

UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

PROJETO ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE:
CÁLCULO DE TRAJETÓRIA DIRECIONAL PARA PERFURAÇÃO DE
POÇOS DE PETRÓLEO: TIPO 1 - *BUILD AND HOLD*
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

ANTÔNIO JOSÉ DOS REIS NETO - Versão 1
TATIANA VITÓRIO ISIDORIO - Versão 1
Prof. André Duarte Bueno

MACAÉ - RJ
JUNHO - 2017

Sumário

1	Introdução	1
1.1	Escopo do problema	1
1.2	Objetivos	2
2	Especificação	3
2.1	Nome do sistema/produto	3
2.2	Especificação do programa	3
2.2.1	Requisitos funcionais	4
2.2.2	Requisitos não funcionais	4
2.3	Casos de uso do software	4
2.3.1	Diagrama de casos de uso geral	5
2.3.2	Diagrama de caso de uso específico	5
3	Elaboração	6
3.1	Análise de domínio	6
3.2	Formulação teórica - conceitos da perfuração direcional	7
3.2.1	Aplicações da perfuração direcional	7
3.2.2	Definições e terminologias	8
3.2.3	Classificação de poços direcionais	10
3.2.4	Planejamento da trajetória de um poço	11
3.2.5	Tipos de poços direcionais e classificação	12
3.3	Formulação matemática	13
3.4	Diagrama de pacotes – assuntos	16
4	AOO – Análise Orientada a Objeto	17
4.1	Diagramas de classes	17
4.1.1	Dicionário de classes	17
4.2	Diagrama de seqüência – eventos e mensagens	19
4.2.1	Diagrama de seqüência geral	19
4.3	Diagrama de comunicação – colaboração	20
4.4	Diagrama de máquina de estado	20
4.5	Diagrama de atividades	21

5	Projeto	22
5.1	Projeto do sistema	22
5.2	Projeto orientado a objeto – POO	23
5.3	Diagrama de componentes	24
5.4	Diagrama de implantação	25
6	Implementação	26
6.1	Código fonte	26
7	Teste	49
7.1	Teste 1: Interface	49
7.2	Teste 2: Resultados	50
8	Documentação	56
8.1	Documentação do usuário	56
8.1.1	Como instalar o software	56
8.2	Documentação para desenvolvedor	57
8.2.1	Dependências	57
8.2.2	Como gerar a documentação usando doxygen	57
	Referências Bibliográficas	60

Capítulo 1

Introdução

No presente projeto de engenharia desenvolve-se o software para cálculo de trajetória direcional para perfuração de poços de petróleo, um software aplicado a engenharia de petróleo e que utiliza o paradigma da orientação a objetos.

A importância deste trabalho está relacionada com a técnica de perfuração direcional, que visa o desvio intencional do poço para atingir um ou mais alvos prefixados. O planejamento de um poço direcional é baseado em escolher, projetar e executar a trajetória de um poço inclinado ou horizontal, bem como indicar os parâmetros compatíveis com a trajetória escolhida.

1.1 Escopo do problema

A perfuração de poços no Brasil tem se destacado ao longo dos anos por causa dos inúmeros desafios que têm sido vencidos, notadamente no que se refere à perfuração em águas profundas e ultra-profundas, como é o caso dos poços do pré-sal. Com o avanço das tecnologias utilizadas na perfuração, deu-se início aos poços direcionais, que possibilitam o aumento na produtividade e, ao mesmo tempo uma redução no impacto ambiental, quando por exemplo, se deseja atingir formações produtoras que estejam abaixo de locações urbanas e/ou ambientalmente sensíveis (lagos, rios, cidades, etc.).

Poços de petróleo podem ser classificados, geometricamente, como poços verticais e direcionais. Estes ainda apresentam algumas variações: poços horizontais, de longo alcance (*ERW* - *Extended Reach Well*) ou multilateral. Um adequado planejamento direcional pode ser a chave do sucesso de um poço, principalmente para aqueles mais complicados como os poços com grande afastamento (*ERW*). O planejamento envolve o estabelecimento da trajetória, análises técnicas e pesquisas operacionais sobre as melhores práticas para a perfuração em determinada região. A trajetória direcional tem como objetivo atingir o alvo geológico definido previamente pela equipe multidisciplinar formada por geólogos, geocientistas e engenheiros de petróleo.

O presente trabalho pretende simplificar, de forma eficiente, o cálculo de uma trajetória

vertical e direcional 2D, mantendo o azimuth constante ao longo da trajetória, e apresentar para o usuário a previsão da inclinação, azimuth e profundidade vertical (*TVD – True Vertical Depth*) do projeto em desenvolvimento, em relação a profundidade medida (*MD – Measured Depth*).

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
 - Desenvolver uma solução para o cálculo da trajetória em 2D de um projeto de poço vertical e direcional.
- Objetivos específicos:
 - Calcular a trajetória vertical do poço;
 - Calcular a trajetória direcional em 2D;
 - Calcular o afastamento, raio de curvatura e profundidade da seção vertical;
 - Simulação dos parâmetros operacionais compatíveis com o projeto requisitado;
 - Plotar os gráficos relacionando a inclinação, azimuth e profundidade vertical em relação a profundidade medida;

Capítulo 2

Especificação

Apresenta-se neste capítulo do projeto de engenharia a concepção, a especificação do sistema a ser modelado e desenvolvido.

2.1 Nome do sistema/produto

Nome	Cálculo da Trajetória Direcional para Perfuração de Poços de Petróleo
Componentes principais	Simulação da trajetória do poço a partir dos parâmetros de entrada que informam as coordenadas da cabeça do poço e do respectivo alvo geológico.
Missão	Auxiliar no desenvolvimento de projetos de perfuração, provendo informações e parâmetros para o desenvolvimento da trajetória de um poço vertical ou direcional.

2.2 Especificação do programa

O principal objetivo deste projeto de engenharia é o desenvolvimento de um software para o cálculo da trajetória de poços. O software será livre com licença GNU/GPL v1.0, multiplataforma e orientado a objeto em C++, de modo a facilitar futuras modificações. Para a execução do software, são necessários os seguintes dados de entrada:

- Coordenadas da cabeça do poço e do alvo, em UTM (*Universal Transversa de Mercator*);
- Profundidade vertical do alvo geológico;
- Profundidade do começo da seção de ganho de ângulo (*KOP - Kick Off Point*);
- Taxa de ganho de ângulo constante (*BUR - Build Up Rate*).

Após a entrada de dados pelo usuário via terminal, o programa irá auxiliar na simulação da melhor trajetória possível para os dados informados. Os dados de saída serão mostrados no terminal para avaliação do usuário, podendo ser salvos em “.txt” quando o usuário achar satisfatória a trajetória criada. Além do arquivo gerado, o programa retornará todas as relações entre profundidade, inclinação e azimute criadas, em forma gráfica para melhorar a visualização dos dados gerados.

2.2.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O usuário deverá ter liberdade para escolher o que deseja calcular.
RF-02	O programa deverá permitir o carregamento de dados de trajetória pelo usuário.
RF-03	Os resultados deverão ser exportados como texto e/ou gráficos.

2.2.2 Requisitos não funcionais

RNF-01	Os cálculos devem ser feitos utilizando-se coordenadas UTM.
RNF-02	O programa deverá ser multi-plataforma, podendo ser executado em <i>Windows</i> , <i>GNU/Linux</i> ou <i>Mac</i> .

2.3 Casos de uso do software

Nesta seção, apresentamos a Tabela 2.1 que exemplifica um caso de uso do sistema, bem como os diagramas de casos de uso.

Tabela 2.1: Exemplo de caso de uso.

Nome do caso de uso:	Cálculo da trajetória de um poço de petróleo.
Resumo/descrição:	Cálculo do raio dos arcos de uma circunferência usados nos cálculos dos trechos de ganho e perda de ângulo.
Etapas:	<ol style="list-style-type: none"> 1. Criar objeto função. 2. Escolher o sistema de unidades. 3. Entrar com a taxa de ganho de ângulo. 4. Calcular o raio de curvatura. 5. Apresentar o valor calculado no terminal.
Cenários alternativos:	Um cenário alternativo envolve a escolha do sistema de unidades não compatível com a unidade da taxa de ângulo.

2.3.1 Diagrama de casos de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário entrando com os dados necessários, calculando a trajetória, visualizando o resultado no terminal e analisando os gráficos.

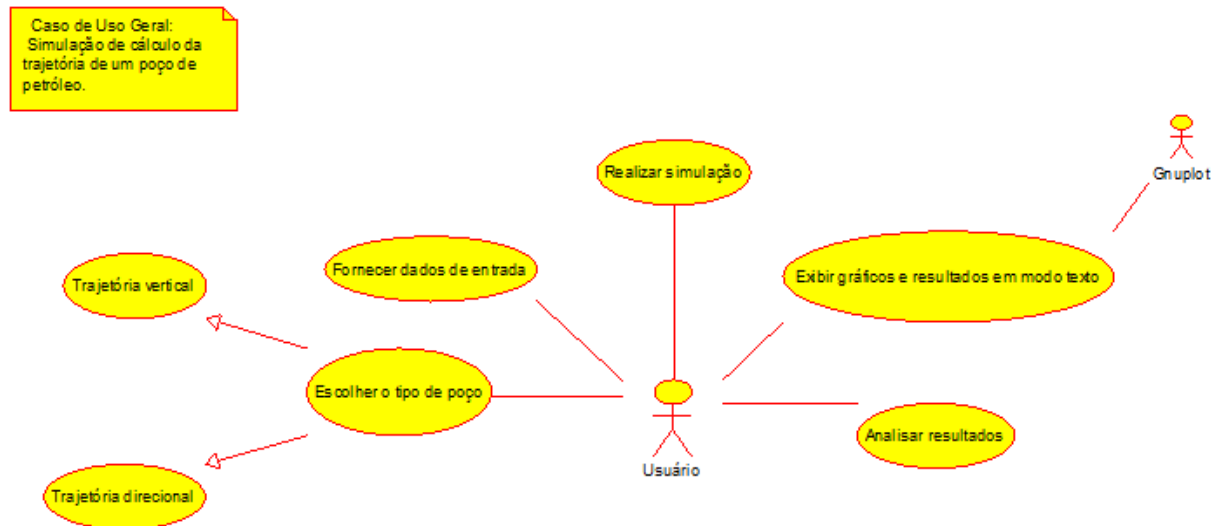


Figura 2.1: Diagrama de caso de uso – Caso de uso geral

2.3.2 Diagrama de caso de uso específico

O diagrama a seguir representa um detalhamento do cálculo da trajetória apresentado na Figura 2.1 que é a classificação da trajetória quanto ao raio de curvatura detalhado na Figura 2.2.

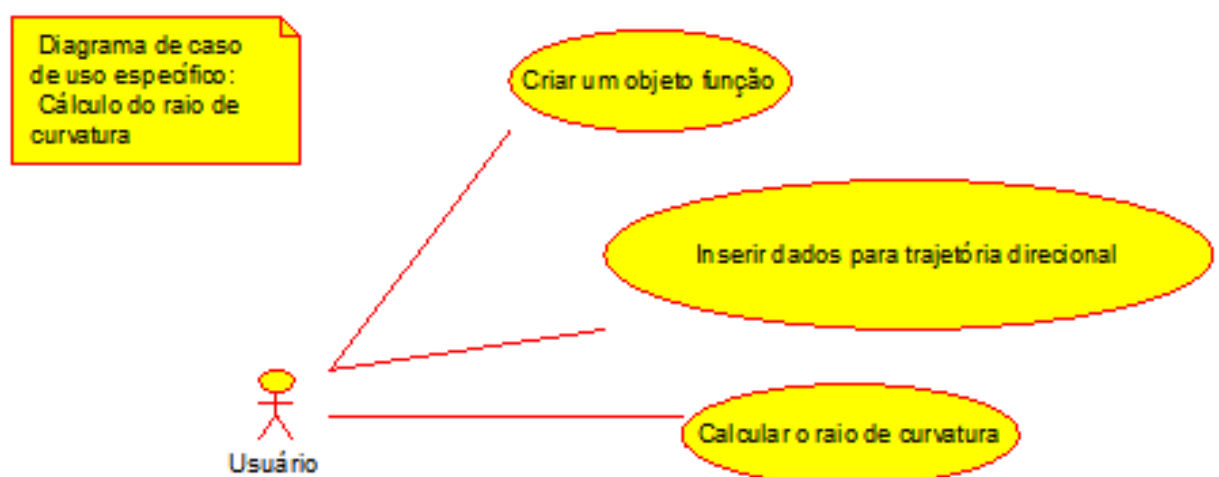


Figura 2.2: Diagrama de caso de uso específico – Título.

Capítulo 3

Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, a equipe de desenvolvimento do projeto de engenharia passa por um processo de elaboração que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

Na elaboração fazemos uma análise dos requisitos, ajustando os requisitos iniciais de forma a desenvolver um sistema útil, que atenda às necessidades do usuário e, na medida do possível, permita seu reuso e futura extensão.

Eliminam-se os requisitos "impossíveis" e ajusta-se a idéia do sistema de forma que este seja flexível, considerando-se aspectos como custos e prazos.

3.1 Análise de domínio

- Área relacionada:

A principal área relacionada a este projeto é a engenharia de poço.

- Sub-área relacionada:

A perfuração de um poço é fundamental para o desenvolvimento de um campo de petróleo, pois ele é a via de ligação entre a superfície e um alvo geológico. Para que a perfuração ocorra são necessários três parâmetros principais: o torque, a rotação e o peso sobre a broca. Além disso, tem a necessidade de uma sonda, local onde se encontram os equipamentos necessários para este processo, além de toda a equipe envolvida.

