges in terms of trajectory design and T&D limitations. ERWs offer numerous advantages such as increased reservoir contact, reduced environmental impact, and improved cost-effectiveness.

One of the key challenges in ERW drilling is wellbore friction. Optimizing the well path design is an effective means of reducing torque and drag [26]. As the lateral length increases, maintaining the desired wellbore path becomes more complex. The trajectory must be carefully planned to navigate through multiple geological targets while considering technical constraints such as T&D, hydraulics, and wellbore stability [27].

To address these challenges, advanced drilling technologies have been developed specifically for ERW operations. Rotary steerable systems are one of the tools that are often used in directional drilling these days as they offer greater control and precision in wellbore steering, enabling operators to navigate complex trajectories more effectively. Planning complex trajectories and optimizing them is still an area of active research [28] [29].

4.1.2 Getting ERWs Data from ProWellPlan

Access to the required data for the case study was facilitated through the ProWellPlan platform, which serves as a comprehensive repository of historical well data. The platform offers a user-friendly interface that allows users to search and retrieve data using various filtering parameters. In this study, the historical database available with ProWellPlan was utilized. By using a combination of filters such as Measured Depth greater than 7500 meters, and True Vertical Depth less than 2000 meters, extended reach wells for the case study were identified. 13 numbers of wells were identified throughout the database. The platform's advanced search capabilities enable precise data retrieval, ensuring that wells meeting the specified criteria are obtained.

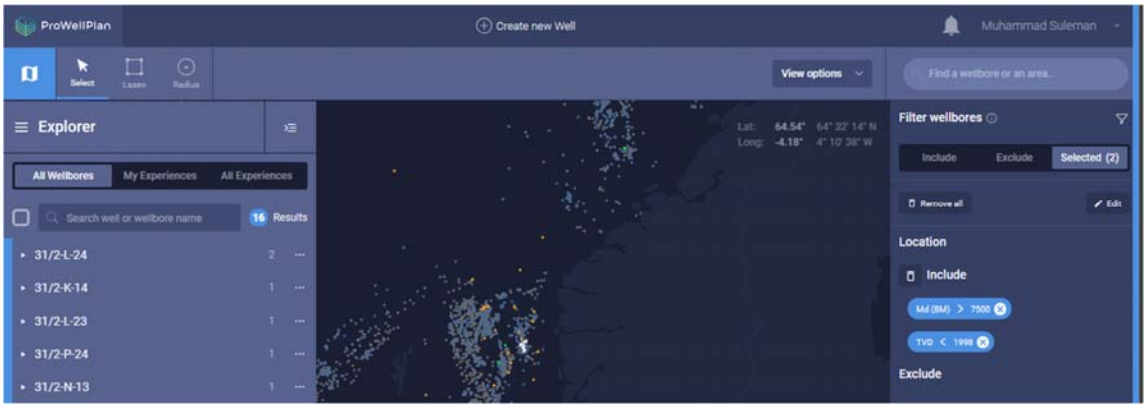

Figure 4.1 Screenshot of ProWellPlan Filtering Feature

Once the data has been filtered and the desired wells have been identified, the ProWellPlan platform provides visualization tools for a comprehensive analysis of the wells' characteristics and properties. These visualization features allow users to gain insights into the well trajectories, geological formations, and other relevant information. Furthermore, the platform enables users to download the selected well data for further research and analysis, ensuring seamless integration with the case study workflow.

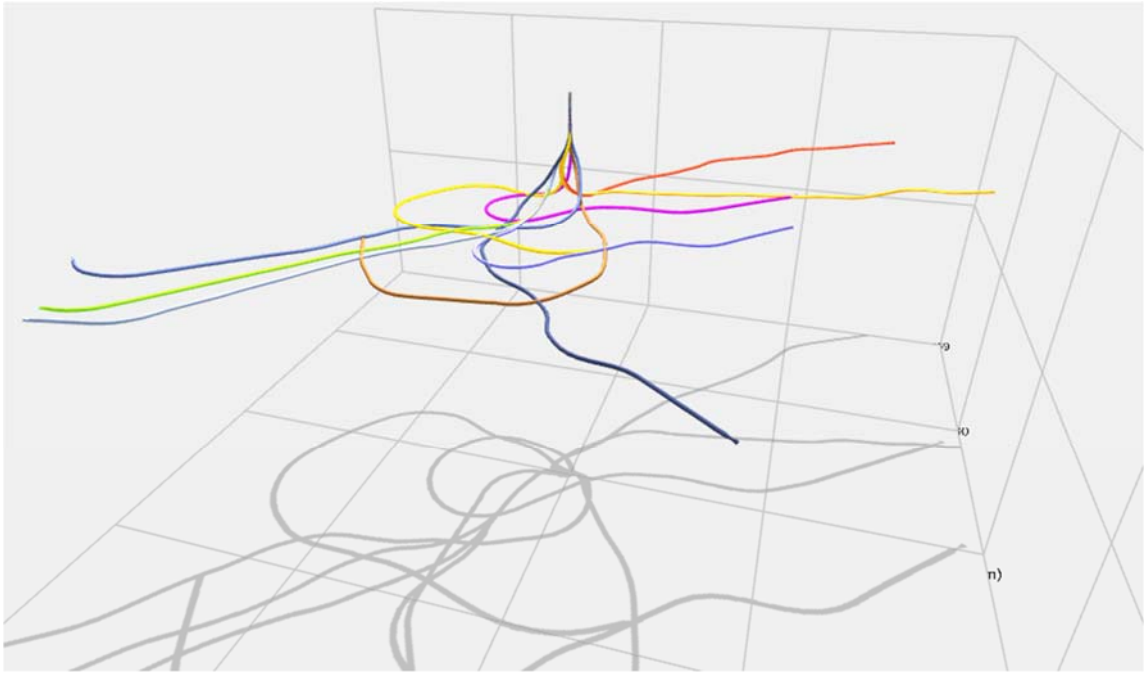

Figure 4.2 Filtered Trajectories for ERWs using ProWellPlan

One of the trajectories is selected for our case study. The survey points for this well are given in Table 4.1

Table 4.1. Survey Points for Extended Reach Well

MD(m)	TVD(m)	Inclination (°)	Azimuth (°)	Northing(m)	Easting(m)	DLS(°/30m)
0	0	0	0	0	0	0
200	200	0	0	0	0	0
400	400	0.3	135	0.02	0.13	1.5
600	599.43	9	33.1	8.51	6.36	3
800	790.39	23.4	24.2	58.77	33.83	0.6
1000	976.56	15.3	38.1	125.1	62.17	2.4
1200	1170.18	17.9	102.5	138.98	107.01	2.4
1400	1349.55	34.9	125.1	100.97	184.56	3.5
1600	1489.08	55.6	131.3	11.5	294.38	3.2
1800	1573.47	74.9	122.4	94.55	439.86	3
2000	1592.44	90.1	116.5	194.72	610.94	1.5
2200	1592.72	89.9	114	278.6	792.48	0
2400	1592.44	90.3	124.4	372.95	968.19	2.1
2600	1589.32	89.8	119.6	477.18	1138.74	0
2800	1585.88	93.1	125.5	583.48	1307.94	3
3000	1583.29	90.1	134.5	717.85	1455.52	1.5
3200	1585.68	88.2	134.5	857.23	1598.89	1.5
3400	1588.13	90.1	135.7	996.86	1742.02	3
3600	1587.91	90	131.1	1136.17	1885.4	1.5
3800	1583.68	89.9	127.1	1261.31	2041.28	1.5
4000	1586.35	87	124.2	1378.68	2203.1	1.5
4200	1586.56	92.4	117.5	1485.48	2372	3.4
4400	1583.88	89.3	117.9	1574	2551.23	1.5
4600	1585.77	89.3	139.4	1697.46	2706.91	2.1
4800	1585.49	89.7	145.9	1858.75	2825	2.1
5000	1585.09	89.8	148.8	2029.57	2928.95	1.5
5200	1584.5	90.8	150.4	2200.06	3033.5	2.1
5400	1584.82	89.6	147.4	2370.96	3137.3	1.5
5600	1584.35	90.1	134.4	2527.62	3260.91	2.1
5800	1583.89	89.7	125.9	2648.38	3419.92	4.2
6000	1584.18	90	129.9	2775.25	3574.4	1.5
6200	1583.07	90	151.2	2927.63	3702	4.5
6400	1582.51	90.6	159.6	3112.5	3777.86	0
6600	1580.78	90.4	158.4	3294.63	3860	1.5
6800	1579.8	90.5	146.2	3472.04	3951.19	1.5
7000	1579.62	90.6	151.6	3645.47	4050.68	0
7200	1579.43	89	138.6	3809.99	4163.42	1.5
7400	1580.83	89.7	147.2	3967.48	4286.38	2.1
7600	1581.23	90	153.7	4143.93	4380.23	0
7800	1579.43	89.8	151.2	4320.26	4474.46	2.1
8000	1579.66	90.1	152.8	4496.07	4569.77	1.5
8032	1579.5	90	153.1	4524.58	4584.3	0

By leveraging the data access and visualization capabilities of the ProWellPlan platform, this study benefits from its reliable and efficient data retrieval process, ensuring the availability of accurate and relevant information for the case study analysis. Figure 4.3 depicts the graph of ERW from ProWellPlan.

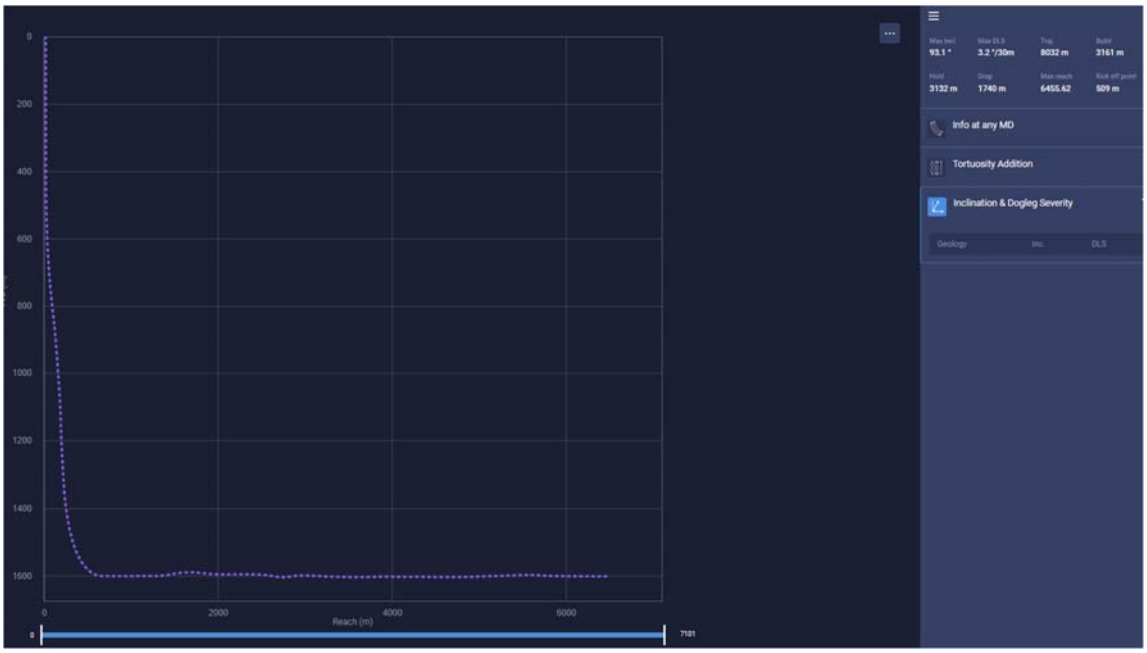

Figure 4.3 Screenshot of ERW from ProWellPlan

Some of the survey points are assumed to be target points in our case study. These target points are then used in our system to check and compare the results. In between KOP and the first target point, an optimization point is given to the system so that it can optimize the trajectory there.