Com o método rotativo, a perfuração é realizada com um conjunto de ferramentas e tubos de perfuração que ao final tem uma broca em contato com a formação, responsável por esmerilhar a rocha. Podemos dividir os poços perfurados em três grupos de acordo com suas trajetórias:

- Verticais com pouca ou nenhuma inclinação em relação a vertical;
- Direcionais, poço no qual possui uma inclinação pré-determinada;

- Horizontais com uma inclinação final de 90 graus em relação a vertical.

Dentro deste contexto, optamos por desenvolver um software que irá calcular a trajetória direcional de um poço em 2D, levando em consideração as profundidades medidas e verticais e a inclinação, mantendo o azimuth constante durante a trajetória. Um poço direcional tem como vantagem acessar diferentes objetivos em coordenadas variadas, economizando assim, tempo de projeto e execução e consequentemente um alto valor de investimento. Além disso, poços direcionais tem a capacidade de aumentarem a produção de um determinado reservatório, pois eles conseguem uma maior área de exposição dentro da zona de óleo.

3.2 Formulação teórica - conceitos da perfuração direcional

O objetivo de direcionar uma trajetória na direção correta para acertar um alvo a muitos quilômetros de profundidade forçou a indústria a focar em ferramentas e métodos para identificar a localidade exata do poço e o seu caminho durante a perfuração. Antigamente nos poços exploratórios, era comum alocar a sonda exatamente acima do alvo e perfurar um poço vertical até ele. Com a evolução da indústria e corrida por mais óleo, se tornou necessário perfurar poços para alcançar alvos que eram afastados horizontalmente da localização da sonda na superfície. Hoje em dia, existe diversas empresas que oferecem serviços e ferramentas para desviar o poço e direcioná-lo, medindo sua inclinação e azimuth durante o processo.

3.2.1 Aplicações da perfuração direcional

De acordo com a Figura 3.1, apresenta-se diferentes razões pelas quais a perfuração direcional se faz necessária:

1. Poços direcionais em áreas de domos salinos;
2. Poços direcionais em zonas fraturadas;
3. Poços multilaterais e horizontais;
4. *Sidetrack* - desvio a partir de um poço já perfurado;
5. Poços direcionais em áreas que geologicamente, não possibilitem um poço vertical;
6. Poços direcionais para a exploração de uma reserva mais rasa;
7. Poços direcionais para o controle de um blowout;
8. Direcionais em áreas urbanas e de proteção ambiental.

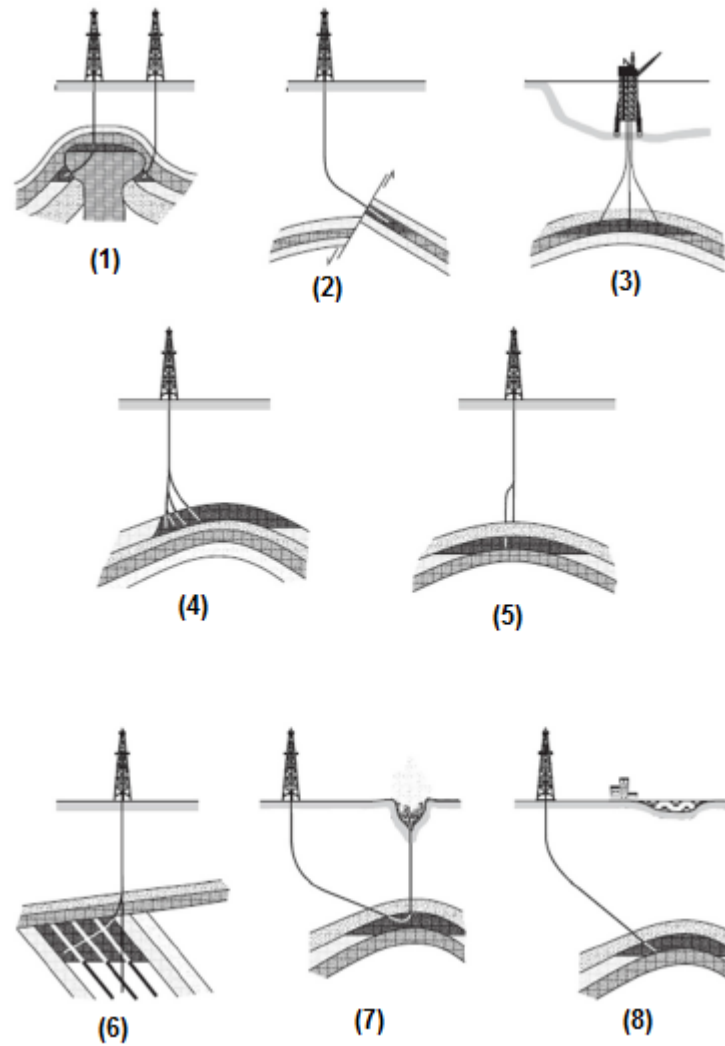


Figura 3.1: Diversas aplicações de poços direcionais utilizadas na indústria de petróleo (Bourgoyne et al., 1991)

3.2.2 Definições e terminologias

Durante o desenvolvimento de um projeto de poço, nos deparamos com diversos termos que devem ser bem definidos para evitar qualquer incompatibilidade de projeto. A terminologia a seguir é utilizada:

- **Afastamento:** termo utilizado para expressar a distância horizontal da cabeça do poço com um alvo pré-determinado.
- **Profundidade Vertical:** representa a distância vertical da mesa rotativa a um ponto do poço;
- **Profundidade Medida:** é a distância percorrida pela broca até atingir determinado ponto do poço;

- **Objetivo:** é o ponto que a trajetória deve atingir. É determinado com o auxílio de geólogos e engenheiros de reservatório;
- **Alvo:** já o alvo representa uma área, onde se encontra o objetivo, mais um raio de tolerância caso ocorra algum desvio da trajetória planejada;
- **Inclinação:** a inclinação é definida como sendo o ângulo entre o vetor local gravitacional e a tangente ao eixo do poço;
- **Azimute:** indica a direção do poço, variando de 0 a 360 graus medindo-se no sentido horário a partir do norte geográfico;

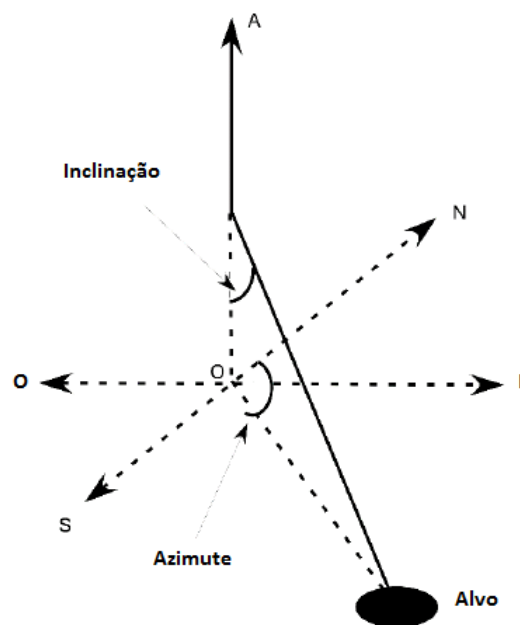


Figura 3.2: Parâmetros medidos de um poço direcional (modificado de Gabolde e Nguyen, 1991)

- **KOP – Kickoff Point:** é o começo da seção de ganho de ângulo máximo do trecho reto: representa o ângulo máximo ao final da seção de ganho de ângulo, no qual é mantido constante no trecho reto;
- **Buildup, Buildup Rate e End-of-buildup:** *buildup* é a seção onde acontece o ganho de ângulo, na qual a inclinação varia com a profundidade. Normalmente esse ganho ocorre a uma taxa de ganho de ângulo constante, na qual chamamos de *Buildup Rate (BUR)*, expressa em graus/30 metros. O final do trecho de ganho de ângulo que precede o trecho reto é chamado de *End-of-build*.
- **Início do Drop off e Seção de Drop off:** é a profundidade onde o poço começa a perder ângulo. O trecho do poço onde ocorre esta perda de ângulo é conhecido como seção de *drop off*.

- **Seção Tangente ou *Slant***: é o trecho onde a inclinação é mantida até atingir o objetivo ou uma nova seção de ganho ou perda de ângulo
- ***Dogleg* e *Dogleg Severity (DLS)***: *dogleg* é o ângulo no espaço formado por dois vetores tangentes à trajetória. Já o *dogleg severity* refere-se ao ângulo dividido pelo comprimento perfurado ou a ser perfurado.
- **Raio de curvatura (R)**: é o raio dos arcos de circunferência usados nos cálculos dos trechos de *buildup*.

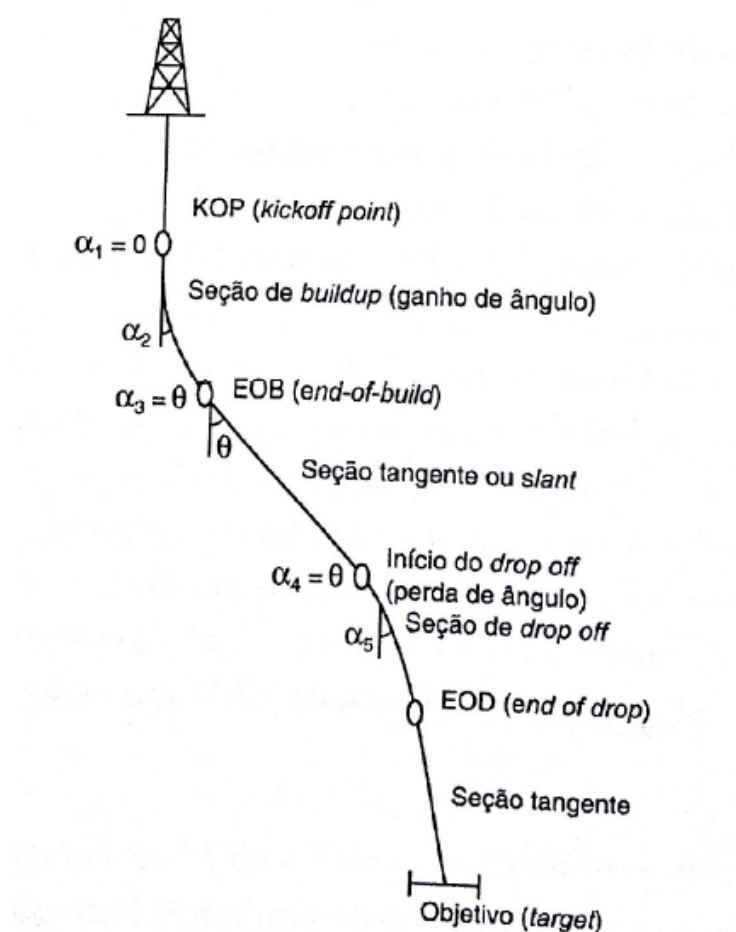


Figura 3.3: Definições direcionais de um poço (Rocha, L.A et al., 2008)

3.2.3 Classificação de poços direcionais

Esta classificação tem o objetivo de identificar o grau de severidade de cada poço direcional.

- **Classificação quanto ao raio de curvatura:**

Os poços podem ser classificados como sendo de raio curto, intermediário, médio e longo. Como a variação da inclinação é considerada constante ao longo do trecho de *buildup*

do poço, o resultado é um arco de círculo com um determinado raio de curvatura. Tal parâmetro pode ser calculado pela seguinte expressão:

$$R = \frac{180}{\pi} * \frac{1}{q} \quad (3.1)$$

Tabela 3.1: Classificação da trajetória quanto ao raio de curvatura.

Classificação	BUR ($^{\circ}$ /30m)	Raio (m)
Raio longo	2 - 8	859 - 215
Raio médio	8 - 30	215 - 57
Raio intermediário	30 - 60	57 - 29
Raio curto	60 - 200	29 - 9

- **Classificação quanto ao afastamento do objetivo:**

Os poços podem ser classificados em convencional, de grande afastamento (*ERW* – *extended reach well*) e de afastamento severo (*S-ERW* – *severe extended reach well*). Os dados utilizados nesta classificação são: afastamento, profundidade vertical (PV) e lâmina d'água (LA) para poços marítimos.

Tabela 3.2: Classificação da trajetória quanto ao raio de curvatura.

Tipo de poço	Afastamento / (PV-LA)
Convencional	< 2
ERW	2 - 3
S-ERW	> 3

- **Classificação quanto ao giro:**

Os poços são classificados em bidimensional (2D) ou tridimensional (3D). Estes são também conhecidos como poço de projetista ou *designer wells*.

3.2.4 Planejamento da trajetória de um poço

A elaboração da trajetória de um poço com uma inclinação específica deve ser escolhida de modo que satisfaça da melhor maneira possível as condições de viabilidade econômica de acordo com a sonda, de modo que o custo seja minimizado. Durante o planejamento e a perfuração do poço a posição de todos os pontos ao longo da trajetória é considerada em três dimensões.

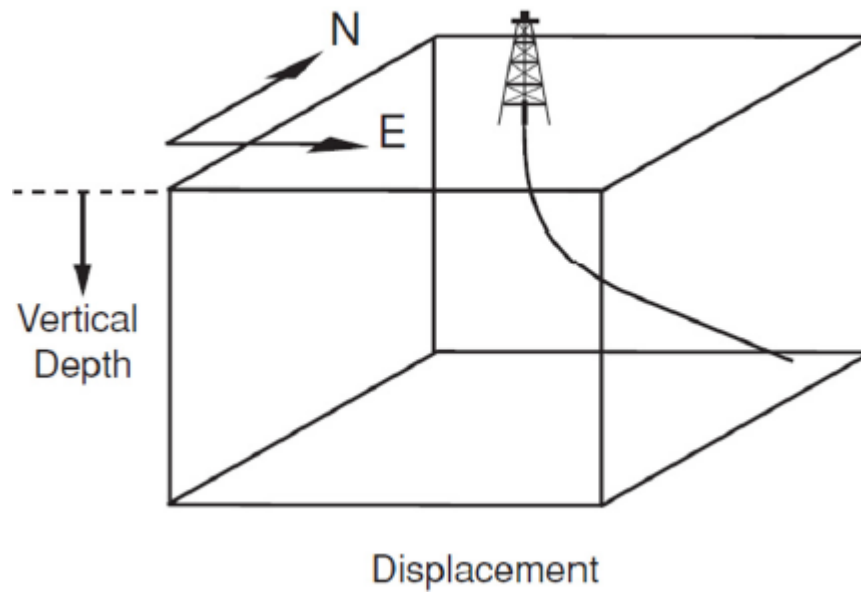


Figura 3.4: Coordenadas de referência para planejamento de um poço.

3.2.5 Tipos de poços direcionais e classificação

Uma vez fixadas as coordenadas da locação da cabeça do poço, e as coordenadas do alvo geológico, escolhe-se o modelo da trajetória do poço. Os principais tipos de poços direcionais são:

- **Tipo 1 (*Build and hold*):** o poço é vertical até o KOP onde o inicia-se o ganho de ângulo. Quando a inclinação desejada é alcançada, o poço é mantido tangente ou reto até que o alvo seja alcançado.
- **Tipo 2 (*Build, hold and drop* ou *S*):** na seção superior é semelhante ao tipo 1. Após a seção tangente, a seção de perda de ângulo é perfurada e a trajetória do poço até o alvo geológico é aproximadamente vertical.
- **Tipo 3 (*Continuous build*):** é um poço perfurado quando existe um obstáculo, por exemplo, um domo salino. O poço é vertical até um KOP com uma profundidade mais acentuada. A fase de *buildup* termina sem um trecho de inclinação constante e então o alvo é alcançado.

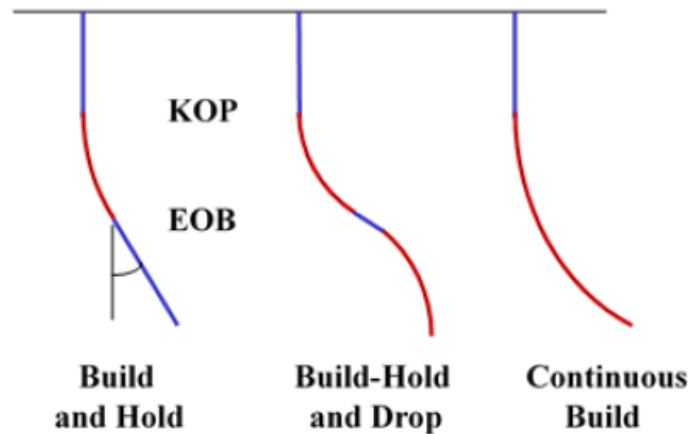


Figura 3.5: Tipos de poços direcionais mais comuns na perfuração de óleo e gás.

Uma vez fixado o modelo da trajetória, o projeto do poço direcional é apresentado sob a forma de projeções do poço nos planos vertical e horizontal.

Uma vez que o engenheiro de perfuração apenas poderá determinar a profundidade vertical do poço, as seguintes informações também serão necessárias:

- Profundidade vertical do KOP (selecionados pelo engenheiro);
- BUR e seção de *buildup* (selecionados pelo engenheiro);
- Direção que poço será perfurado após o KOP em graus a partir do norte (definida pela posição do alvo em relação a sonda);
- Profundidade vertical ao final do *buildup* e o início da seção tangente;
- Profundidade vertical do alvo geológico.

3.3 Formulação matemática

O desenvolvimento deste projeto de engenharia será restrito ao perfil de poço tipo 1, também conhecido como perfil J.

A escolha do sistema de coordenadas depende de diversos fatores incluindo a localização do alvo geológico e da proximidade com outros poços. As coordenadas do alvo e a profundidade são selecionadas pelo geologista. O engenheiro de perfuração é responsável pela escolha do KOP e do BUR.

A Figura 3.6 mostra a trajetória de um perfil *build-and-hold*.

Onde:

TVD_{AB} : distância da superfície ao KOP, em metros;

B-D: distância do KOP ao objetivo, em metros;

D_h : deslocamento horizontal, em metros;

TVD_{AG} : profundidade vertical, em metros;

MD (A-D): profundidade medida, em metros;

q: BUR, °/30 m.

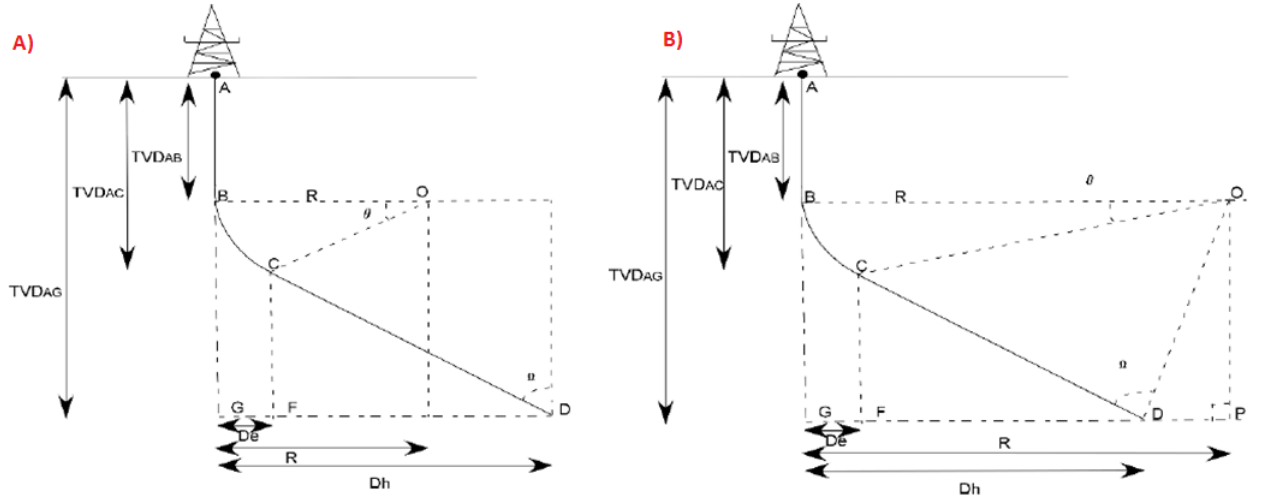


Figura 3.6: Geometria da trajetória tipo 1 com A) $D_h > R$, e B) $D_h < R$

Para as equações seguintes, note que $TVD_3 = TVD_{AG}$, $TVD_2 = TVD_{AC}$, $TVD_1 = TVD_{AB}$.

A inclinação máxima, θ , considerado na Figura 3.6 pode ser descrita por:

$$90 = \theta + (90 - \Omega) + \tau \quad (3.2)$$

$$\theta = \Omega - \tau \quad (3.3)$$

O ângulo τ pode ser encontrado considerando o triângulo OPD, no caso ($R > D_h$):

$$\tan(\tau) = \frac{DP}{PO} = \frac{R - D_h}{TVD_3 - TVD_1} \quad (3.4)$$

e assim,

$$\tau = \arctan\left(\frac{R - D_h}{TVD_3 - TVD_1}\right) \quad (3.5)$$

O ângulo Ω pode ser encontrado considerando ODC, onde:

$$\sin(\Omega) = \frac{R}{OP} \quad (3.6)$$

e

$$L_{ob} = \sqrt{(R - D_h)^2 + (TVD_3 - TVD_1)^2} \quad (3.7)$$

substituindo OP na equação 3.6 obtemos:

$$\sin(\Omega) = \frac{R}{\sqrt{(R - D_h)^2 + (TVD3 - TVD1)^2}} \quad (3.8)$$

O ângulo de inclinação máxima, θ , para o caso deste tipo de poço onde ($D_h < R$) é:

$$\theta = \arcsin \left[\frac{R}{\sqrt{[(R - D_h)^2 + (TVD3 - TVD1)^2]}} \right] - \arctan \left(\frac{R - D_h}{TVD3 - TVD1} \right) \quad (3.9)$$

O comprimento do arco da seção BC é:

$$L(BC) = \frac{\pi}{180} * R * \theta \quad (3.10)$$

ou

$$L(BC) = \frac{\theta}{q} \quad (3.11)$$

O comprimento do caminho da trajetória, CD, a um ângulo de inclinação constante pode ser determinado pelo triângulo DCO da seguinte maneira:

$$\tan(\Omega) = \frac{CO}{Lcb} = \frac{R}{Lcb} \quad (3.12)$$

e

$$Lcb = \frac{R}{\tan(\Omega)} \quad (3.13)$$

A profundidade medida total, D_M , para a profundidade vertical de TVD3 é:

$$D_M = TVD1 + \frac{\theta}{q} + \frac{R}{\tan(\Omega)} \quad (3.14)$$

onde D_M é igual a seção vertical até o *kickoff* mais a seção de *build* mais a seção *slant* (Figura 3.6).

O afastamento horizontal GF (D_E) ao final do ganho de ângulo pode ser determinado considerando D'CO, onde:

$$D_E = R - R \cos(\theta) = R(1 - \cos(\theta)) \quad (3.15)$$

Para encontrar a profundidade medida e o afastamento horizontal ao longo de qualquer parte da seção de ganho antes de alcançar o ângulo máximo θ , basta considerar a inclinação intermediária θ' , o ângulo de inclinação em C', que fornecerá um novo afastamento horizontal, D_n .

Estas equações serão válidas apenas para o caso onde ($D_h < R$). Uma outra maneira de expressar o ângulo máximo de inclinação, θ , em função de R, TVD1, TVD3, e D_h para

$(D_h > R)$ é:

$$\theta = \arctan\left(\frac{TVD3 - TVD1}{R - D_h}\right) - \arccos\left[\left(\frac{R}{TVD3 - TVD1}\right) * \sin\left(\arctan\left(\frac{TVD3 - TVD1}{R - D_h}\right)\right)\right] \quad (3.16)$$

3.4 Diagrama de pacotes – assuntos

Apresentados os conceitos matemáticos relacionados ao desenvolvimento do software, os assuntos identificados são descritos na Figura 3.7:

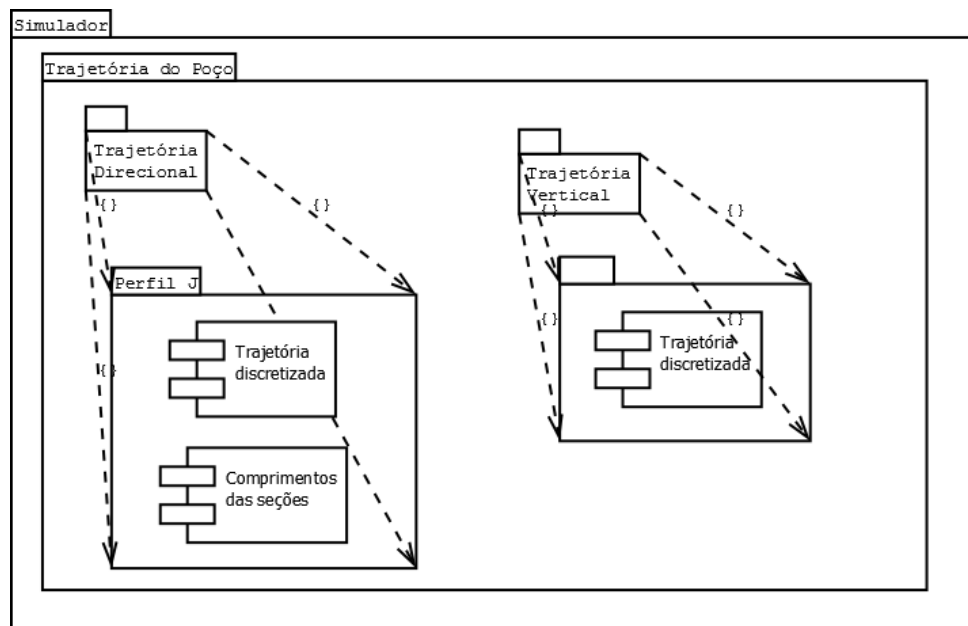


Figura 3.7: Diagrama de Pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

Apresenta-se neste capítulo a Análise Orientada a Objeto - AOO, as relações entre as classes, os atributos, os métodos e suas associações. A análise consiste em modelos estruturais dos objetos e seus relacionamentos, e modelos dinâmicos, apresentando as modificações do objeto com o tempo. O resultado da análise é um conjunto de diagramas que identificam os objetos e seus relacionamentos.

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.

4.1.1 Dicionário de classes

- Classe CTrajVertical: foi desenvolvida para calcular a trajetória vertical de um poço.
- Classe CParametrosDirecional: representa a classe que irá calcular todos os parâmetros necessários para os cálculos seguintes da trajetória. Os parâmetros podem ser: o ângulo máximo, o comprimento da seção de ganho de ângulo, o raio de curvatura, entre outros.
- Classe CTrajDirecionalBuildUp: representa a seção de ganho de ângulo, o cálculo dos ângulos que o poço ganha até atingir o ângulo máximo especificado pelo usuário.
- Classe CTrajDirecionalSlant: representa o cálculo da parte final da trajetória, onde o ângulo máximo é mantido e se tem a seção tangente para encontrar o alvo geológico.
- Classe CSimulador: representa a simulação do programa como um todo.
- Classe CGnuplot: possibilita a geração de gráficos usando o software externo Gnuplot.