Table 4.2 Target Points for Extended Reach Wells

	North (m)	East (m)	TVD (m)
Surface Point	0	0	0
KOP Point	0	0	400
Optimization Point	X_1	X_2	X_3
Target Point	717.85	1455.52	1583.29
Target Point	1378.68	2203.1	1586.35
Target Point	2029.57	2928.95	1585.09

Target Point	2775.25	3574.4	1584.18
Target Point	3645.47	4050.68	1579.62
Target Point	4320.26	4474.46	1579.43
Target Point	4524.58	4584.3	1579.5

4.1.3 Other Input Parameters

In addition to target points, the following properties of pipe and hole are included in the system for calculations as shown in Table 4.3.

Table 4.3 Hole and Pipe Properties for ERW Case Study

	Value	unit
Pipe OD	5.5	inch
Pipe ID	5	inch
Hole Size	8.5	inch
friction factor	0.24	
WOB	50	kN
torque on Bit	15	kN x m
mud weight	1.3	sg
Makeup Torque	53	kN x m

Note that we are only using a single drill pipe for calculations. It is also possible to generate the results for a more complex drill string that includes all the components like Drill Collars, Heavy Weight Drill Pipes, Rotary Steerable System, and Stabilizers. Also, it is possible to give different friction factors against each MD in the form of an array. For testing, we are only using a simplified case. More testing can be done in case of all data availability.

Optimization parameters used for the genetic algorithm before running it are depicted in Table 4.4. Note that we use maximum iteration of only 100 to reduce optimization time. It was successful because we defined the upper and lower boundary for search in between the coordinates of previous and next point. Due to this search space was considerably reduced and large number of iterations were not required.

Table 4.4 Optimization Parameters

	Value	unit
Max DLS	2.5	°/30m
Max iteration	100	
Population size	100	
Constraints	Max DLS, Max Makeup Torque	

4.1.4 Results

The case study analysis yielded promising results, demonstrating the effectiveness of the autonomous trajectory design system in generating well trajectories that adhere to user-defined constraints. The focus of the study was to achieve trajectory design by fulfilling user-defined Dog Leg Severity value and makeup torque while connecting the target points.

Figure 4.4 visualizes the generated trajectory with the definition of KOP, optimized point, and target points. The same trajectory is represented with the top view in Figure 4.5 whereas Figure 4.6 represents a 3D view of generated trajectory.

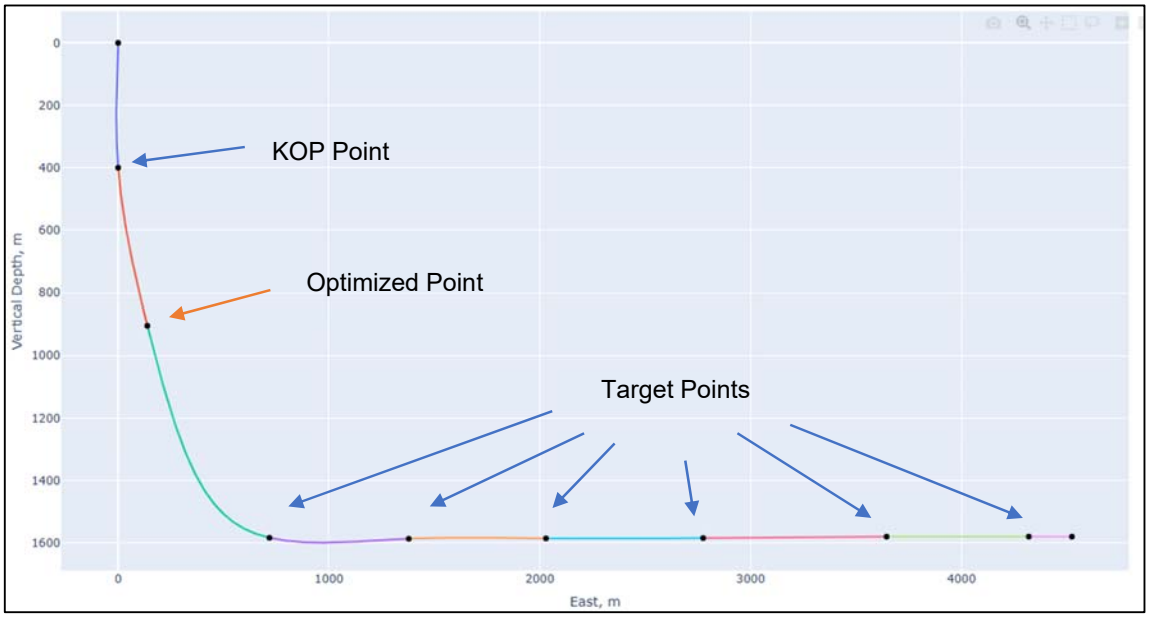

Figure 4.4 Optimized Trajectory Design Through Target Points

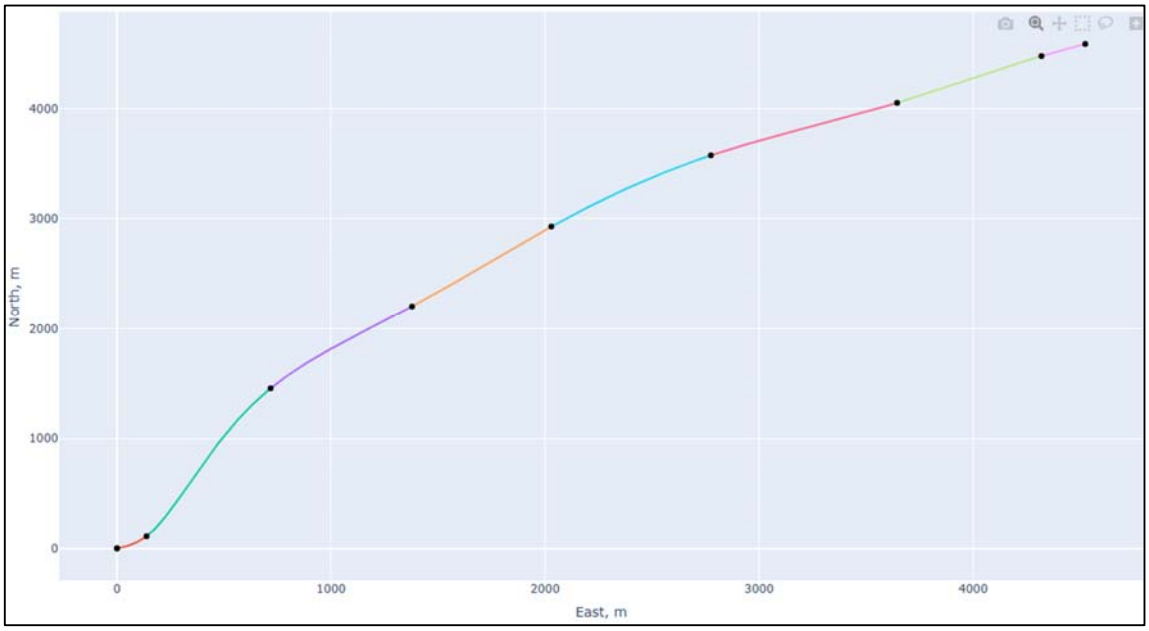

Figure 4.5 Top View of Generated Optimized Trajectory

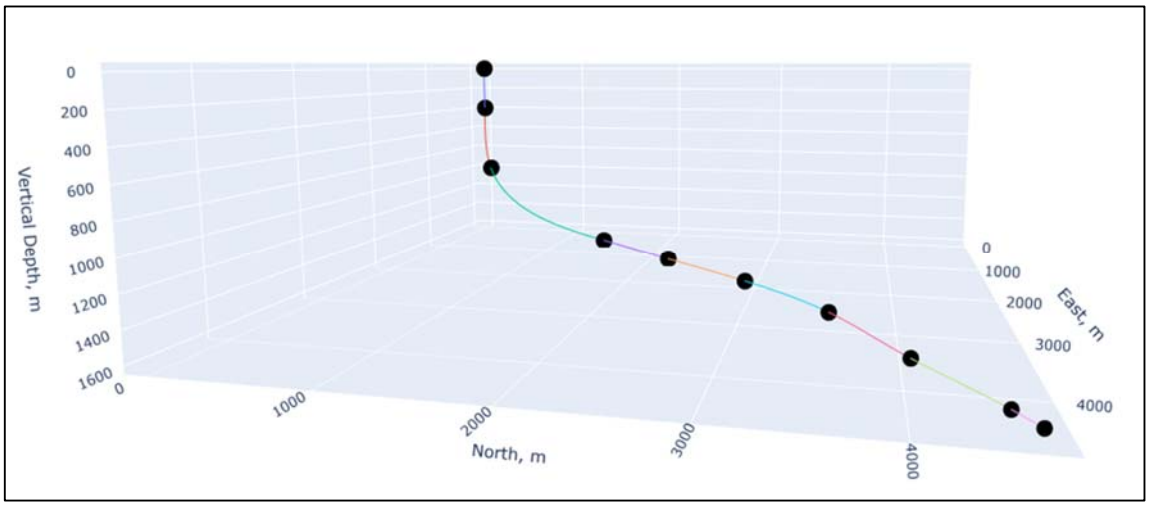

Figure 4.6 3D View of Generated Optimized Trajectory

The system successfully generated trajectories with DLS and makeup torque values within the specified limits.

Comparison of generated trajectory with original survey reveals that there is positive difference in properties of trajectory where optimization is allowed. For the trajectory section where all the points were given, the difference in results are minimal. The original survey in Table 4.1 shows that the maximum DLS is 3.5 (°/30m) whereas,

the optimized trajectory has 2.4 (°/30m) which is under the defined constraint of 2.5 (°/30m). The system also ensures that the trajectory follows a smooth and controlled path. Additionally, the algorithm calculated and plotted the necessary parameters such as azimuth, inclination, DLS, Torque, and Drag which are visualized in Figure 4.7, Figure 4.8, and Figure 4.9 respectively

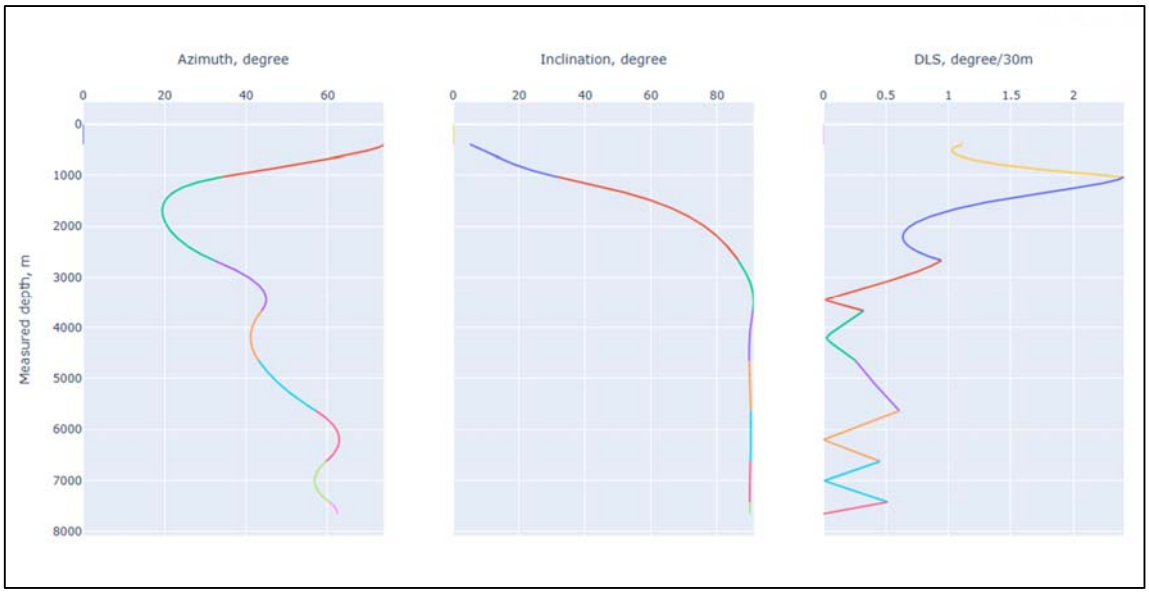

Figure 4.7 Azimuth, Inclination, and DLS Graph for Generated Trajectory

The generated trajectory is analyzed to assess three types of drag forces: tripping in, tripping out, and rotating off the bottom. These drag forces are significant factors that impact the drilling operation and must be carefully considered during well planning.