4.2 Diagrama de seqüência – eventos e mensagens

O diagrama de seqüência enfatiza a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do software.

4.2.1 Diagrama de seqüência geral

Veja os diagramas de seqüência nas Figuras 4.2 e 4.3.

- O diagrama de seqüência aqui representado, demonstra como o software faz para plotar os gráficos com os resultados finais. Primeiramente o usuário insere os dados pedidos pelo simulador que por sua vez chama a classe CTrajVertical para calcular o primeiro trecho do poço. Em seguida o software calcula todos os parâmetros necessários para criar as seções de ganho de ângulo e tangente. Finalizamos a simulação da trajetória com a apresentação dos gráficos para serem analisados.

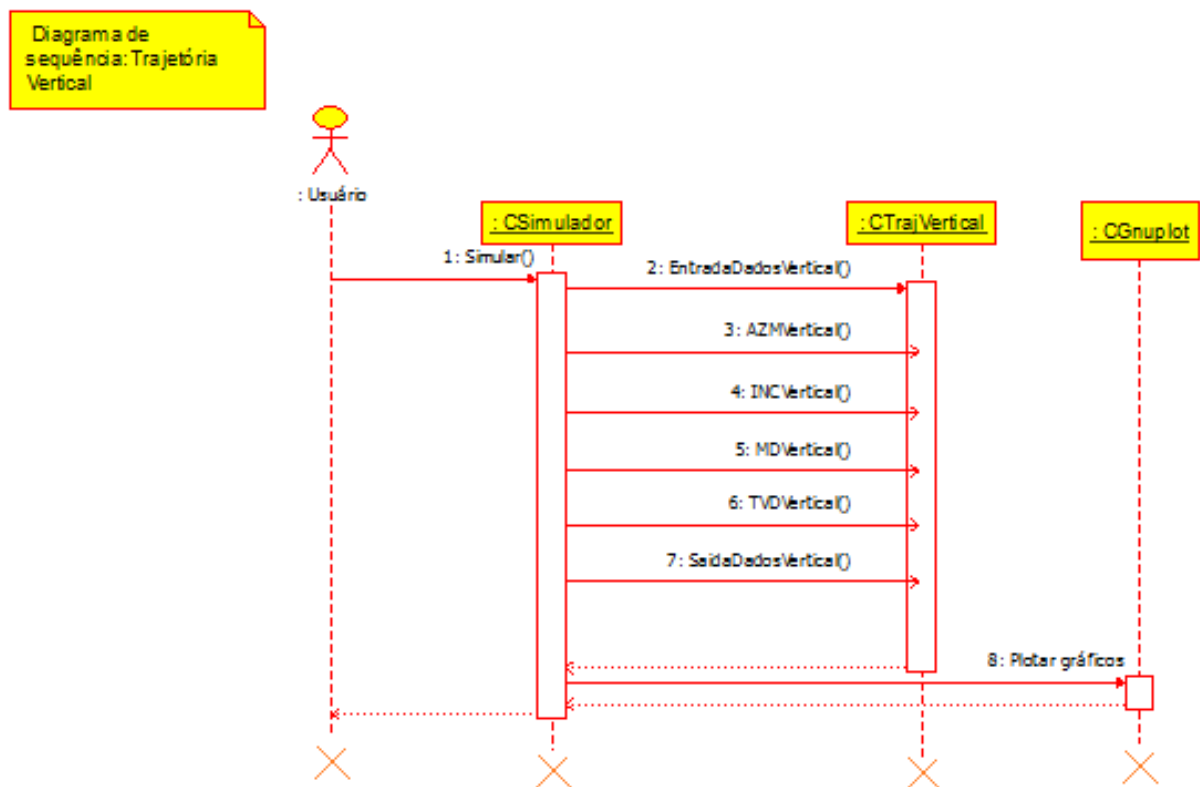


Figura 4.2: Diagrama de seqüência: trajetória vertical.

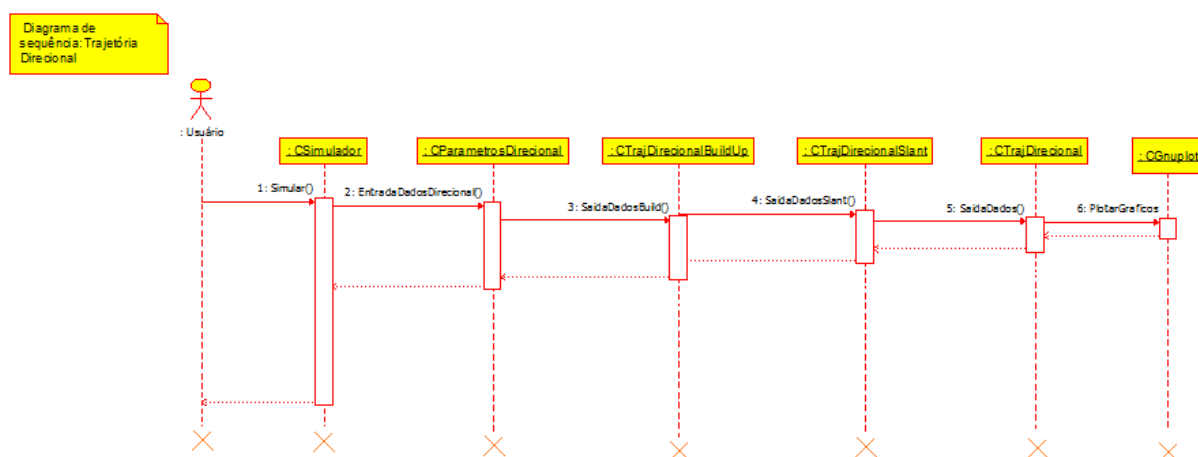


Figura 4.3: Diagrama de seqüência: trajetória direcional.

4.3 Diagrama de comunicação – colaboração

No diagrama de comunicação o foco é a interação e a troca de mensagens e dados entre os objetos. Veja na Figura 4.3 o diagrama de comunicação mostrando a sequência que um objeto da classe CSimulador utiliza para construir a trajetória completa do poço.

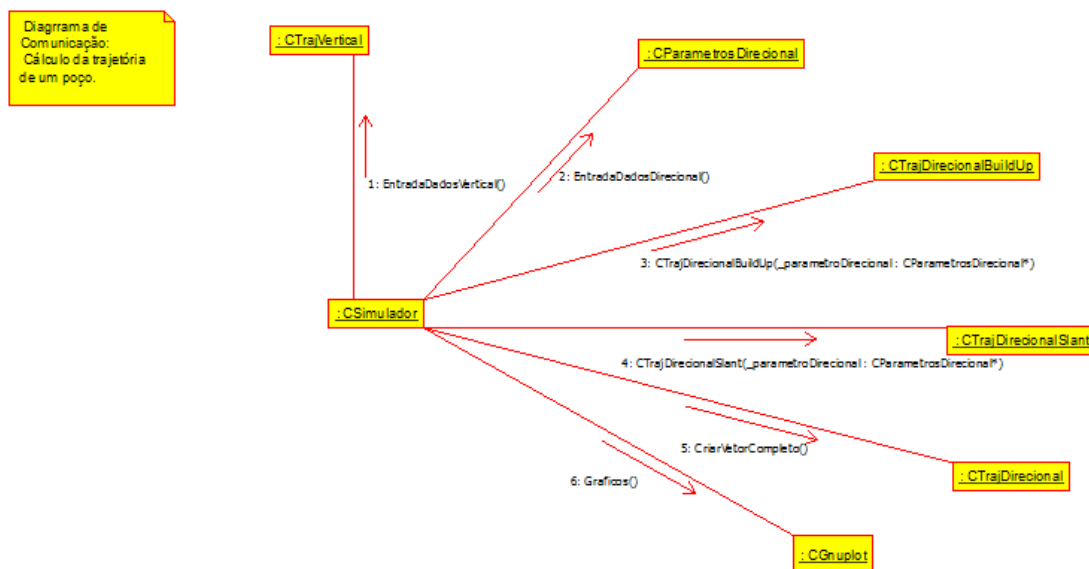


Figura 4.4: Diagrama de comunicação

4.4 Diagrama de máquina de estado

Um diagrama de máquina de estado representa os diversos estados que o objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico do objeto). É usado para modelar aspectos dinâmicos do objeto.

A Figura 4.5 demonstra os estados que a Classe CParametros apresenta durante a execução do software.

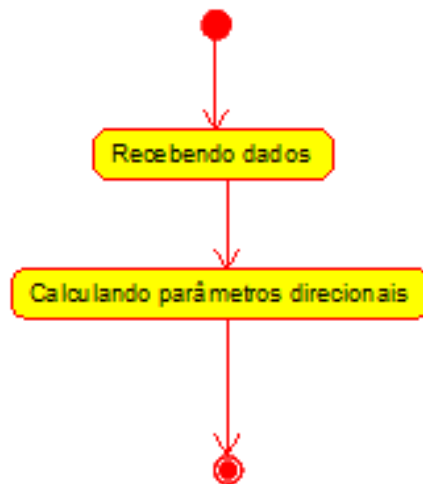


Figura 4.5: Diagrama de máquina de estado: Classe Parâmetros Direcionais

4.5 Diagrama de atividades

Veja na Figura 4.6 o diagrama de atividades corresponde a uma atividade específica da Classe CParametros, onde calcula-se o ângulo máximo de inclinação a partir do raio de curvatura, coordenadas UTM, afastamento horizontal do alvo e profundidade de KOP.

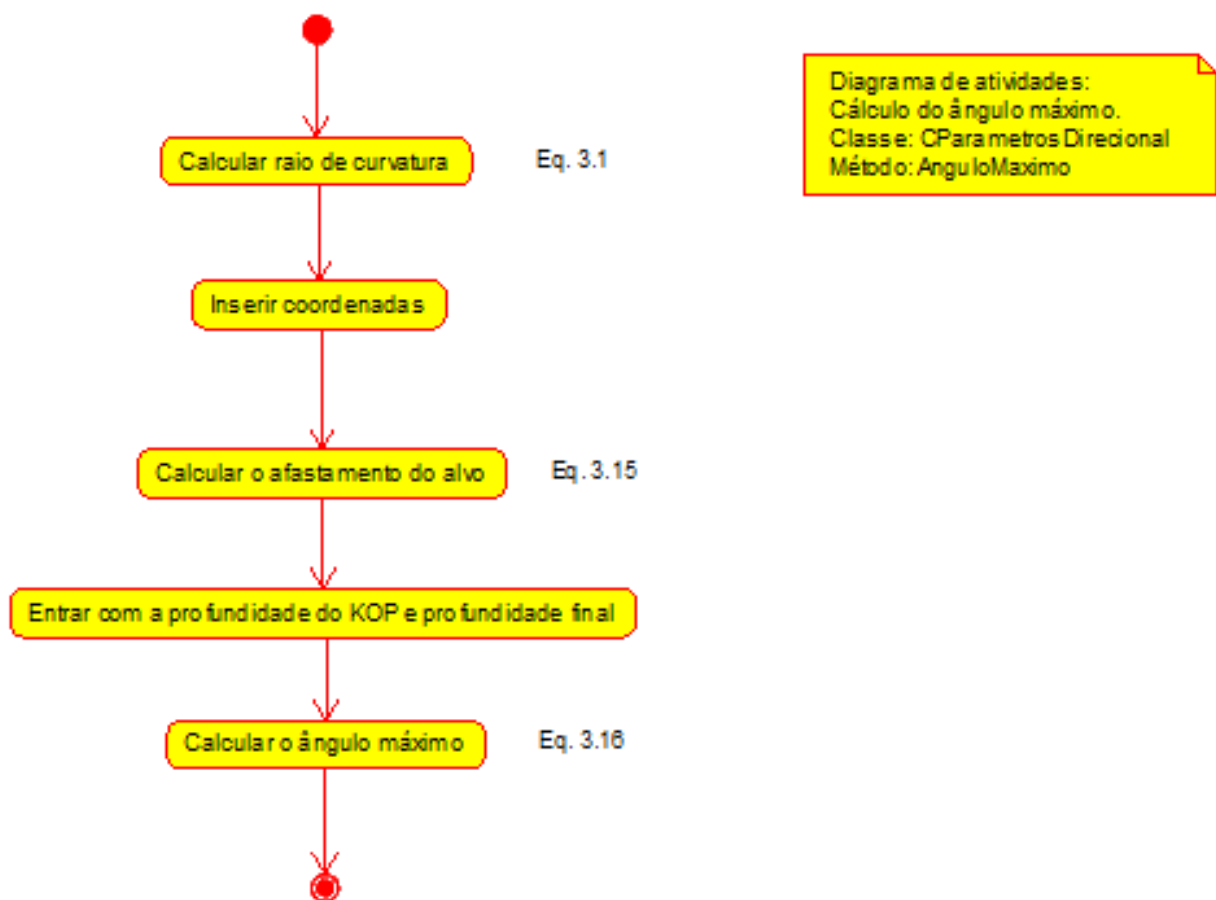


Figura 4.6: Diagrama de atividades - Cálculo do ângulo máximo.

Capítulo 5

Projeto

Neste capítulo do projeto de engenharia veremos questões associadas ao projeto do sistema, incluindo protocolos, recursos, plataformas suportadas, implicações nos diagramas feitos anteriormente, diagramas de componentes e implantação. Na segunda parte revisamos os diagramas levando em conta as decisões do projeto do sistema.

5.1 Projeto do sistema

Depois da análise orientada a objeto desenvolve-se o projeto do sistema, qual envolve etapas como a definição dos protocolos, da interface API, o uso de recursos, a subdivisão do sistema em subsistemas, a alocação dos subsistemas ao hardware e a seleção das estruturas de controle, a seleção das plataformas do sistema, das bibliotecas externas, dos padrões de projeto, além da tomada de decisões conceituais e políticas que formam a infraestrutura do projeto.

Deve-se definir padrões de documentação, padrões para o nome das classes, padrões de retorno e de parâmetros em métodos, características da interface do usuário e características de desempenho.

1. Protocolos

- A única intercomunicação será entre o software desenvolvido e o software Gnuplot, que plotará os gráficos desejados pelo usuário;
- O software receberá dados via teclado;
- A interface utilizada será em modo texto;
- O software terá como saída arquivos de extensão .dat e gráficos em arquivos de extensão .png.

2. Recursos

- O presente programa precisará utilizar o HD, o processador, o teclado, a tela, o mouse, a memória e demais componentes internos do computador.

3. Controle

- Não haverá necessidade de grande espaço na memória visto que o programa e seus componentes trabalham com dados relativamente pequenos.
- Neste projeto todos os cálculos necessitam de estruturas de repetição, pois são feitos ao longo da profundidade do poço.
- Neste projeto não há necessidade de uso de processos de processamento paralelo, pois os cálculos realizados requerem pouco esforço de processamento.

4. Plataformas

- O software irá operar nos sistemas operacionais Windows e GNU/Linux, sendo desenvolvido e testado em ambos os sistemas.
- Não haverá necessidade de grandes mudanças para tornar o programa multiplataforma pois a linguagem escolhida, C++, tem suporte em todos estes sistemas operacionais, [Bueno, 2003].
- Para a geração de gráficos sera utilizado o software livre Gnuplot.
- Os ambientes de desenvolvimento serão o CodeBlocks (Windows) e Kate (Linux).

5.2 Projeto orientado a objeto – POO

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida e as características da plataforma escolhida (hardware, sistema operacional e linguagem de softwareção). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.

Envolve a otimização da estrutura de dados e dos algoritmos, a minimização do tempo de execução, de memória e de custos. Existe um desvio de ênfase para os conceitos da plataforma selecionada.

Efeitos do projeto no modelo estrutural

- Adicionar nos diagramas de pacotes as bibliotecas e subsistemas selecionados no projeto do sistema (exemplo: a biblioteca gráfica selecionada).
 - Neste projeto foi utilizada como biblioteca graca CGnuplot.
- Novas classes e associações oriundas das bibliotecas selecionadas e da linguagem escolhida devem ser acrescentadas ao modelo.

- Após a análise e o projeto do sistema surgiu a necessidade da criação de novas classes e associações. Problemas como esse poderão surgir durante a implementação do banco de dados, sendo assim passível de modificação ou criação de novas classes, atributos e métodos. Neste projeto foi feita uma associação entre a biblioteca CGnuplot com CTrajVertical e CTrajDirecional para a geração dos gráficos.
- Estabelecer as dependências e restrições associadas à plataforma escolhida.
 - Não se aplica.

Efeitos do projeto no modelo dinâmico

- Revisar os diagramas de seqüência e de comunicação considerando a plataforma escolhida.
- Verificar a necessidade de se revisar, ampliar e adicionar novos diagramas de máquinas de estado e de atividades.

Efeitos do projeto nos métodos

- Em função da plataforma escolhida, verifique as possíveis alterações nos métodos. O projeto do sistema costuma afetar os métodos de acesso aos diversos dispositivos.
- De maneira geral os métodos devem ser divididos em dois tipos: i) tomada de decisões, métodos políticos ou de controle; devem ser claros, legíveis, flexíveis e usam polimorfismo. ii) realização de processamentos, podem ser otimizados e em alguns casos o polimorfismo deve ser evitado.

Efeitos do projeto nas heranças

- Foram estabelecidas heranças entre as seguintes classes: CTrajDirecional herdou CTrajVertical, CParametrosDirecionais, CTrajDirecionalBuild e CTrajDirecionalSlant.

5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do software se relacionam, suas dependências. Inclui itens como: componentes, subsistemas, executáveis, nós, associações, dependências, generalizações, restrições e notas.

Veja na Figura 5.1 um exemplo de diagrama de componentes. Observe que este inclui dependências, ilustrando as relações entre os arquivos. O software executável a ser gerado depende da biblioteca gerada e do módulo de arquivos.

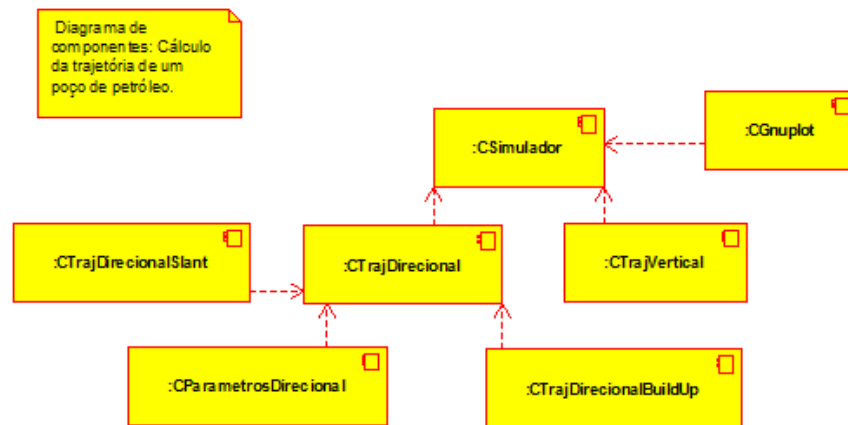


Figura 5.1: Diagrama de componentes

5.4 Diagrama de implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução. O diagrama de implantação deve incluir os elementos necessários para que o sistema seja colocado em funcionamento: computador, periféricos, processadores, dispositivos, nós, relacionamentos de dependência, associação, componentes, subsistemas, restrições e notas.

Veja na Figura 5.2 um diagrama de implantação.

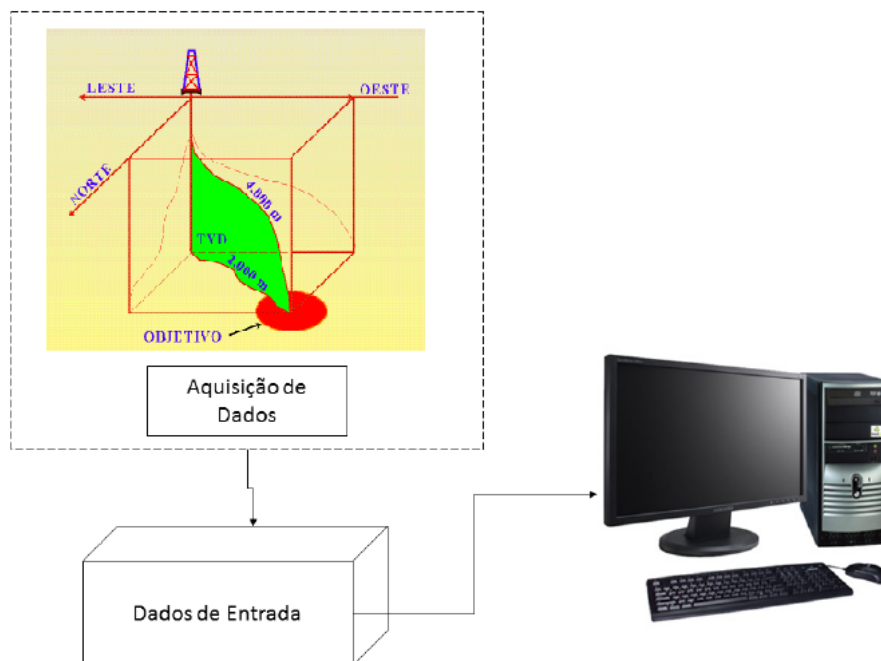


Figura 5.2: Diagrama de implantação.

Capítulo 6

Implementação

Neste capítulo do projeto de engenharia apresentamos os códigos fonte que foram desenvolvidos.

6.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa main.

Apresenta-se na listagem 6.1 o arquivo com código da classe CSimulador.

Listing 6.1: Arquivo de cabeçalho da classe CSimulador.h

```
1 #ifndef CSIMULADOR_H
2 #define CSIMULADOR_H
3
4 #include "CTrajVertical.h"           ///O SIMULADOR REUNIE TODAS AS
   CLASSES E ACESSA TODAS PARA CHAMAR OS RESULTADOS
5 #include "CParametrosDirecional.h"
6 #include "CTrajDirecionalBuildUp.h"
7 #include "CTrajDirecionalSlant.h"
8 #include "CTrajDirecional.h"
9 #include "CGnuplot.h"
10
11
12 class CSimulador
13 {
14     public:
15
16     CSimulador() { ///CONSTRUTOR DO SIMULADOR E DOS OBJETOS
17         trajvert    = new CTrajVertical;
18         trajdir      = new CParametrosDirecional;
19         buildup      = new CTrajDirecionalBuildUp(trajdir);
20         slant        = new CTrajDirecionalSlant(trajdir);
21         direcional   = new CTrajDirecional(trajdir, buildup, slant);
22     };
```

```

23
24     virtual ~CSimulador() {          ///DESTRUTOR DE TODOS OS OBJETOS
        JUNTO COM A CLASSE
25         delete trajvert;
26         delete trajdir;
27         delete buildup;
28         delete slant;
29         delete direcional;
30     };
31
32     void Simular(); ///METODO DA CLASSE
33
34     char tipo; ///VARIABEL UTILIZADA PARA DETERMINAR O TIPO DO POÇO
35
36     CGnuplot plotar4; ///OBJETOS UTILIZADOS PARA CRIAR OS TRES
        GRAFICOS
37     CGnuplot plotar1;
38     CGnuplot plotar2;
39     CGnuplot plotar3;
40
41     CTrajVertical*      trajvert; /// PONTEIROS CRIADOS QUE
        APONTAM PARA OS OBJETOS DAS CLASSES
42     CParametrosDirecional* trajdir;
43     CTrajDirecionalBuildUp* buildup;
44     CTrajDirecionalSlant*  slant;
45     CTrajDirecional*      direcional;
46 };
47
48 #endif ///CSIMULADOR_H

```

Apresenta-se na listagem 6.2 o arquivo de implementação da classe CSimulador.

Listing 6.2: Arquivo de implementação da classe CSimulador.cpp

```

49 #include "CSimulador.h"
50
51 #include <iostream>
52 #include <iomanip>
53 using namespace std;
54
55 void CSimulador::Simular(){
56     ///CABEÇALHO DO NOSSO PROGRAMA
57     cout << "
        -----
        " << endl;
58     cout << "░░░░░░░░░░Universidade░Estadual░do░Norte░Fluminense░Darcy░
        Ribeiro░░░░░░░░░░░░░░░░" << endl;
59     cout << "░░░░░░░░░░Laboratorio░de░Engenharia░e░Exploracao░de░
        Petroleo░░░░░░░░░░░░░░░░░░░░" << endl;

```

```

60     cout << "SIMULADOR PARA CALCULO DA TRAJETORIA DE POCOS DE
        PETROLEO" << endl;
61     cout << "Desenvolvido por: Antonio Jose dos Reis Neto e Tatiana
        Vitorio Isidorio" << endl;
62     cout << "
        -----
        " << endl;
63
64     cout << "Entre com a letra V para pouco vertical e D para direcional:
        "; ///AQUI O USUÁRIO ESCOLHE O TIPO DO POÇO
65     cin >> tipo;
66
67     while (tipo != 'V' && tipo != 'v' && tipo != 'D' && tipo != 'd') ///
        MENSAGEM DE ERRO CASO O USUARIO DIGITAR ALGO NAO
68     {
        ESPERADO
69         cout<< "Opcao invalida." << endl;
70         cout<< "Entre com V para pouco vertical e D para direcional:
        ";
71         cin>> tipo;
72     }
73
74     if (tipo == 'V' || tipo == 'v') ///CASO O USUARIO ESCOLHA V PARA
        POÇO VERTICAL, ENTRAMOS NESSA ROTINA
75     {
76         trajvert->EntradaDadosVertical(); ///entra atributos
        independentes e calcula atributos dependentes
77         trajvert->SaidaDadosVertical();
78
79         plotar4 << "set term png\n"; /// NESTA PARTE DO NOSSO
        CODIGO GERAMOS OS GRAFICOS NECESSARIOS PARA ANALISE DOS
        RESULTADOS
80         plotar4 << "set out \"MDxINC_Verical.png\" \n";
81         plotar4.XRange(-5,90); plotar4.XLabel("
        Inclinação");
82         plotar4.YRange(trajvert->tvd, 0); plotar4.YLabel("
        Profundidade Medida");
83         plotar4.Style("lines");
84         plotar4.Title("MDxINC");
85         plotar4.Grid(1);
86         plotar4.plotfile_xy("trajetoriaVertical.txt", 3,1);
87     }
88     else ///SE ESCOLHER UM POÇO DIRECIONAL, ENTRAMOS NESSA ROTINA
89     {
90         trajdir->EntradaDadosDirecional(); ///entra atributos
        independentes e calcula atributos dependentes
91         trajdir->SaidaDadosDirecional();
92         buildup->SaidaDadosBuild();

```



```

93         slant->SaidaDadosSlant();
94         direcional->CriarVetorCompleto();
95         direcional->SaidaDados();
96
97         plotar1 << "set_term.png\n";      ///NESTA PARTE DO NOSSO
          CODIGO GERAMOS OS GRÁFICOS NECESSÁRIOS PARA ANÁLISE DOS
          RESULTADOS
98         plotar1 << "set_out_\\"MDxTVD.png\\"\\n";
99         plotar1.XRange(0, trajdir->tvd);          plotar1.XLabel("
          Profundidade_Vertical");
100        plotar1.YRange(trajdir->md_total, 0);    plotar1.YLabel("
          Profundidade_Medida");
101        plotar1.Style("lines");
102        plotar1.Title("MD_x_TVD");
103        plotar1.Grid(1);
104        plotar1.plotfile_xy("trajetoriaDirecional.txt", 2,1);
105
106        plotar2 << "set_term.png\n";
107        plotar2 << "set_out_\\"MDxINC.png\\"\\n";
108        plotar2.XRange(-5,90);                  plotar2.XLabel("
          Inclinação");
109        plotar2.YRange(trajdir->md_total,0);      plotar2.YLabel("
          Profundidade_Medida");
110        plotar2.Style("lines");
111        plotar2.Title("MD_x_INC");
112        plotar2.Grid(1);
113        plotar2.plotfile_xy("trajetoriaDirecional.txt", 3,1);
114
115        plotar3 << "set_term.png\n";
116        plotar3 << "set_out_\\"MDxAZM.png\\"\\n";
117        plotar3.XRange(0,360);                  plotar3.XLabel("
          Azimute");
118        plotar3.YRange(trajdir->md_total, 0);    plotar3.YLabel("
          Profundidade_Medida");
119        plotar3.Style("lines");
120        plotar3.Title("MD_x_AZM");
121        plotar3.Grid(1);
122        plotar3.plotfile_xy("trajetoriaDirecional.txt", 4,1);
123
124    }
125}

```

Apresenta-se na listagem 6.3 o arquivo com código da classe CTrajVertical.