During the tripping process, when the drill string is being lowered into the wellbore, drag forces act in the opposite direction, opposing the downward movement of the string. Similarly, during the tripping-out process, when the drill string is removed from the wellbore, drag forces act in the direction of the upward movement. In addition to these forces, the system also evaluates the rotating off-bottom drag forces. These forces occur when the drill string is rotating while it is at the bottom but not in contact with the bottom of the wellbore. These forces are influenced by factors of trajectory, wellbore geometry, mud properties, and contact friction.

Drag forces are limited by the buckling limit of the drill string. To calculate and analyze the buckling limit, effective force is required to be calculated. Currently, this is not included in our system, and it will be included in under next development phase with the help of the ProWellPlan. This information empowers drilling engineers to make informed decisions, optimize drilling parameters, and enhance overall drilling performance.

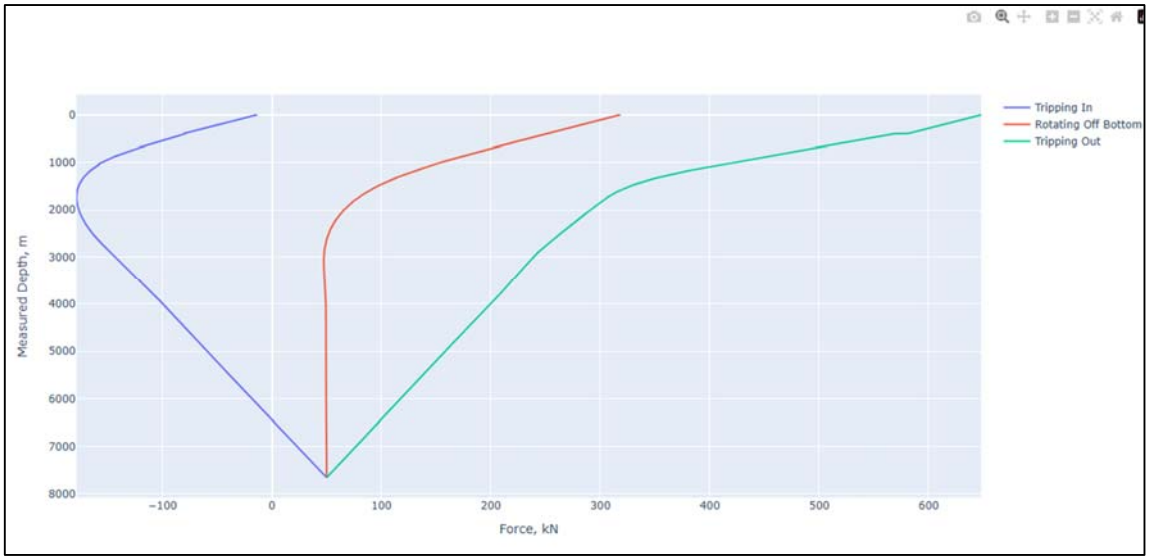

Figure 4.8 Drag Force Graph for Generated Trajectory.

Similarly, the generated trajectory is analyzed to evaluate the torque forces experienced during drilling operations. Torque refers to the rotational force exerted on the drill string while drilling. It is crucial to monitor and control torque to ensure the efficient and safe operation of the drilling equipment.

The system provides a comprehensive torque graph that depicts the variations in torque along the trajectory. Importantly, the torque values generated by the system are compared against the makeup torque limit. The makeup torque limit represents the maximum torque that the drill string can withstand without exceeding its operational limits. By ensuring that the torque values remain within the makeup torque limit, the system helps prevent potential issues such as drill string failures or excessive wear and tear.

The system's ability to generate torque values that are within the makeup torque limit assures the drilling equipment's integrity and reliability. This information allows drilling engineers to make informed decisions and optimize drilling parameters to ensure safe and efficient drilling operations.

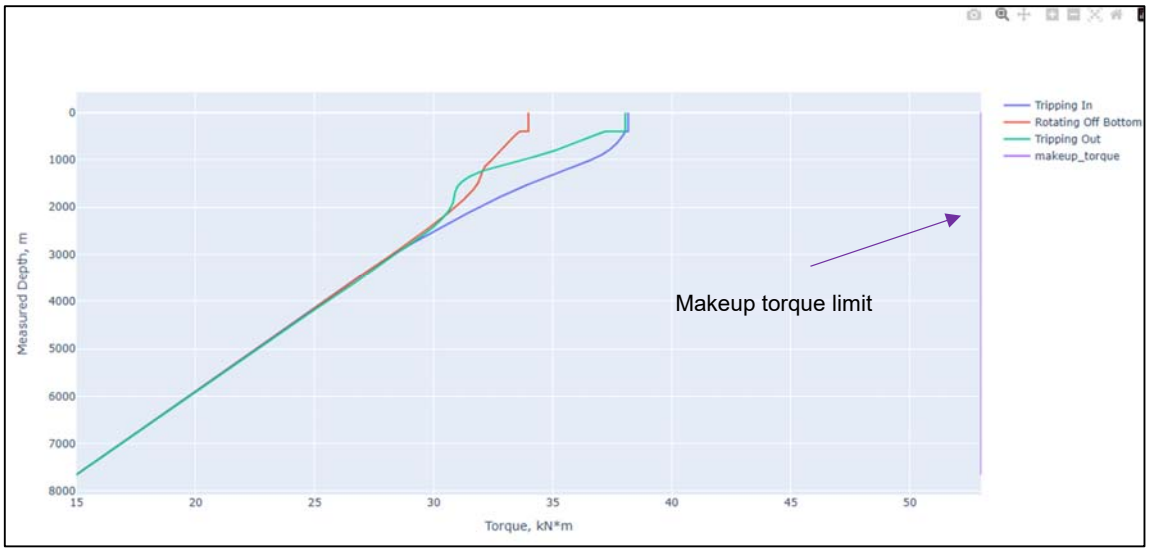

Figure 4.9 Torque Graph for Generated Trajectory

The results obtained from the case study showcase the capabilities of the autonomous trajectory design system in generating trajectories that meet the defined limitations. The system's ability to optimize the trajectory while considering torque and drag constraints ensures the safe and efficient drilling of extended-reach wells. Furthermore, the integration of the system with the torque and drag calculations allows for a comprehensive analysis of the drilling operation, ensuring the feasibility and viability of the generated trajectories.

4.1.5 Discussion

This case is just one of the simplest examples of the system’s capability. Only one optimization point between KOP and the first target point is used. Similarly, the process can be extended to multiple optimization points within each target point, or the optimization of target points itself. More engineering constraints can be added as well for complex autonomous trajectory design. The whole system is designed in such

a way that it is very easy to integrate more engineering models, like hydraulics, casing plans, and geological formations in it.

With more complex optimization problems, the computational power required would also be increased. But it can be solved by placing the system on the cloud-connected to powerful computers to be accessed by users.

4.2 Case Study on Optimization of KOP

In this case study, the focus is on optimizing the kick-off point of the well trajectory. The kick-off point refers to the location where the wellbore deviates from the vertical path and starts to build inclination. By strategically selecting the KOP, it is possible to achieve specific objectives such as reaching target formations, achieving minimum DLS, and generating a more benign trajectory under specified constraints.

Using the autonomous trajectory design system developed in this research, an additional algorithm is implemented with some changes to the previous one to iteratively modify the well trajectory and calculate the optimal KOP. The algorithm considers the multi objectives constraints, and optimization criteria to determine the most suitable KOP.

KOP optimization algorithm makes sure that the optimized point is always on origin with the possibility to move in the TVD direction. The objectives include minimizing MD, Torque, and Drag forces. The constraints were put on DLS and makeup torque.

4.2.1 Getting Survey Data from ProWellPlan

In ProWellPlan, wellbores were searched where KOP values were higher than 400 meters. One of the well named 33/9-A-15 CT4 was selected to work with. Survey data was downloaded, and target points were extracted out of it. Well trajectory for this well is shown in Figure 4.10

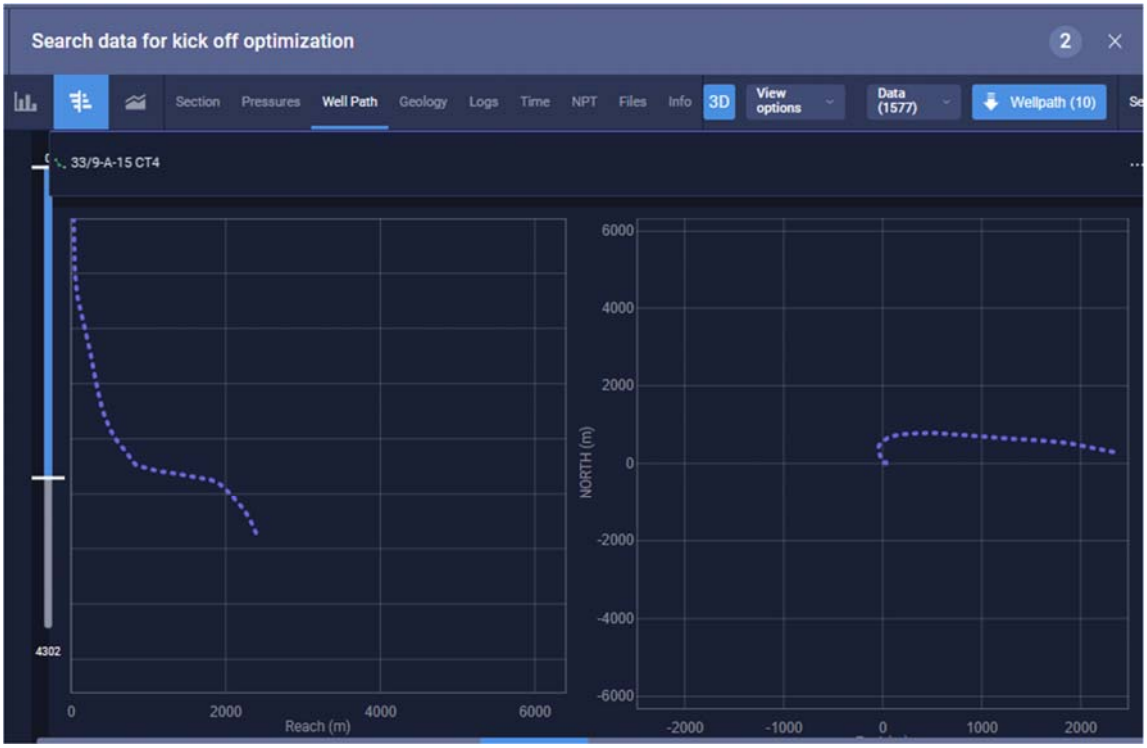

Figure 4.10 Well Trajectory for 33/9-A-15 CT4 from ProWellPlan

Survey data for this well is shown in Table 4.5

Table 4.5 Survey Points for 33/9-A-15 CT4 from ProWellPlan

MD(m)	TVD(m)	Inclination (°)	Azimuth (°)	Northing(m)	Easting(m)	DLS(°/30m)
0	0	0	0	0	0	0
200	199.94	1.4	27.65	3.66	2.75	0
400	399.49	5.07	240.77	8.67	-3.95	1.61
600	598.13	7.55	320.19	15.69	-24.35	3.55
800	791.88	19.03	351.81	61.96	-32.47	0.37
1000	980.6	19	350.08	127.25	-43.57	0.78
1200	1170.38	17.76	349.96	189.37	-54.54	0.42
1400	1362.46	15.62	355.32	244.75	-60.41	1.22
1600	1554.54	17.53	355.58	300.23	-64.84	1.26
1800	1743.19	24.13	357.77	366.2	-70.05	3.71
2000	1915.63	35.65	15.42	466.05	-59.69	2.87
2200	2055.33	53.23	26.53	595.98	-3.6	1.23
2400	2166.53	60.64	63.95	710.79	111.42	4.2
2600	2244.01	78.8	83.77	758.04	287.76	4.77
2800	2263.85	85.54	92.57	769	486.26	3.44
3000	2281.09	84.39	101.42	735.26	682.41	0.43
3200	2300.89	86.05	100.98	700.35	878.32	0.92

3400	2313.65	85.6	101.27	660.82	1073.95	0.31
3600	2331.61	83.99	100.3	622.98	1269.51	0.7
3800	2353.33	83.33	98.65	594.36	1466.23	0.79
4000	2374.15	81.64	102.72	557.46	1661.63	1.68
4200	2430.18	65.89	110.01	506.96	1846.01	2.73
4400	2529.28	57.63	117.92	432.8	2002.68	0.09
4600	2638.38	53.5	115.89	357.07	2152.14	1.83
4800	2775.49	38.99	116.79	292.73	2281.79	1.28
4945.28	2891.43	36.12	113.4	256.87	2361.6	0.47

Target points extracted from the well survey are shown in Table 4.6. KOP is defined as a variable and is selected for optimization.

Table 4.6 Target Points for KOP Optimization Case Study

	Northing	Easting	TVD
Surface Point	0	0	0
KOP Point	X_1	X_2	X_3
Target Point	15.69	-24.35	598.13
Target Point	127.25	-43.57	980.6
Target Point	244.75	-60.41	1362.46
Target Point	366.2	-70.05	1743.19
Target Point	595.98	-3.6	2055.33
Target Point	758.04	287.76	2244.01
Target Point	735.26	682.41	2281.09
Target Point	660.82	1073.95	2313.65
Target Point	594.36	1466.23	2353.33
Target Point	506.96	1846.01	2430.18
Target Point	357.07	2152.14	2638.38
Target Point	256.87	2361.6	2891.43

Values for KOP are determined after running the KOP optimization algorithm on the developed code.

4.2.2 Other Input Parameters

The same T&D and optimization parameters were used as in the previous case study for simplicity. Hole and Pipe properties are shown in Table 4.3. Other optimization parameters are shown in Table 4.4

4.2.3 Results

The results of the case study on the optimization of the kick-off point are presented through graphical representations, showcasing the trajectory properties and the impact of the optimization process. One of the key results is the graph displaying the trajectory with the optimized kick-off point in Figure 4.11.

The graph visually illustrates the well trajectory, in which the second point is KOP. Its coordinates in Northing, Easting, and TVD from after optimization are [0, 0, 304.33]

In addition to the graphical representation of the optimized trajectory, other graphs present the trajectory properties such as azimuth, inclination, DLS, torque, and drag in Figure 4.12, Figure 4.13, and Figure 4.14. These properties are crucial indicators of the wellbore's direction, angle, and curvature, which directly impact drilling operations.

By integrating the trajectory design and optimization process, the system provides a comprehensive approach to determining the optimal KOP. It takes into account the complex interplay between the well path, subsurface targets, and engineering constraints. The system evaluates multiple scenarios, iteratively adjusting the KOP and trajectory parameters to identify the configuration that best meets the defined objectives.

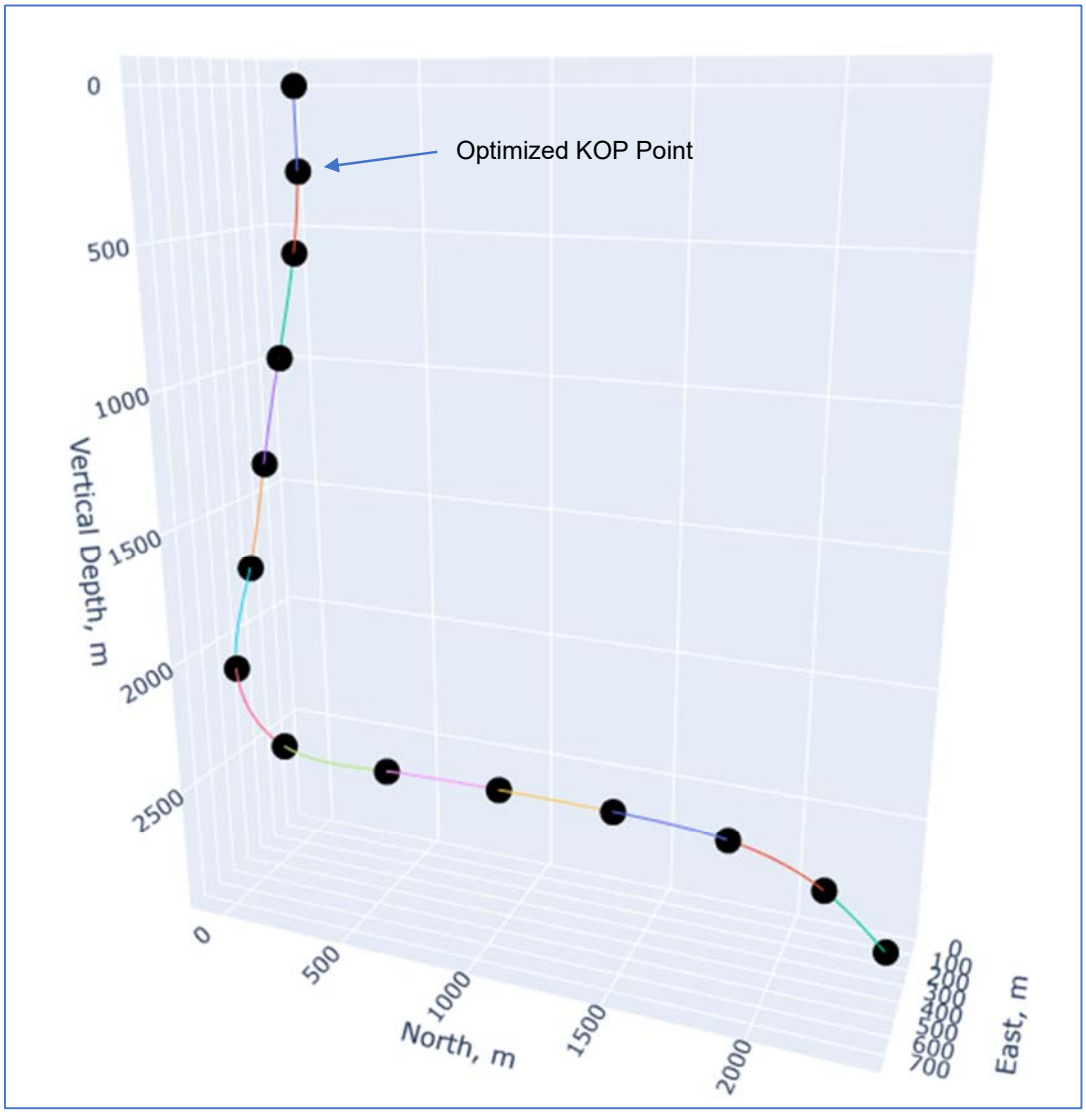

Figure 4.11. 3D-Generated Trajectory with Optimized KOP

The autonomous system successfully generates the coordinates of the Kick-Off Point for the trajectory as [0, 0, 304.33].

Comparison of generated trajectory with original survey reveals that the original KOP at 400 meters in the survey data has corresponding maximum Dogleg Severity value of 3.5 degrees/30 meters. However, as per the defined constraints, the DLS for the generated trajectory through our autonomous system is under 2 degrees/30 meters which is clear from Figure 4.12. All the other results through the target points which were not put for optimization show equivalent results to the original one.

By adhering to the specified DLS constraint, the autonomous system ensures that the trajectory maintains a smoother and more gradual change in direction. This can have significant implications for well planning and operational challenges in drilling.

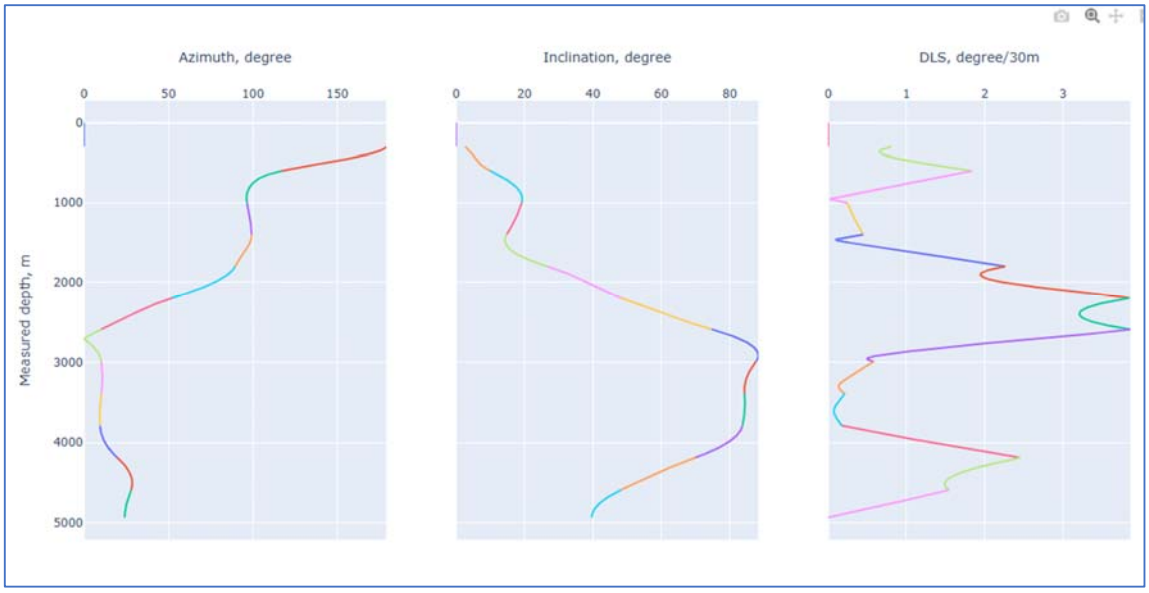

Figure 4.12 KOP-Optimized Trajectory properties

In Figure 4.12, DLS values at MD of almost 2000 meters to 3000 meters vary between 3 and 4 (deg/30m). This is because we did not allow any optimization beyond KOP. So, these values are similar to the original values in the survey.