Listing 6.3: Arquivo de cabeçalho da classe CTrajVertical.h

```

126 #ifndef CTRAJVERTICAL_H
127 #define CTRAJVERTICAL_H
128
129 #include <vector>

```

```

130
131 class CTrajVertical
132 {
133 public:
134
135     double wellhead[2]; ///DADOS DE ENTRADA DA CLASSE
136     double target[2];
137     double tvd;
138
139     double azm;          ///PARAMETROS CALCULADOS
140     double A;
141     double B;
142     double C;
143     int tamanho;
144
145     std::vector<double> tvdVertical;    /// Vetor que recebe a
146     profundidade vertical, ponto a ponto que será calculada
147     std::vector<double> azimuth;       /// Vetor que recebe o azimuth,
148     ponto a ponto que será calculado
149     std::vector<double> mdVertical;    /// Vetor que recebe a
150     profundidade medida, ponto a ponto que será calculada
151     std::vector<double> incVertical;    /// Vetor que recebe a
152     inclinação, constante em 0 na parte vertical
153
154     void TVDVertical(); ///METODOS DA CLASSE
155     void AZMVertical(); ///METODOS DA CLASSE
156     void MDVertical(); ///METODOS DA CLASSE
157     void INCVertical(); ///METODOS DA CLASSE
158
159     void EntradaDadosVertical();
160     void SaidaDadosVertical();
161 };
162
163 #endif /// CTRAJVERTICAL_H_INCLUDED

```

Apresenta-se na listagem 6.4 o arquivo de implementação da classe CTrajVertical.

Listing 6.4: Arquivo de implementação da classe CTrajVertical.cpp

```

163 #include "CTrajVertical.h"
164
165 #include <iostream>
166 #include <fstream>
167 #include <iomanip>
168 #include <cmath>
169 #include <vector>
170
171 using namespace std;

```

```

172
173 void CTrajVertical::EntradaDadosVertical(){ /// METODO QUE RECEBE OS
      DADOS NECESSARIOS PARA OS CALCULOS DO AZIMUTE E TVD
174     cout<< "Entre com a coordenada x da cabeca do poc, em UTM: ";
175     cin>> wellhead[0];
176     cout<< "Entre com a coordenada y da cabeca do poc, em UTM: ";
177     cin>> wellhead[1];
178
179     cout<< "Entre com as coordenada x do alvo geologico, em UTM: ";
180     cin>> target[0];
181     cout<< "Entre com as coordenada y do alvo geologico, em UTM: ";
182     cin>> target[1];
183
184     cout<< "Entre com a profundidade final vertical (TVD), em metros
      : ";
185     cin>> tvd;
186 }
187
188 void CTrajVertical::TVDVertical(){
189
190     tamanho = tvd/10;
191     if ((tvd - (tamanho*10))!= 0.0) /// LOGICA PARA EVITAR PASSO DE
      TEMPO SEM SER INTEIRO
192     {
193         tamanho = tamanho + 1;
194     }
195
196     tamanho = tamanho + 1;
197
198     double salto_tvd = tvd/(tamanho-1);
199
200     for (int i=0; i < tamanho; i++) /// CRIACAO DO VETOR TVD,
      DIVIDINDO O TVD TOTAL, PELO ESPAÇO EM (m) QUE QUEREMOS
201     {
202         tvdVertical.push_back(i*salto_tvd);
203     }
204 }
205
206 void CTrajVertical::AZMVertical(){ ///CALCULO DO PARAMETRO AZIMUTE DE
      ACORDO COM A POSICAO DA SONDA E DO ALVO
207
208     this->A = target[1] - wellhead[1];
209     B = target[0] - wellhead[0];
210     C = A/B;
211     azm = atan(C) * (180/M_PI);
212
213     for (int i = 0; i < tamanho; i++) /// PREENCHENDO O VETOR
214     {

```

```

215         azimuth.push_back(azm);
216     }
217 }
218
219 void CTrajVertical::MDVertical(){    /// COMO O POÇO E VERTICAL, O MD E
    IGUAL QUE O TVD
220     mdVertical = tvdVertical;
221 }
222
223 void CTrajVertical::INCVertical(){ /// COMO O POÇO E VERTICAL, NÃO TEM
    INCLINAÇÃO. ASSIM, O VETOR E PREENCHIDO COM ZEROS
224     for (int i=0; i < tamanho; i++)
225     {
226         incVertical.push_back(0.00);
227     }
228 }
229
230 void CTrajVertical::SaidaDadosVertical(){
231
232     TVDVertical();    /// CHAMANDO TODOS OS METODOS
233     AZMVertical();
234     MDVertical();
235     INCVertical();
236
237     cout << endl;
238     cout << "
        =====
        " << endl;
239     cout.width(10);    /// CONFIGURANDO A SAIDA E FAZENDO O CABEÇALHO
240     cout << "MD(m)\t" << "TVD(m)\t" << "INC(°)\t" << "AZM
        AZM(°)\t" << endl;
241
242     for (int i=0; i < tamanho; i++)    ///
        GERANDO A SAIDA DE UMA FORMA ORGANIZADA
243     {
244         cout << setiosflags (ios::fixed) << setprecision(2);    ///
            PRECISAO DE DUAS CASAS DECIMAIS
245         cout.width(8);
246         cout << mdVertical[i] << right << "\t";    ///
            ALINHAMENTO A DIREITA COM TABULACAO
247
248         cout.width(8);
249         cout << tvdVertical[i] << right << "\t";
250
251         cout.width(4);
252         cout << incVertical[i] << right << "\t";
253
254         cout.width(4);

```

```

255         cout << azimuth[i] << right << endl;
256     }
257
258     fstream fout2;          ///ARQUIVO COM A TABELA DOS RESULTADOS FINAIS DO
                             PROGRAMA
259     fout2.open("trajetoriaVertical.txt", ios::app);
260     fout2 << setw(50) << "Vetores das trajetorias calculados" << endl;
261     fout2 << setiosflags (ios::fixed) << setprecision(2);
262     fout2 << setw(12) << "MD(m)\t" << setw(12) << "TVD(m)\t" << setw
        (12) << "INC(°)\t" << setw(10) << "AZM(°)\t" << endl;
263     for (int i = 0; i<tvdVertical.size(); i++){
264         fout2 << setw(10) << mdVertical[i] << '\t' << setw(10) <<
            tvdVertical[i] << '\t' << setw(10) << incVertical[i] << '\t'
            << setw(10) << azimuth[i] << '\t' << setw(10) << endl;
265     }
266 }

```

Apresenta-se na listagem 6.5 o arquivo com código da classe CParametrosDirecional.

Listing 6.5: Arquivo de cabeçalho da classe CParametrosDirecional.h

```

268 #ifndef CPARAMETROSDIRECIONAL_H
269 #define CPARAMETROSDIRECIONAL_H
270
271 #include "CTrajVertical.h"
272
273 class CParametrosDirecional {
274 public:
275
276     double kop; /// DECLARACAO DE TODOS OS ATRIBUTOS QUE SERAO
                UTILIZADOS
277     double bur;
278     double tvd;
279     double wellhead[2];
280     double target[2];
281     double r;
282     double afast_alvo;
283     double angMax;
284     double prof_eob;
285     double afast_eob;
286     double comp_build;
287     double comp_slant;
288     double K;
289     double md_total;
290
291     /// OCORRE A REPETICAO, POIS AO FINAL QUANDO E ESCOLHIDO POÇO
        DIRECIONAL, TAMBEM PRECISAMOS
292     /// DA PARTE VERTICAL DO POÇO, E NA CLASSE VERTICAL, E CALCULADO
        APENAS QUANDO O USUARIO
293     /// ESCOLHE QUE DESEJA UM POÇO TOTALMENTE VERTICAL

```

```

294
295     double azm; ///PARAMETROS UTILIZADOS NOS CALCULOS DA SECAO VERTICAL
296     double A;
297     double B;
298     double C;
299     int tamanho;
300
301     std::vector<double> tvdVertical; ///VETORES VERTICAIS
302     std::vector<double> azimuth;
303     std::vector<double> mdVertical;
304     std::vector<double> incVertical;
305
306     void TVDVertical(); ///METODOS VERTICAIS
307     void AZMVertical();
308     void MDVertical();
309     void INCVertical();
310
311     void RaioCurvatura(); ///METODOS DA CLASSE QUE CALCULAM OS
        PARÂMETROS DIRECIONAIS UTILIZADOS NAS TRAJETORIAS
312     void AfastamentoAlvo();
313     void AnguloMaximo();
314     void AfastamentoEob();
315     void ProfEob();
316     void CompSlant();
317     void CompBuild();
318     void MD();
319
320     void EntradaDadosDirecional(); ///METODO RECEBE TODOS OS DADOS
321     void SaidaDadosDirecional(); ///METODO MOSTRA NO CONSOLE TODOS
        OS PARAMETROS CALCULADOS
322
323 };
324 #endif ///CPARAMETROSDIRECIONAL_H_INCLUDED

```

Apresenta-se na listagem 6.6 o arquivo de implementação da classe CParametrosDirecional.

Listing 6.6: Arquivo de implementação da classe CParametrosDirecional.cpp

```

325 #include "CParametrosDirecional.h"
326
327 #include <iostream>
328 #include <iomanip>
329 #include <cmath>
330 #include <vector>
331
332 using namespace std;
333
334 void CParametrosDirecional::EntradaDadosDirecional(){ ///METODO QUE
    PEDE TODOS OS DADOS DE ENTRADA NECESSARIOS
335

```

```

336     cout << "Entre com a coordenada x da cabeca do poc, em UTM: ";
337     cin >> wellhead[0];
338     cout << "Entre com a coordenada y da cabeca do poc, em UTM: ";
339     cin >> wellhead[1];
340
341     cout << "Entre com as coordenada x do alvo geologico, em UTM: ";
342     cin >> target[0];
343     cout << "Entre com as coordenada y do alvo geologico, em UTM: ";
344     cin >> target[1];
345
346     do {
347         cout << "Entre com a profundidade final vertical (TVD), em metros: ";
348         cin >> tvd;
349
350         cout << "## Lembre que KOP precisa ser menor que TVD. ##" << endl;
351         cout << "Entre com a profundidade inicial da secao de ganho de angulo (KOP), em metros: ";
352         cin >> kop;
353
354     } while (kop > tvd);
355
356     cout << "Entre com a taxa de ganho de angulo (BUR), em graus/30m: ";
357     cin >> bur;
358 }
359
360 void CParametrosDirecional::TVDVertical() {           /// CALCULO VERTICAL
361
362     tamanho = kop/10;
363     if ((kop - (tamanho*10)) != 0.0)
364     {
365         tamanho = tamanho + 1;
366     }
367
368     tamanho = tamanho + 1;
369
370     double salto_kop = kop/(tamanho-1);
371
372     for (int i=0; i < tamanho; i++)
373     {
374         tvdVertical.push_back(i*salto_kop);
375     }
376 }
377
378 void CParametrosDirecional::AZMVertical() {           /// CALCULO VERTICAL
379

```

```

380     this->A = target[1] - wellhead[1];
381     B = target[0] - wellhead[0];
382     C = A/B;
383     azm = atan(C) * (180/M_PI);
384
385     for (int i = 0; i < tamanho; i++)
386     {
387         azimuth.push_back(azm);
388     }
389 }
390
391 void CParametrosDirecional::MDVertical() {           /// CALCULO VERTICAL
392     mdVertical = tvdVertical;
393 }
394
395 void CParametrosDirecional::INCVertical() {           /// CALCULO VERTICAL
396     for (int i=0; i < tamanho; i++)
397     {
398         incVertical.push_back(0.00);
399     }
400 }
401
402 void CParametrosDirecional::RaioCurvatura() {       ///PARAMETRO QUE CALCULA
403     cout << endl;                                     O RAO DA CURVATURA GERADO PELA INCLINACAO DO POCO
404     cout.width(85);
405     cout << "
406         =====
407         ";
408     cout << endl;
409
410     r = (180./M_PI)*(30./bur);
411
412     cout.width(30);
413     cout << left << "Raio de curvatura(m)";
414     cout << r << right << endl;
415 }
416
417 void CParametrosDirecional::AfastamentoAlvo() {    /// O AFASTAMENTO E
418     CALCULADO A PARTIR DAS COORDENADAS DA Sonda E DO ALVO
419     double A;
420     double B;
421
422     A = target[0] - wellhead[0];
423     B = target[1] - wellhead[1];
424     afast_alvo = pow((pow(A,2)+ pow(B,2)),0.5);
425
426     cout.width(30);