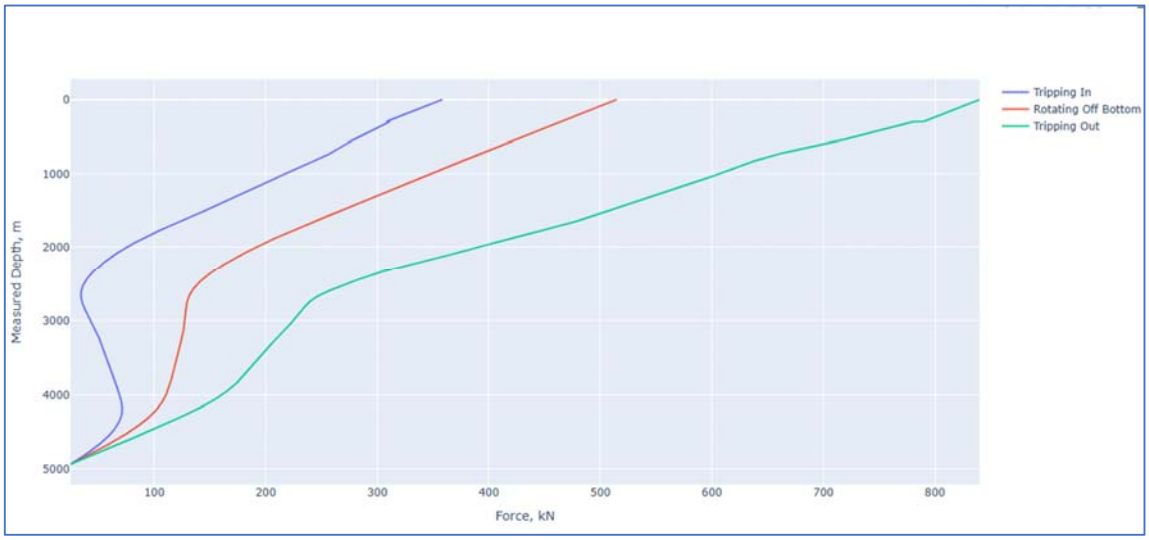

Figure 4.13. Force Values for KOP-Optimized Trajectory

The torque and drag forces generated for the trajectory are depicted in Figure 4.13 and Figure 4.14. These figures illustrate the magnitudes and variations of torque and drag forces along the wellbore.

The torque values start from 15 kN.m at the bottom as per torque at bit value defined initially and increase upwards towards the surface. Torque values are higher while tripping out. This upward motion creates an additional resistance due to which more torque is required to overcome this friction and rotate the drill string during tripping out. On the other hand, while tripping in, the drill string is being lowered into the wellbore. The downward motion helps reduce the friction between the drill string and the wellbore walls. With less friction to overcome, the torque required to rotate the drill string during tripping is comparatively smaller.

Also, torque values are under the defined torque limit. The same technique can be implemented for more than one point optimization problem. More optimization points will enable the system to generate trajectory within more space.

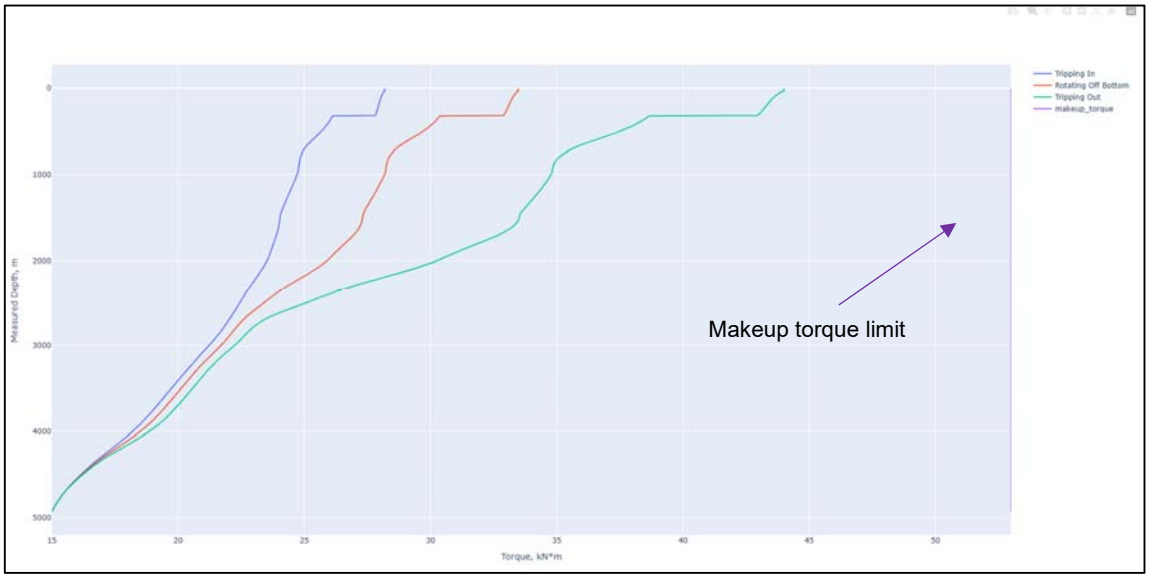

Figure 4.14. Torque Values for KOP-Optimized Trajectory

The results obtained from the case study demonstrate the successful optimization of the kick-off point. These results validate the effectiveness of the methodology in optimizing well trajectories.

4.2.4 Discussion

It is important to note that the results presented in the case study are obtained by assuming a simplified scenario where the drill string consists solely of drill pipes with the bit at the bottom. This simplification is made to focus specifically on the optimization problem and getting trajectory within the constraints of DLS and torque values.

It is essential to consider that in real drilling scenarios, the drill string composition may include additional components such as collars, stabilizers, or other BHA elements. The inclusion of these components can be done in our code with simple modifications.

Moreover, the single averaged friction factor is used in our case study, but for more complex cases where open holes and cased holes are clearly defined, friction factors can be put in the form of arrays with varying MD against each section.

The approach described in case studies can be extended to address **multiple-point optimization problems** in the form of a series of optimization and target points as shown in Table 4.7. By optimizing multiple points, the system can generate trajectories that cover a larger spatial area, allowing for improved well placement and drilling efficiency.

Table 4.7 Multi Point Optimization Problem

Point #	Coordinates (m)	Type
0	[0, 0, 0]	Given
1	[X_1, X_2, X_3]	Require Optimization
2	[15.69, -24.35, 598.13]	Given
3	[X_4, X_5, X_6]	Require Optimization
4	[X_7, X_8, X_9]	Require Optimization
5	[758.04, 287.76, 2244.01]	Given
6	[660.82, 1073.95, 2313.65]	Given

Chapter 5

Conclusion and Future Work

In this chapter, we discuss the applications along with the limitations of our work in addition to potential future research areas. We explore the various areas where this system can be applied, with its limitations and potential areas for improvement, and finally provide a comprehensive conclusion.

5.1 Conclusion

In conclusion, this thesis has presented an automated 3D trajectory design system that incorporates engineering calculations, specifically focusing on torque and drag analysis. The system addresses the limitations of traditional trajectory design processes by streamlining the optimization of drilling trajectories and considering critical constraints.

Through the development and implementation of the system, several key findings and outcomes have been achieved. Firstly, the automated trajectory design system successfully generates trajectories that adhere to user-defined constraints. It utilizes GA to efficiently search for the optimal trajectory within the defined constraints.

The system also integrates torque and drag calculations, providing valuable insights into the forces exerted on the drill string during drilling operations. By considering torque and drag constraints, the system ensures the drill string's ability to withstand these forces, reducing the risk of failures and improving overall drilling efficiency.

Furthermore, the case studies conducted in this research have demonstrated the effectiveness and applicability of the automated trajectory design system. The results show that the system successfully generates trajectories with torque and drag forces that are under the limits of the drill string.

While this research has made significant progress in automating trajectory design and incorporating torque and drag analysis, there are still areas for further exploration and improvement. The outcomes of this research contribute to the ongoing efforts of industry to enhance operational efficiency, reduce costs, and minimize risks.

5.2 Limitations

It is important to recognize that every system, no matter how advanced or sophisticated, has its limitations. This autonomous trajectory design system is no exception. While we have made significant strides in optimizing well trajectories and addressing torque and drag considerations, it is crucial to acknowledge the limitations within which our system operates. By understanding these limitations, we can better appreciate the system's strengths and areas where further development and refinement may be needed. Below are some of the limitations that are inherent to this system:

1. Torque and Drag Model Assumptions: The system relies on certain assumptions and simplifications to calculate torque and drag forces. One of them is assuming the soft string model for torque and drag forces. These assumptions might introduce minor errors between the simulated results and actual field conditions.

2. Computational Constraints: The optimization algorithms employed in the system have computational limitations, such as convergence speed and computational resources. Large-scale multiple optimization problems may require substantial computational resources and time.

3. Sequence optimization of target points: This system does not cater to the capability to optimize the sequence of the target points. It relies only on the sequence defined by the user. If the user-defined the position of the far-off target point is located before the near one, then trajectory will be generated through the far-off target point first. However, in an ideal drilling scenario, it should be able to detect and optimize the sequence of target points as well. This can be easily developed in future modifications.

4. Practical Validation: The system's performance and reliability should be validated against actual use in the industry to verify the various complex conditions. While simulations and case studies provide valuable insights, practical validation is necessary to ensure the system's accuracy and effectiveness in real-world drilling operations.

5.3 Future Work

The code developed can be used in multiple industry applications, revolutionizing well planning. By discussing these potential applications, we aim to demonstrate the versatility and practicality of our algorithm in addressing key industry challenges.

This system implemented the concept of automation in one of the foremost and primary tasks in well planning. Trajectory design with engineering calculations can be connected to offset data, cost calculation models, and risk analysis for comprehensive auto well plan generation. This system will be developed further with ProWellPlan to be implemented into the software by connecting it to other modules.

5.3.1 Integration of Other Models

To further enhance the trajectory design system and its applicability in drilling operations, the integration of additional engineering models and geological considerations holds great potential. One key area for future work is the integration of hydraulics models into the trajectory design process. By incorporating hydraulics calculations, the system can optimize drilling parameters such as mud flow rates and pressures, ensuring efficient cuttings transport and wellbore stability. This integration would enable engineers to generate trajectory designs that not only meet engineering constraints but also optimize drilling performance from a hydraulic perspective.

Another important aspect to consider is the integration of casing design plans into the trajectory design system. By incorporating casing design calculations, the system can recommend optimal casing sizes and placement along the wellbore trajectory, ensuring the integrity and stability of the wellbore. This integration would provide a

comprehensive approach to well planning, considering both drilling and casing considerations.

Furthermore, the incorporation of geological formation models would greatly enhance the accuracy and reliability of the trajectory design system. By integrating geological data, such as formation properties and boundaries, the system can generate trajectories that optimize wellbore placement within the target formations. This integration would allow for improved reservoir access and resource recovery, leading to more efficient drilling operations.

5.3.2 Recommendation of Drill String

The potential of the algorithm to reverse calculate and recommend the drill string properties and hole properties are also possible with some code modification. By connecting the algorithm with relevant data and optimization techniques, it can provide valuable insights into the selection and optimization of drilling equipment, leading to improved performance and cost-efficiency.

5.3.3 Recommendation of Operational Parameters

The algorithm developed in this study has the potential to provide recommendations for drilling operational parameters. By analyzing the data and applying optimization techniques, the algorithm can suggest optimal values for parameters such as weight on bit, rotary speed, drilling fluid properties, and other operational variables. These recommendations can help improve drilling efficiency and minimize the risk of operational issues. Implementing the algorithm's recommendations can lead to improved performance and cost-effectiveness in drilling operations. However, further research and refinement of the algorithm are necessary to ensure its accuracy and effectiveness in real-world drilling scenarios.

5.3.4 Relief Well Design

The relief well design is a critical aspect of well control operations, aimed at mitigating the impact of well blowouts or uncontrolled releases of fluids from a well. This algorithm can play a significant role in optimizing relief well design and improving the overall effectiveness of well control efforts.

By leveraging the capabilities of the algorithm, engineers can simulate various scenarios and analyze different relief well designs. The system can assist in determining the optimal trajectory for the relief well, considering constraints and objectives related to torque, drag, measured depth, and DLS.

Integration of geological data, hydraulics calculation, and other parameters will enable this algorithm to calculate and optimize the relief well trajectory more effectively. Engineers can incorporate advanced well control methodologies, computational fluid dynamics simulations, or dynamic kill modeling to enhance the accuracy and effectiveness of relief well designs. The algorithm serves as a platform for integrating additional models and updating the existing ones.

5.3.5 Live Trajectory Correction

This system can be modified for auto trajectory correctness considering limitations to torque and drag while using RSS. It can incorporate RSS live data to continuously monitor and correct the trajectory of the wellbore during drilling operations. This helps maintain the desired wellbore path and improves drilling efficiency.

The algorithm can be modified to incorporate real-time survey data instead of target points from monitoring systems to generate trajectories. These can be used along with original target points to generate newer drilling paths in 3D space. By continuously updating and adjusting the trajectory based on actual well conditions, the system enables real-time decision-making and adaptation to change circumstances. This dynamic approach helps navigate the bit from the optimized path.

References

[1] M. . Payne og F. . Abbassian, «Advanced Torque and Drag Considerations in Extended-Reach Wells,» *Distributed Computing,* vol. , nr. , p. , 1996.

[2] H. G. Suryadi, H. Li, L. Jiang og Q. Liu, «Automating Trajectory Design in a Cloud-Based Planning Solution,» 2022.

[3] J. Bourgoyne, K. K. Millheim, M. E. Chenevert og J. Young, Applied Drilling Engineering, Society of Petroleum Engineers.

[4] H. L. Taylor og M. C. Mason, «A Systematic Approach to Well Surveying Calculations,» *Society of Petroleum Engineers Journal,* vol. 12, pp. 474-488, December 1972.

[5] S. J. Sawaryn og J. L. Thorogood, «A Compendium of Directional Calculations Based on the Minimum Curvature Method,» *SPE Drilling & Completion,* vol. 20, pp. 24-36, March 2005.

[6] G. J. Wilson, «Radius Of Curvature Method For Computing Directional Surveys,» 1968.

[7] M. Gang, W. Xie, H. Luo, S. Chen, Y. Cui, Y. Su og R. Wang, «Simulation of directional well trajectory using tension spline,» *Journal of Physics: Conference Series,* vol. 1074, p. 012013, September 2018.

[8] J. H. B. Sampaio, «Designing Three-Dimensional Directional Well Trajectories Using Bézier Curves,» *Journal of Energy Resources Technology-transactions of The Asme,* vol. 139, nr. 3, p. 032901, 2017.

[9] R. Samuel, «A Compelling Case: Time to Change Minimum Curvature Survey Method for Well Engineering Calculations,» 2021.

[10] «Compass - Landmark Solution - E&P software,» Haliburton, [Internett]. Available: https://www.landmark.solutions/COMPASS-Directional-Well-Path-Planning. [Funnet June 2023].

[11] A. K. Sharma, G. . Yadava og S. . Deshmukh, «A literature review and future perspectives on maintenance optimization,» *Journal of Quality in Maintenance Engineering,* vol. 17, nr. 1, pp. 5-25, 2011.

[12] Y. . Jin og B. . Sendhoff, «A systems approach to evolutionary multiobjective structural optimization and beyond,» *IEEE Computational Intelligence Magazine,* vol. 4, nr. 3, pp. 62-76, 2009.

[13] H. . Parvin, B. . Minaei og S. . Ghatei, «A new particle swarm optimization for dynamic environments,» , 2011. [Internett]. Available: https://link.springer.com/chapter/10.1007/978-3-642-21323-6_37. [Funnet 13 6 2023].

[14] Z. . Beheshti og S. M. Shamsuddin, «CAPSO: Centripetal accelerated particle swarm optimization,» *Information Sciences,* vol. 258, nr. , pp. 54-79, 2014.

[15] D. . Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, red., vol. , , : Addison-Wesley Professional, 1989, p. 41.

[16] G. . Li, K.-H. . Lee og K.-S. . Leung, «Genetic algorithm based on independent component analysis for global optimization,» *Lecture Notes in Computer Science,* vol. , nr. , pp. 172-181, 2006.

[17] A. . Konak, D. W. Coit og A. E. Smith, «Multi-objective optimization using genetic algorithms: A tutorial,» *Reliability Engineering & System Safety,* vol. 91, nr. 9, pp. 992-1007, 2006.

[18] J. Cao og D. Sui, «Well Path Design and Optimization Using Composite Cubic Bezier Curves,» *SPE Journal,* vol. 27, pp. 3254-3270, December 2022.

[19] C. A. Johancsik, D. B. Friesen og R. Dawson, «Torque and Drag in Directional Wells-Prediction and Measurement,» *Journal of Petroleum Technology,* vol. 36, pp. 987-992, June 1984.

[20] R. F. Mitchell og R. Samuel, «How Good Is the Torque/Drag Model?,» *SPE Drilling & Completion,* vol. 24, pp. 62-71, March 2009.

[21] T. . Sherratt, «Sketching with Python and Plotly,» , 2014. [Internett]. Available: http://discontents.com.au/sketching-with-python-and-plotly. [Funnet 13 6 2023].

[22] «NumPy Homepage,» , . [Internett]. Available: http://www.numpy.org/. [Funnet 13 6 2023].

[23] «SciPy Stack,» , . [Internett]. Available: http://www.scipy.org/install.html. [Funnet 13 6 2023].

[24] «Python Data Analysis Library – pandas: Python Data Analysis Library,» , . [Internett]. Available: https://pandas.pydata.org/. [Funnet 13 6 2023].

[25] «Dash by Plotly,» , . [Internett]. Available: https://plot.ly/products/dash/. [Funnet 13 6 2023].

[26] X. Liu og R. Samuel, «Catenary Well Profiles for Extended and Ultra-Extended Reach Wells,» 2009.

[27] T. . Aarrestad og H. . Blikra, «Torque and Drag-Two Factors in Extended-Reach Drilling,» *Journal of Petroleum Technology,* vol. 46, nr. 09, pp. 800-803, 1994.

[28] G. . Guild, T. . Hill og . . Summers, «Designing and drilling extended reach wells. Part 1,» *Petroleum engineer international,* vol. , nr. , p. , 1994.

[29] B. . Isa og I. . Handono, «Extended Reach Drilling Experience at Ujung Pangkah Field,» , 2014. [Internett]. Available: http://archives.datapages.com/data/ipa_pdf/2014/ipa14-e-104.pdf. [Funnet 13 6 2023].

[30] A. H. Strømhaug, «Directional Drilling - Advanced Trajectory Modelling,» , 2014. [Internett]. Available: http://diva-portal.org/smash/record.jsf?pid=diva2:746157. [Funnet 12 6 2023].

Appendix

main.py

```
from Optimization_problem import OptimizationProblem
import numpy as np
from gn_bezier_fit import GnBezierFit
from evaluate_bezier_struct3 import EvaluateBezierStruct3
from sko.GA import GA


# Input parameters

P = np.array([[0, 0, 0],
              [0, 0, 400],
              [0, 0, 0],
              [717.85, 1455.52, 1583.29],
              [1378.68, 2203.1, 1586.35],
              [2029.57, 2928.95, 1585.09],
              [2775.25, 3574.4, 1584.18],
              [3645.47, 4050.68, 1579.62],
              [4320.26, 4474.46, 1579.43],
              [4524.58, 4584.3, 1579.5]])

DLS_max = 2

nt = 100
dimensions = {'pipe': {'od': 5.5, 'id': 5, 'bottom':
EvaluateBezierStruct3(GnBezierFit(P.T, 2, 1), nt)[-1]['Lt'][-1][-1],
'makeup_torque': 53}, 'odAnn': 8.5}
wob = 50
tbit = 15
fric = 0.24
densities = {'rhof': 1.3, 'rhod': 7.8} # rhof = fluid density, rhod = drill
string density

optimize_this_point = 2  # starting from 0

# Optimized Trajectory Plot

lb = P[optimize_this_point - 1, :]
ub = P[optimize_this_point + 1, :]

problem = OptimizationProblem(P, nt, optimize_this_point, DLS_max,
dimensions, wob, tbit, fric, densities)
constrain_DLS = {'type': 'ineq', 'fun': problem.constrain_DLS}
ga = GA(func=problem.combined_objective, n_dim=3, size_pop=100, lb=lb,
ub=ub, max_iter=100,
        constraint_eq=[problem.constrain_DLS,
problem.constrain_makeup_torque])
best_x, best_y = ga.run()
print('coordinates of optimized point', best_x, '\n', 'value of combined
function of T&D and MD:', best_y)
problem.plot_bezier(best_x)
```

kop_optimization.py

```
from Optimization_problem import OptimizationProblem
import numpy as np
from gn_bezier_fit import GnBezierFit
from evaluate_bezier_struct3 import EvaluateBezierStruct3
from sko.GA import GA

# Input parameters

P = np.array([[0, 0, 0],
              [8.67, -3.95, 399.49],
              [15.69, -24.35, 598.13],
              [127.25, -43.57, 980.6],
              [244.75, -60.41, 1362.46],
              [366.2, -70.05, 1743.19],
              [595.98, -3.6, 2055.33],
              [758.04, 287.76, 2244.01],
              [735.26, 682.41, 2281.09],
              [660.82, 1073.95, 2313.65],
              [594.36, 1466.23, 2353.33],
              [506.96, 1846.01, 2430.18],
              [357.07, 2152.14, 2638.38],
              [256.87, 2361.6, 2891.43]
              ])

DLS_max = 3

nt = 100
dimensions = {
    'pipe': {'od': 5.5, 'id': 5, 'bottom':
EvaluateBezierStruct3(GnBezierFit(P.T, 2, 1), nt)[-1]['Lt'][-1][-1],
             'makeup_torque': 53}, 'odAnn': 8.5}
wob = 50
tbit = 15
fric = 0.24
densities = {'rhof': 1.3, 'rhod': 7.8}  # rhof = fluid density, rhod =
drill string density

optimize_this_point = 1  # starting from 0

# Optimized Trajectory Plot

lb = [0, 0, P[optimize_this_point-1, 2]]
ub = [1, 1, P[optimize_this_point + 1, 2]]

problem = OptimizationProblem(P, nt, optimize_this_point, DLS_max,
dimensions, wob, tbit, fric, densities)
constrain_DLS = {'type': 'ineq', 'fun': problem.constrain_DLS}
ga = GA(func=problem.combined_objective, n_dim=3, size_pop=100, lb=lb,
ub=ub, max_iter=100,
        constraint_eq=[problem.constrain_DLS,
problem.constrain_makeup_torque])
best_x, best_y = ga.run()
best_x = [0, 0, best_x[2]]
print('coordinates of optimized point', best_x, '\n', 'value of combined
function of T&D and MD:', best_y)
problem.plot_bezier(best_x)
```

gn_bezier_fit.py

Contact author for accessing this file.