```



```

424     cout << left << "Afastamento_␣(m)␣";
425     cout << afast_alvo << right << endl;
426
427 }
428
429 void CParametrosDirecional::AnguloMaximo(){      /// O ANGULO MAXIMO
        REPRESENTA A MAXIMA INCLINACAO QUE O POÇO IRA ALCANCAR
430
431     double C;      /// VARIÁVEIS LOCAIS CRIADAS PARA FACILITAR OS CALCULOS
        DESTE METODO
432     double D;
433     double E;
434     double F;
435     double G;
436     double H;
437     double I;
438     double J;
439     double L;
440
441     C = r - afast_alvo; /// O CALCULO DO ANGULO MAXIMO SAI POR RELACOES
        TRIGONOMETRICAS
442     D = tvd - kop;
443     E = C/D;
444     F = atan(E);
445     G = (F*180)/M_PI;
446
447     cout.width(30);
448     cout << left << "Tal_␣(°)";
449     cout << G << right << endl;
450
451     H = pow((pow(C,2)+ pow(D,2)),0.5);
452     I = r/H;
453     J = asin(I);
454     L = (J*180)/M_PI;
455
456     cout.width(30);
457     cout << left << "Omega_␣(°)␣";
458     cout << L << right << endl;
459
460     angMax = L - G;
461
462     cout.width(30);
463     cout << left << "Angulo_␣maximo_␣(°)␣";
464     cout << angMax << right << endl;
465 }
466
467 void CParametrosDirecional::AfastamentoEob(){      /// ESTE METODO CALCULA
        O AFASTAMENTO HORIZONTAL NO FINAL DA SECAO DE GANHO DE ANGULO

```

```

468     K = (angMax*M_PI)/180;
469     afast_eob = r*(1- cos(K));
470
471     cout.width(30);
472     cout << left << "Afastamento_(EOB)_(m)";
473     cout << afast_eob << right <<endl;
474 }
475
476 void CParametrosDirecional::ProfEob(){          /// PROFUNDIDADE
    VERTICAL NO FINAL DA SECAO DE GANHO DE ANGULO
477     prof_eob = (kop + r*sin(K));
478
479     cout.width(30);
480     cout << left << "Profundidade_(EOB)_(m)";
481     cout << prof_eob << right <<endl;
482 }
483
484 void CParametrosDirecional::CompBuild(){        /// ESTE METODO E
    RESPONSAVEL POR CALCULAR O COMPRIMENTO DO ARCO GERADO NA SECAO DE
    GANHO DE ANGULO
485     comp_build = (30*angMax)/bur;
486
487     cout.width(30);
488     cout << left << "Comprimento_da_secao_build_(m)";
489     cout << comp_build << right <<endl;
490 }
491
492 void CParametrosDirecional::CompSlant(){        /// COMPRIMENTO DA SECAO
    TANGENTE DO POÇO APOS O FINAL DA SECAO DE GANHO DE ANGULO ATE O ALVO
493     comp_slant = (tvd-prof_eob)/cos(K);
494
495     cout.width(30);
496     cout << left << "Comprimento_da_secao_slant_(m)";
497     cout << right << comp_slant <<endl;
498 }
499
500 void CParametrosDirecional::MD(){              /// METODO RESPONSAVEL
    POR CALCULAR A PROFUNDIDADE MEDIDA TOTAL DO POÇO
501     md_total = kop + comp_build + comp_slant;
502
503     cout.width(30);
504     cout << left << "MD_Final_(m)";
505     cout << md_total << right <<endl;
506 }
507
508 void CParametrosDirecional::SaidaDadosDirecional(){
509
510     RaioCurvatura();      /// APOS REALIZAR TODOS OS CALCULOS, ESTE METODO

```

```

                JOGA OS RESULTADOS NA TELA
511     AfastamentoAlvo();
512     AnguloMaximo();
513     AfastamentoEob();
514     ProfEob();
515     CompBuild();
516     CompSlant();
517     MD();
518     TVDVertical();
519     AZMVertical();
520     MDVertical();
521     INCVertical();
522 }

```

Apresenta-se na listagem 6.7 o arquivo com código da classe CTrajDirecionalBuildUp.

Listing 6.7: Arquivo de cabeçalho da classe CTrajDirecionalBuildUp.h

```

523 #ifndef CTRAJDIRECIONALBUILDUP_H
524 #define CTRAJDIRECIONALBUILDUP_H
525
526 #include "CParametrosDirecional.h" ///ESSA CLASSE CRIA A TRAJETORIA DA
    SECAO DE GANHO DE ANGULO, PARA ISSO PRECISAMOS
527 #include <vector> ///UTILIZAR OS PARAMETROS DIRECIONAIS
    CALCULADOS NA CLASSE "CPARAMETROSDIRECIONAL"
528
529 class CTrajDirecionalBuildUp{
530     public:
531
532     double kop; ///AQUI CHAMAMOS OS PARAMETROS CALCULADOS NA OUTRA
        CLASSE QUE SERAO UTILIZADOS NO CALCULO DOS VETORES
533     double bur;
534     double r;
535     double angMax;
536     double comp_build;
537     double md_total;
538     double tvd;
539     int tamanho1;
540
541     std::vector<double> tvd_vetor; ///VETORES QUE SERAO CRIADOS
        NESSA CLASSE PARA REPRESENTAR A TRAJETORIA DE GANHO DE ANGULO
542     std::vector<double> inclination;
543     std::vector<double> md_vetor;
544
545     CParametrosDirecional* parametroDirecional; ///CONSTRUTOR
        CHAMANDO POR PONTEIRO OS OBJETOS CRIADOS
546     CTrajDirecionalBuildUp (CParametrosDirecional*
        _parametroDirecional) : parametroDirecional(
        _parametroDirecional) {};
547

```

```

548     void MdBuild();          ///METODOS DA CLASSE
549     void IncBuild();
550     void TvdBuild();
551     void SaidaDadosBuild();
552 };
553
554 #endif ///CTRAJDIRECIONALBUILDUP_H

```

Apresenta-se na listagem 6.8 o arquivo de implementação da classe CTrajDirecionalBuildUp.

Listing 6.8: Arquivo de implementação da classe CTrajDirecionalBuildUp.cpp

```

556 #include "CTrajDirecionalBuildUp.h"
557
558 #include <iostream>
559 #include <cmath>
560 #include <vector>
561
562 using namespace std;
563
564 void CTrajDirecionalBuildUp::IncBuild(){          ///CRIACAO DO
    METODO QUE CALCULA O VETOR DE INCLINACOES
565
566     inclination.push_back(0);                      ///PRIMEIRO VALOR DO
    VETOR E 0, PORQUE E INCLINAÇÃO IGUAL A 0
567     int cont = 0;                                  ///CRIAMOS UM
    CONTADOR E INICIAMOS ELE COM O VALOR 0
568
569     while ((inclination[cont]+parametroDirecional->bur) <=
        parametroDirecional->angMax) ///LOGICA DA CRIACAO DESTA VETOR
570     {
        ///A CADA PASSO DO CONTADOR A INCLINACAO
571         inclination.push_back(inclination[cont] +
            parametroDirecional->bur);          ///MUDA COM BUR, ATE O
            ANGULO MAXIMO
572         cont++;
573     }
574     cont++;
575     inclination[cont] = parametroDirecional->angMax;    ///ULTIMO VALOR
    DO NOSSO VETOR TEM QUE SER O ANGULO MAXIMO
576 }
577
578 void CTrajDirecionalBuildUp::TvdBuild(){          ///A
    PROFUNDIDADE VERTICAL E CALCULADA UTILIZANDO O RAO DE CURVATURA
579
    ///SÃO RELAÇÕES
    TRIGONOMÉTRICAS
    PARA
    ENCONTRAR A
    PARTE

```

```

                                                    VERTICAL
580
581     tvd_vetor.push_back(parametroDirecional->kop+10);    ///PARA NAO
        SOBREPOR VALORES, INICIAMOS ESTE VETOR 10M DEPOIS DO
582     int cont = 0;                                        ///FINAL DO
        VETOR TVD DA PARTE VERTICAL
583
584     while ((inclination[cont]+parametroDirecional->bur) <=
        parametroDirecional->angMax)
585     {
586         tvd_vetor.push_back(parametroDirecional->kop + (
            parametroDirecional->r * sin((inclination[cont+1]*M_PI)
            /180)));
587         cont++;
588     }
589     cont++;
590     tvd_vetor[cont] = parametroDirecional->kop + (parametroDirecional->r
        * sin((parametroDirecional->angMax*M_PI)/180));
591 }
592
593 void CTrajDirecionalBuildUp::MdBuild() {                ///COMO NA SECAO DE
        GANHO DE ANGULO O NOSSO POÇO SAI DA VERTICAL,
594
        ///PRECISAMOS
        CALCULAR A
        PROFUNDIDADE
        MEDIDA DELE, PARA
        SABER EXATAMENTE
595
        ///A PROFUNDIDADE
        PERCORRIDA PELO
        POÇO
596
597     double A = parametroDirecional->kop + parametroDirecional->
        comp_build; ///SOMANDO A PROF. DE KOP COM O COMPRIMENTO
598     int cont = 0;
                                                    ///DA
        SEÇÃO DE GANHO, OBTEMOS O MD ATE O FINAL
599
                                                    //
                                                    /
                                                    DA
                                                    FAS
600
601     md_vetor.push_back(parametroDirecional->kop+(10/cos((
        parametroDirecional->angMax*M_PI)/180)));    ///PARA NÃO SOBREPOR,
        AUMENTAMOS 10m, PORÉM COMO E INCLINADO, UTILIZAMOS O COSENO PARA
        AUMENTAR ESTE PASSO DE PROF.
602

```

```

603     while ((md_vetor[cont] + (parametroDirecional->comp_build/
        inclination.size())) <= A)          ///AQUI UTILIZAMOS O NUMERO DE
        CASAS DO VETOR INCLINAÇÃO PARA LIMITAR ESSE VETOR, POIS TODOS
        VETORES PRECISAM ESTAR
604
605     {
606         md_vetor.push_back(md_vetor[cont] + (parametroDirecional->
            comp_build/inclination.size()));
607         cont++;
608     }
609     cont++;
610     md_vetor[cont] = A; ///GARANTIA QUE O ULTIMO VALOR DO VETOR E O MD
        FINAL DESTA FASE
611 }
612
613 void CTrajDirecionalBuildUp::SaidaDadosBuild() {          ///EXIBE NO CONSOLE
    OS RESULTADOS
614
615     IncBuild();          ///APOS PREENCHER TODOS OS VETORES JOGAMOS NA TELA
        PARA CONFERIR OS RESULTADOS
616     TvdBuild();
617     MdBuild();
618
619 }
```

Apresenta-se na listagem 6.9 o arquivo com código da classe CTrajDirecionalSlant.

Listing 6.9: Arquivo de cabeçalho da classe CTrajDirecionalSlant.h

```

621 #ifndef CTRAJDIRECIONALSLANT_H
622 #define CTRAJDIRECIONALSLANT_H
623
624 #include "CTrajVertical.h"
625 #include "CParametrosDirecional.h"
626 #include <vector>
627
628 class CTrajDirecionalSlant {
629 public:
630
631     double kop;          ///TAMBEM UTILIZAMOS OS PARAMETROS CALCULADOS NA
        CLASSE "CPARAMETROSDIRECIONAL"
632     double angMax;
633     double comp_build;
634     double md_total;
635     double prof_eob;
636     double tvd;
637     double inc_slant;
638
639     std::vector<double> tvd_vetor_slant;      ///VETORES QUE SAO
        CRIADOS NESSA CLASSE
640     std::vector<double> inclination_slant;
641     std::vector<double> md_vetor_slant;
642
643     CParametrosDirecional* parametroDirecional;      ///CONSTRUTOR
        RECEBENDO POR PONTEIRO AS INFORMACOES DA CLASSE "
        CPARAMETROSDIRECIONAL"
644     CTrajDirecionalSlant (CParametrosDirecional* _parametroDirecional) :
        parametroDirecional(_parametroDirecional) {};
645
646     void MdSlant();      ///METODOS DA CLASE
647     void IncSlant();
648     void TvdSlant();
649     void SaidaDadosSlant();
650
651 };
652
653 #endif /// CTRAJDIRECIONALSLANT_H_INCLUDED

```

Apresenta-se na listagem 6.10 o arquivo de implementação da classe CTrajDirecionalSlant.