pascal.py

Contact author for accessing this file.

bezier_eval3.py

Contact author for accessing this file.

evaluate_bezier_struct3.py

```
import numpy as np
from bezier_eval3 import BezierEval3

def EvaluateBezierStruct3(Bez, nt):

    for ii in range(len(Bez)):

        Q = Bez[ii]['Q']
        if 'w' in Bez[ii]:
            w = Bez[ii]['w']
        else:
            w = np.ones((1, Q.shape[1]))

        # define 'time' vectors for each segment
        t = np.linspace(0, 1, nt)
        X, theta, alpha, curvature, L = BezierEval3(Q, t)
        Bez[ii]['n'] = t
        Bez[ii]['X'] = X  # curve values for segment
        Bez[ii]['theta'] = theta
        Bez[ii]['C'] = curvature
        Bez[ii]['L'] = L
        Bez[ii]['alpha'] = alpha
        if ii == 0:
            Bez[ii]['Lt'] = L
        else:
            Bez[ii]['Lt'] = L + Bez[ii-1]['Lt'][0, -1]

    return Bez
```

Optimization_problem.py

```
import numpy as np
from gn_bezier_fit import GnBezierFit, GnBezierFit1
from evaluate_bezier_struct3 import EvaluateBezierStruct3
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import pandas as pd
import well_profile as wp
```

```
import torque_drag as td


class OptimizationProblem:
    def __init__(self, P, nt, optimize_this_point, DLS_max, dimensions,
wob, tbit, fric, densities):

        self.DLS_max = DLS_max
        self.optimize_this_point = optimize_this_point
        self.P = P
        self.nt = nt
        self.dimensions = dimensions
        self.wob = wob
        self.tbit = tbit
        self.fric = fric
        self.densities = densities

    def combined_objective(self, x):
        P = self.P
        optimize_this_point = self.optimize_this_point
        P[optimize_this_point, :] = x

        CBZ = GnBezierFit(P.T, 2, 1)
        CBZ = EvaluateBezierStruct3(CBZ, self.nt)

        # Calculate objective MD
        MD_t = np.zeros((self.nt, len(CBZ)))
        for i in range(len(CBZ)):
            MD_t[:, i] = CBZ[i]['Lt']
        MD = MD_t[-1, -1]

        # Calculate objective T and D

        inclination = []  # Theta values
        azimuth = []  # Alpha values
        md = []

        for i in range(len(CBZ)):
            for inc in range(len(CBZ[i]['theta'][0]) - 1):
                if CBZ[i]['theta'][0][inc] * 180 / np.pi < 360:
                    inclination.append(CBZ[i]['theta'][0][inc] * 180 /
np.pi)
                else:
                    inclination.append(CBZ[i]['theta'][0][inc] * 180 /
np.pi - 360)

            for azi in range(len(CBZ[i]['alpha'][0]) - 1):
                azimuth.append(CBZ[i]['alpha'][0][azi] * 180 / np.pi)

            for depths in range(len(CBZ[i]['Lt'][0]) - 1):
                md.append(CBZ[i]['Lt'][0][depths])

        data_dict = {
            'md': md,
            'inc': inclination,
            'azi': azimuth

        }

        data = pd.DataFrame(data_dict)
        # data.to_excel('output.xlsx', index=False)
```

```
        well = wp.load(data)

        result = td.calc(well.trajectory, dimensions=self.dimensions,
densities=self.densities, case='all', fric=self.fric, torque_calc=True,
wob=self.wob, tbit=self.tbit)
        T = result.torque['hoisting'][0]
        D = result.force['hoisting'][0]

        combined_objective = 0.3 * T + 0.3 * D + 0.4 * MD
        return combined_objective

    def constrain_DLS(self, x):
        P = self.P
        optimize_this_point = self.optimize_this_point

        P[optimize_this_point, :] = x

        CBZ = GnBezierFit(P.T, 2, 1)
        CBZ = EvaluateBezierStruct3(CBZ, self.nt)
        Curv = np.zeros((self.nt, len(CBZ)))
        DLS = np.zeros(len(CBZ))
        for i in range(len(CBZ)):
            Curv[:, i] = CBZ[i]['C']
            DLS[i] = np.max(Curv[:, i] * 30 * 180 / np.pi)
        DLS = np.max(DLS)
        DLS = np.abs(self.DLS_max - DLS)
        return DLS

    def constrain_makeup_torque(self, x):
        P = self.P
        optimize_this_point = self.optimize_this_point

        P[optimize_this_point, :] = x

        CBZ = GnBezierFit(P.T, 2, 1)
        CBZ = EvaluateBezierStruct3(CBZ, self.nt)

        inclination = []  # Theta values
        azimuth = []  # Alpha values
        md = []

        for i in range(len(CBZ)):
            for inc in range(len(CBZ[i]['theta'][0]) - 1):
                if CBZ[i]['theta'][0][inc] * 180 / np.pi < 360:
                    inclination.append(CBZ[i]['theta'][0][inc] * 180 /
np.pi)
                else:
                    inclination.append(CBZ[i]['theta'][0][inc] * 180 /
np.pi - 360)

            for azi in range(len(CBZ[i]['alpha'][0]) - 1):
                azimuth.append(CBZ[i]['alpha'][0][azi] * 180 / np.pi)

            for depths in range(len(CBZ[i]['Lt'][0]) - 1):
                md.append(CBZ[i]['Lt'][0][depths])

        data_dict = {
            'md': md,
            'inc': inclination,
            'azi': azimuth
```

```
        }

        data = pd.DataFrame(data_dict)
        # data.to_excel('output.xlsx', index=False)
        well = wp.load(data)

        result = td.calc(well.trajectory, densities=self.densities,
dimensions=self.dimensions, case='all', fric=self.fric, torque_calc=True,
wob=self.wob, tbit=self.tbit)
        T = result.torque['hoisting'][0]
        T = T - result.torque['makeup_torque'][0]
        return T

    def plot_bezier(self, x):
        P = self.P
        optimize_this_point = self.optimize_this_point

        P[optimize_this_point, :] = x

        CBZ = GnBezierFit1(P.T, 2, 1)
        CBZ = EvaluateBezierStruct3(CBZ, self.nt)
        m = len(CBZ)
        PC = np.zeros((3, 4, m))
        Coord = np.zeros((3, self.nt, m))
        Theta = np.zeros((self.nt, m))
        Alpha = np.zeros((self.nt, m))
        MD_t = np.zeros((self.nt, m))
        Curv = np.zeros((self.nt, m))
        for i in range(m):
            PC[:, :, i] = CBZ[i]['Q']
            Coord[:, :, i] = CBZ[i]['X']
            Theta[:, i] = CBZ[i]['theta']
            Alpha[:, i] = CBZ[i]['alpha']
            MD_t[:, i] = CBZ[i]['Lt']
            Curv[:, i] = CBZ[i]['C']
        DLS = np.abs(Curv) * 30 * 180 / np.pi

        # Plotting code goes here
        # Define the layout of the 3D scatter plot with a reversed Z-axis
        layout = go.Layout(scene=dict(xaxis=dict(title='East, m'),
                                      yaxis=dict(title='North, m'),
                                      zaxis=dict(title='Vertical Depth, m',
autorange='reversed')),
                           margin=dict(l=0, r=0, b=0, t=0))

        # Create the traces for the curve and the scatter plot
        traces = []
        for i in range(m):
            trace = go.Scatter3d(x=Coord[0, :, i], y=Coord[1, :, i],
z=Coord[2, :, i], mode='lines', line=dict(width=2))
            traces.append(trace)

        scatter_trace = go.Scatter3d(x=P[:, 0], y=P[:, 1], z=P[:, 2],
mode='markers', marker=dict(color='black'))

        # Create the figure with the layout and the traces
        fig = go.Figure(layout=layout, data=traces + [scatter_trace])

        # Show the figure
        fig.show()
```

```
        # Define the layout of the first 2D scatter plot (North and East)
        layout1 = go.Layout(xaxis=dict(title='East, m'),
                            yaxis=dict(title='North, m'),
                            margin=dict(l=0, r=0, b=0, t=0))

        # Define the layout of the second 2D scatter plot (East and TVD)
        layout2 = go.Layout(xaxis=dict(title='East, m'),
                            yaxis=dict(title='Vertical Depth, m',
autorange='reversed'),
                            margin=dict(l=0, r=0, b=0, t=0))

        # Create the traces for the first 2D scatter plot (North and East)
        traces1 = []
        for i in range(m):
            trace = go.Scatter(x=Coord[0, :, i], y=Coord[1, :, i],
mode='lines', line=dict(width=2))
            traces1.append(trace)

        scatter_trace1 = go.Scatter(x=P[:, 0], y=P[:, 1], mode='markers',
marker=dict(color='black'))

        # Create the figure with the first layout and the first set of
traces
        fig1 = go.Figure(layout=layout1, data=traces1 + [scatter_trace1])

        # Create the traces for the second 2D scatter plot (East and TVD)
        traces2 = []
        for i in range(m):
            trace = go.Scatter(x=Coord[0, :, i], y=Coord[2, :, i],
mode='lines', line=dict(width=2))
            traces2.append(trace)

        scatter_trace2 = go.Scatter(x=P[:, 0], y=P[:, 2], mode='markers',
marker=dict(color='black'))

        # Create the figure with the second layout and the second set of
traces
        fig2 = go.Figure(layout=layout2, data=traces2 + [scatter_trace2])

        # Show the figures
        fig1.show()
        fig2.show()

        # Convert alpha to degree
        Alpha = Alpha * 180 / np.pi
        Alpha[Alpha > 360] = Alpha[Alpha > 360] - 360

        fig = make_subplots(rows=1, cols=3, shared_yaxes=True)

        # Create subplot 1 with reversed y-axis and azimuth on x-axis at
top
        for i in range(0, m):
            fig.add_trace(
                go.Scatter(x=np.abs(Alpha[:, i]), y=MD_t[:, i], name='Curve
1', mode='lines', line=dict(width=2)),
                row=1, col=1)
        fig.update_layout(xaxis_title='Azimuth, degree',
yaxis_title='Measured depth, m', xaxis=dict(range=[0, 360]),
                          yaxis=dict(autorange='reversed'),
xaxis_side='top', boxmode='overlay')
```

```
        # Create subplot 2
        for i in range(0, m):
            fig.add_trace(
                go.Scatter(y=MD_t[:, i], x=Theta[:, i] * 180 / np.pi,
name=f'Curve {i + 1}', mode='lines',
                           line=dict(width=2)),
                row=1, col=2)
        fig.update_layout(yaxis_title='Measured depth, m',
xaxis=dict(autorange=True), yaxis=dict(autorange='reversed'),
                          boxmode='overlay')
        fig.update_xaxes(title_text='Inclination, degree', row=1, col=2,
side='top')

        # Create subplot 3
        for i in range(0, m):
            fig.add_trace(
                go.Scatter(y=MD_t[:, i], x=DLS[:, i], name=f'Curve {i +
1}', mode='lines', line=dict(width=2)),
                row=1, col=3)
        fig.update_layout(yaxis_title='Measured depth, m',
xaxis=dict(autorange=True), yaxis=dict(autorange='reversed'),
                          boxmode='overlay')
        fig.update_xaxes(title_text='DLS, degree/30m', row=1, col=3,
side='top')

        fig.show()

        inclination = []  # Theta values
        azimuth = []  # Alpha values
        md = []

        for i in range(m):
            for inc in range(len(CBZ[i]['theta'][0]) - 1):
                if CBZ[i]['theta'][0][inc] * 180 / np.pi < 360:
                    inclination.append(CBZ[i]['theta'][0][inc] * 180 /
np.pi)
                else:
                    inclination.append(CBZ[i]['theta'][0][inc] * 180 /
np.pi - 360)

            for azi in range(len(CBZ[i]['alpha'][0]) - 1):
                azimuth.append(CBZ[i]['alpha'][0][azi] * 180 / np.pi)

            for depths in range(len(CBZ[i]['Lt'][0]) - 1):
                md.append(CBZ[i]['Lt'][0][depths])

        data_dict = {
            'md': md,
            'inc': inclination,
            'azi': azimuth

        }

        data = pd.DataFrame(data_dict)
        # data.to_excel('output.xlsx', index=False)
        well = wp.load(data)

        result = td.calc(well.trajectory, dimensions=self.dimensions,
densities=self.densities, case='all', fric=self.fric, torque_calc=True,
wob=self.wob, tbit=self.tbit)
```

```
        result.plot(plot_case='Force').show()
        result.plot(plot_case='Torque').show()
```

Modified Torque and Drag Package

main.py

```
from math import pi, sin, cos, radians


def calc(trajectory, dimensions, densities=None, case="all", fric=None,
wob=0, tbit=0, torque_calc=False):
    """
    Function to generate the torque and drag profiles. Model Source: SPE-
11380-PA

    Arguments:
        trajectory: a trajectory object from well_profile library
        dimensions: dict for dimensions {'pipe': {'od', 'id', 'length',
'bottom', 'makeup_torque'},
                                          'odAnn'}   # Makeup torque in kN*m
        densities: dict for densities {'rhof': 1.3, 'rhod': 7.8} # density
of fluid and density of drill string
        case: "lowering", "static", "hoisting" or "all"
        fric: num or list. sliding friction coefficient between DP-
wellbore. default: 0.24
        tbit: torque on bit, kN*m
        wob: weight on bit, kN
        torque_calc: boolean, include torque calculation

    Returns:
        object with drag force and torque in kN and kN*m
    """

    well = set_conditions(trajectory, dimensions, densities, wob, tbit)

    unit_pipe_weight = well.rhod * 9.81 * pi * (well.pipe_or ** 2 -
well.pipe_ir ** 2)
    area_a = pi * ((well.ann_or ** 2) - (well.pipe_or ** 2))     # annular
area in m2
    area_ds = pi * (well.pipe_ir ** 2)       # drill string inner area in
m2

    for point in well.trajectory:
        point['buoyancy'] = 1 - ((point['rhof'] * area_a) - (point['rhof']
* area_ds)) / (well.rhod * (area_a-area_ds))
        point['weight'] = unit_pipe_weight * point['delta']['md'] *
point['buoyancy']
        point['makeup_torque'] = well.makeup_torque

    if type(fric) is not list:
        for point in well.trajectory:
            point['fric'] = 0.24
    else:
        for idx, point in enumerate(well.trajectory):
            point['fric'] = fric[idx]
```

```
    well.trajectory[-1]['force'] = {'lowering': well.wob,
                                    'static': well.wob,
                                    'hoisting': well.wob}
    well.trajectory[-1]['torque'] = {'lowering': None,
                                     'static': None,
                                     'hoisting': None,
                                     'makeup_torque': None}

    if torque_calc:
        well.trajectory[-1]['torque'] = {'lowering': well.tbit,
                                         'static': well.tbit,
                                         'hoisting': well.tbit,
                                         'makeup_torque':
well.makeup_torque}

    for idx, point in reversed(list(enumerate(well.trajectory[:-1]))):
        point['incAvg'] = radians((point['inc'] + well.trajectory[idx-
1]['inc']) / 2)
        delta_azi = -radians(well.trajectory[idx+1]['delta']['azi'])
        delta_inc = -radians(well.trajectory[idx+1]['delta']['inc'])
        point['force'] = {}
        point['torque'] = {}
        # DRAG FORCE CALCULATIONS
        if (case == "lowering") or (case == "all"):
            # Drag force
            fn_1 = ((well.trajectory[idx+1]['force']['lowering'] *
delta_azi * sin(point['incAvg'])) ** 2 +
                    (well.trajectory[idx+1]['force']['lowering'] *
delta_inc + point['weight'] *
                     sin(point['incAvg'])) ** 2) ** 0.5

            delta_ft_1 = point['weight'] * cos(point['incAvg']) -
point['fric'] * fn_1
            point['force']['lowering'] =
well.trajectory[idx+1]['force']['lowering'] + delta_ft_1

            if torque_calc:
                # Torque calculation
                delta_torque_1 = point['fric'] * fn_1 * well.pipe_or
                point['torque']['lowering'] =
well.trajectory[idx+1]['torque']['lowering'] + delta_torque_1

        if (case == "static") or (case == "all"):

            # Drag force
            fn_2 = ((well.trajectory[idx+1]['force']['static'] * delta_azi
* sin(point['incAvg'])) ** 2 +
                    (well.trajectory[idx+1]['force']['static'] * delta_inc
+ point['weight'] *
                     sin(point['incAvg'])) ** 2) ** 0.5

            delta_ft_2 = point['weight'] * cos(point['incAvg'])
            point['force']['static'] =
well.trajectory[idx+1]['force']['static'] + delta_ft_2

            if torque_calc:
                # Torque calculation
                delta_torque_2 = point['fric'] * fn_2 * well.pipe_or
                point['torque']['static'] =
well.trajectory[idx+1]['torque']['static'] + delta_torque_2
```

```
        if (case == "hoisting") or (case == "all"):

            # Drag force
            fn_3 = ((well.trajectory[idx+1]['force']['hoisting'] *
delta_azi * sin(point['incAvg'])) ** 2 +
                    (well.trajectory[idx+1]['force']['hoisting'] *
delta_inc + point['weight'] *
                     sin(point['incAvg'])) ** 2) ** 0.5

            delta_ft_3 = point['weight'] * cos(point['incAvg']) +
point['fric'] * fn_3
            point['force']['hoisting'] =
well.trajectory[idx+1]['force']['hoisting'] + delta_ft_3

            if torque_calc:
                # Torque calculation
                delta_torque_3 = point['fric'] * fn_3 * well.pipe_or
                point['torque']['hoisting'] =
well.trajectory[idx+1]['torque']['hoisting'] + delta_torque_3

    class TaD(object):
        def __init__(self):
            self.force = {
                "lowering": [],
                "static": [],
                "hoisting": []
            }
            self.torque = {
                "lowering": [],
                "static": [],
                "hoisting": [],
                "makeup_torque": []
            }
            self.depth = []
            self.trajectory = well.trajectory
            self.makeup_torque = well.makeup_torque

            for point in well.trajectory:
                self.depth.append(point['md'])
                if (case == "lowering") or (case == "all"):

self.force['lowering'].append(point['force']['lowering'] / 1000)
                    if torque_calc:

self.torque['lowering'].append(point['torque']['lowering'] / 1000)

self.torque['makeup_torque'].append(point['torque'].get('makeup_torque',
well.makeup_torque) / 1000)

                if (case == "static") or (case == "all"):
                    self.force['static'].append(point['force']['static'] /
1000)
                    if torque_calc:

self.torque['static'].append(point['torque']['static'] / 1000)

self.torque['makeup_torque'].append(point['torque'].get('makeup_torque',
well.makeup_torque) / 1000)
                if (case == "hoisting") or (case == "all"):
```

```
self.force['hoisting'].append(point['force']['hoisting'] / 1000)
                    if torque_calc:

self.torque['hoisting'].append(point['torque']['hoisting'] / 1000)

self.torque['makeup_torque'].append(point['torque'].get('makeup_torque',
well.makeup_torque) / 1000)

        def plot(self, plot_case='Force'):
            from .plot import tnd
            fig = tnd(self, plot_case)

            return fig

    return TaD()


def set_conditions(trajectory, dimensions, densities=None, wob=0, tbit=0):

    wob *= 1000
    tbit *= 1000

    if densities is None:
        densities = {'rhof': 1.3, 'rhod': 7.8}

    class NewWell(object):
        def __init__(self):
            self.bottom = dimensions['pipe']['bottom']
            self.makeup_torque = dimensions['pipe']['makeup_torque'] * 1000
# convert to Nm
            self.pipe_length = dimensions['pipe'].get('length',
self.bottom)
            self.top = self.bottom - self.pipe_length
            self.trajectory = [x for x in trajectory if self.top <= x['md']
<= self.bottom]
            self.pipe_ir = dimensions['pipe']['id'] / 2 / 39.37
            self.pipe_or = dimensions['pipe']['od'] / 2 / 39.37
            self.ann_or = dimensions['odAnn'] / 2 / 39.37
            self.rhod = densities['rhod'] * 1000

            if type(densities['rhof']) is not list:
                for point in self.trajectory:
                    point['rhof'] = densities['rhof'] * 1000   # in kg/m3
            else:
                for idx, point in enumerate(self.trajectory):
                    point['rhof'] = densities['rhof'][idx] * 1000   # in
kg/m3

            self.wob = wob      # in N
            self.tbit = tbit        # in Nm

    return NewWell()
```

plot.py

```
import plotly.graph_objects as go
```

```
def tnd(tq_n_dg, plot_case='Force'):

    if plot_case == 'Force':
        fig = plot_sequence(tq_n_dg.force, tq_n_dg.depth, case='Force')

    else:
        fig = plot_sequence(tq_n_dg.torque, tq_n_dg.depth, case='Torque')

    return fig


def plot_sequence(data, depth, case='Force'):

    if case == 'Force':
        case_units = 'kN'
    else:
        case_units = 'kN*m'

    fig = go.Figure()

    values = []
    if len(data["lowering"]) > 0:
        fig.add_trace(go.Scatter(x=data["lowering"], y=depth,
                                 mode='lines',
                                 name='Tripping In'))
        values += [min(data["lowering"]), max(data["lowering"])]
    if len(data["static"]) > 0:
        fig.add_trace(go.Scatter(x=data["static"], y=depth,
                                 mode='lines',
                                 name='Rotating Off Bottom'))
        values += [min(data["static"]), max(data["static"])]
    if len(data["hoisting"]) > 0:
        fig.add_trace(go.Scatter(x=data["hoisting"], y=depth,
                                 mode='lines',
                                 name='Tripping Out'))
        values += [min(data["hoisting"]), max(data["hoisting"])]
    fig.update_yaxes(autorange="reversed")
    fig.update_layout(
                      xaxis_title=case + ', ' + case_units,
                      yaxis_title='Measured Depth, m')
    if 'makeup_torque' in data and len(data['makeup_torque']) > 0:
        fig.add_trace(go.Scatter(x=data["makeup_torque"], y=depth,
                                 mode='lines',
                                 name='makeup_torque'))
        values += [min(data["makeup_torque"]), max(data["makeup_torque"])]

    return fig
```