Listing 6.10: Arquivo de implementação da classe CTrajDirecionalSlant.cpp

```

655 #include "CTrajDirecionalSlant.h"
656
657 #include <iostream>
658 #include <cmath>
659 #include <vector>

```

```

660
661 using namespace std;
662
663 void CTrajDirecionalSlant::MdSlant(){           ///METODO QUE CALCULA O
        VECTOR DA PROFUNDIDADE MEDIDA PARA A SECAO TANGENTE
664     double A = parametroDirecional->kop + parametroDirecional->
        comp_build;
665     int cont = 0;
666     md_vetor_slant.push_back(A+(10/cos((parametroDirecional->angMax*M_PI
        )/180)));
667
668     while ((md_vetor_slant[cont] + ((parametroDirecional->md_total - A)
        /10/cos((parametroDirecional->angMax*M_PI)/180)) <=
        parametroDirecional->md_total))
669     {
670         md_vetor_slant.push_back(md_vetor_slant[cont] + (
        parametroDirecional->md_total-A)/10/cos((
        parametroDirecional->angMax*M_PI)/180));
671         cont++;
672     }
673     cont++;
674     md_vetor_slant[cont] = parametroDirecional->md_total; ///A
        PROFUNDIDADE MEDIDA FINAL FOI CALCULADA EM PARAMETROSDIRECIONAL
675 }
676
677 void CTrajDirecionalSlant::IncSlant(){           ///COMO ESTAMOS NA SECAO
        TANGENTE DO POÇO, A INCLINACAO PERMANECE CONSTANTE
678     inc_slant = parametroDirecional->angMax;      ///COM O VALOR DO ANGULO
        MAXIMO
679     for (int i=0; i <= md_vetor_slant.size(); i++)
680     {
681         inclination_slant.push_back(inc_slant);
682     }
683 }
684
685 void CTrajDirecionalSlant::TvdSlant(){           ///A PROFUNDIDADE
        VERTICAL FINAL E CALCULADA POR RELACOES TRIGONOMETRICAS
686     int cont = 1;                                ///O VALOR FINAL TEM QUE
        BATER COM O TVD QUE O USUÁRIO ENTROU
687     tvd_vetor_slant.push_back(parametroDirecional->prof_eob+10);
688
689     for(cont; cont<= md_vetor_slant.size();cont++)
690     {
691         tvd_vetor_slant.push_back((md_vetor_slant[cont] - md_vetor_slant
        [0])*cos((parametroDirecional->angMax*M_PI)/180)+
        tvd_vetor_slant[0]);
692     }
693 }

```



```

694
695 void CTrajDirecionalSlant::SaidaDadosSlant(){    ///METODO QUE JOGA PARA
        A TELA TODOS OS VETORES CALCULADOS
696     MdSlant();
697     IncSlant();
698     TvdSlant();
699 }

```

Apresenta-se na listagem 6.11 o arquivo com código da classe CTrajDirecional.

Listing 6.11: Arquivo de cabeçalho da classe CTrajDirecional.h

```

701 #ifndef CTRAJDIRECIONAL_H_INCLUDED
702 #define CTRAJDIRECIONAL_H_INCLUDED
703
704 #include "CParametrosDirecional.h"    ///CLASSE CRIADA PARA CONCATENAR
        TODOS OS VETORES CRIADOS E MOSTRAR O RESULTADO DE TODA A TRAJETÓRIA
        DO POÇO
705 #include "CTrajDirecionalBuildUp.h"    ///PARA ISSO PRECISAMOS INCLUIR
        TODAS AS CLASSES
706 #include "CTrajDirecionalSlant.h"
707
708 class CTrajDirecional
709 {
710 public:
711
712     std::vector<double> azimuth;        ///VETORES DA SECAO VERTICAL
713     std::vector<double> tvdVertical;
714     std::vector<double> mdVertical;
715
716     std::vector<double> tvd_vetor;      ///VETORES DA SECAO BUILD (
        GANHO DE ANGULO)
717     std::vector<double> inclination;
718     std::vector<double> md_vetor;
719
720     std::vector<double> tvd_vetor_slant; ///VETORES DA SECAO SLANT (
        TANGENTE)
721     std::vector<double> inclination_slant;
722     std::vector<double> md_vetor_slant;
723
724     std::vector<double> AZM;            ///VETORES CONCATENADOS,
        EXIBINDO OS RESULTADOS DA TRAJETÓRIA COMPLETA DO POÇO
725     std::vector<double> INC;
726     std::vector<double> TVD;
727     std::vector<double> MD;
728
729     void CriarVetorCompleto();          ///METODOS DA CLASSE
730     void SaidaDados();
731

```

```

732     CParametrosDirecional* trajdir;          ///OBJETOS CRIADOS PARA
           UTILIZAR OS VETORES DAS OUTRAS CLASSES
733     CTrajDirecionalBuildUp* buildup;
734     CTrajDirecionalSlant* slant;
735     CTrajDirecional(CParametrosDirecional* _trajdir,
           CTrajDirecionalBuildUp* _buildup,CTrajDirecionalSlant* _slant) :
           trajdir(_trajdir), buildup(_buildup), slant(_slant) {};
736
737 };
738 #endif ///CTRAJDIRECIONAL_H_INCLUDED

```

Apresenta-se na listagem 6.12 o arquivo de implementação da classe CTrajDirecional.

Listing 6.12: Arquivo de implementação da classe CTrajDirecional.cpp

```

739 #include "CTrajDirecional.h"
740
741 #include <iostream>
742 #include <fstream> ///CRIAMOS ARQUIVOS EXTEMOS PARA SALVAR OS
           RESULTADOS E PARA CRIAR OS GRAFICOS
743 #include <cmath>
744 #include <vector>
745 #include <iterator> ///UTILIZAMOS PARA CONCATENAR OS VETORES
746 #include <iomanip>
747
748 using namespace std;
749
750 void CTrajDirecional::CriarVetorCompleto(){
751
752     for(int i = 0; i < trajdir->tvDVertical.size(); i++) ///VETOR TVD
753     {
754         TVD.push_back(trajdir->tvDVertical[i]);
755     }
756
757     for(int i = 0; i < buildup->tvD_vetor.size(); i++)
758     {
759         TVD.push_back(buildup->tvD_vetor[i]);
760     }
761     for(int i = 0; i < slant->tvD_vetor_slant.size(); i++)
762     {
763         TVD.push_back(slant->tvD_vetor_slant[i]);
764     }
765
766     for(int i = 0; i < trajdir->mdVertical.size(); i++) ///VETOR MD
767     {
768         MD.push_back(trajdir->mdVertical[i]);
769     }
770
771     for(int i = 0; i < buildup->md_vetor.size(); i++)
772     {

```

```

773         MD.push_back(buildup->md_vetor[i]);
774     }
775     for(int i = 0; i <= slant->md_vetor_slant.size(); i++)
776     {
777         MD.push_back(slant->md_vetor_slant[i]);
778     }
779
780     for(int i = 0; i < trajdir->incVertical.size(); i++)    ///VETOR INC
781     {
782         INC.push_back(trajdir->incVertical[i]);
783     }
784
785     for(int i = 0; i < buildup->inclination.size(); i++)
786     {
787         INC.push_back(buildup->inclination[i]);
788     }
789     for(int i = 0; i < slant->inclination_slant.size(); i++)
790     {
791         INC.push_back(slant->inclination_slant[i]);
792     }
793
794     for(int i = 0; i < TVD.size(); i++)    ///VETOR AZM
795     {
796         AZM.push_back(trajdir->azm);
797     }
798
799     cout << endl;
800     cout << "
=====
" << endl;
801
802     cout.width(10); ///AQUI JOGAMOS O RESULTADO NA TELA EM COLUNAS
803     cout << "MD_(m)_\t" << "____TVD_(m)_\t" << "___INC_(°)_\t" << "___
      AZM_(°)_\t" << endl;
804
805     for (int i=0; i < TVD.size(); i++)
806     {
807         cout << setiosflags (ios::fixed) << setprecision(2);
808         cout.width(8);
809         cout << MD[i] << right << "\t";
810
811         cout.width(8);
812         cout << TVD[i] << right << "\t";
813
814         cout.width(6);
815         cout << INC[i] << right << "\t";
816
817         cout.width(6);

```

```

818         cout << AZM[i] << right << endl;
819     }
820 }
821
822 void CTrajDirecional::SaidaDados(){
823
824     fstream fout;    ///ARQUIVO COM A TABELA DOS RESULTADOS FINAIS DO
                        PROGRAMA
825     fout.open("trajetoriaDirecional.txt", ios::app);
826     fout << setw(50) << "└Vetores└das└trajetorias└calculados" << endl;
827     fout << setiosflags (ios::fixed) << setprecision(2);
828     fout << setw(12) << "MD└(m)└\t" << setw(12) << "TVD└(m)└\t" << setw
        (12) << "INC└(°)└\t" << setw(10) << "AZM└(°)└└\t" << endl;
829     for (int i = 0; i<TVD.size(); i++){
830         fout << setw(10) << MD[i] << '\t' << setw(10) << TVD[i] << '\t'
            << setw(10) << INC[i] << '\t' << setw(10) << AZM[i] << '\t'
            << setw(10) << endl;
831     }
832 }

```

Apresenta-se na listagem 6.13 o arquivo com código da classe main.

Listing 6.13: Arquivo de cabeçalho da classe main.cpp

```

833 #include <iostream>
834 #include "CSimulador.h"
835
836
837 using namespace std;
838
839 int main()                ///COMO CRIAMOS A CLASSE "CSIMULADOR", NOSSO
                        MAIN PRECISA APENAS CRIAR UM OBJETO E
840                        ///ACESSAR O METODO SIMULAR
841 {
842     CSimulador poco;
843     poco.Simular();
844
845     return 0;
846 }

```

Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos alguns testes do software desenvolvido. Estes testes devem dar resposta aos diagramas de caso de uso inicialmente apresentados (diagramas de caso de uso geral e específicos).

7.1 Teste 1: Interface

O presente trabalho apresenta interface em modo texto. Veja na Figura 7.1 a tela inicial do software, caso o usuário opte pela trajetória vertical. Quando a opção for trajetória direcional, a interface fica da seguinte maneira, exemplificado pela Figura 7.2.

```
-----  
Universidade Estadual do Norte Fluminense Darcy Ribeiro  
Laboratorio de Engenharia e Exploracao de Petroleo  
SIMULADOR PARA CALCULO DA TRAJETORIA DE POCOS DE PETROLEO  
Desenvolvido por: Antonio Jose dos Reis Neto e Tatiana Vitorio Isidorio  
-----  
Entre com a letra V para poço vertical e D para direcional: v  
Entre com a coordenada x da cabeça do poço, em UTM: 0  
Entre com a coordenada y da cabeça do poço, em UTM: 0  
Entre com a coordenada x do alvo geológico, em UTM: 300  
Entre com a coordenada y do alvo geológico, em UTM: 400  
Entre com a profundidade final vertical (TVD), em metros: 2000
```

Figura 7.1: Tela do programa mostrando os dados de entrada necessários

```
-----  
Universidade Estadual do Norte Fluminense Darcy Ribeiro  
Laboratorio de Engenharia e Exploracao de Petroleo  
SIMULADOR PARA CALCULO DA TRAJETORIA DE POCOS DE PETROLEO  
Desenvolvido por: Antonio Jose dos Reis Neto e Tatiana Vitorio Isidorio  
-----  
Entre com a letra V para poco vertical e D para direcional: d  
Entre com a coordenada x da cabeca do poco, em UTM: 0  
Entre com a coordenada y da cabeca do poco, em UTM: 0  
Entre com as coordenada x do alvo geologico, em UTM: 300  
Entre com as coordenada y do alvo geologico, em UTM: 400  
Entre com a profundidade final vertical (TVD), em metros: 2000  
## Lembre que KOP precisa ser menor que TVD. ##  
Entre com a profundidade inicial da secao de ganho de angulo (KOP), em metros: 500  
Entre com a taxa de ganho de angulo (BUR), em graus/30m: 1
```

Figura 7.2: Tela do programa mostrando os dados de entrada necessários

7.2 Teste 2: Resultados

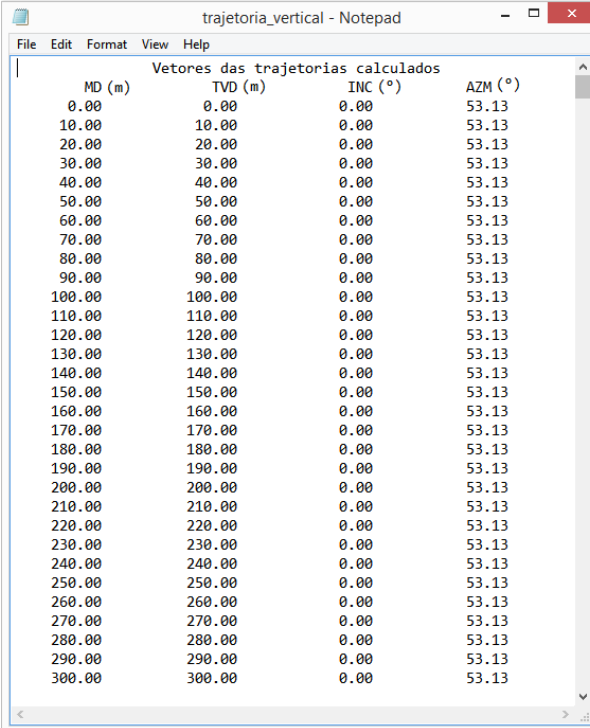
Neste teste foi introduzido os dados de entrada para o poço vertical e para o direcional. O software realiza os cálculos e exibe os seguintes resultados (Figura 7.3, 7.4). Além dos resultados exibidos no console, tanto para o poço vertical, como para o direcional, o software salva em disco arquivos “.txt” contendo o resultados dos vetores calculados, que são apresentados na Figura 7.5 e 7.6. Finalizando o teste seguem os gráficos gerados ao final dos cálculos (Figura 7.7, 7.8, 7.9 e 7.10).

MD (m)	TVD(m)	INC (°)	AZM (°)
0.00	0.00	0.00	53.13
100.00	100.00	0.00	53.13
200.00	200.00	0.00	53.13
300.00	300.00	0.00	53.13
400.00	400.00	0.00	53.13
500.00	500.00	0.00	53.13
600.00	600.00	0.00	53.13
700.00	700.00	0.00	53.13
800.00	800.00	0.00	53.13
900.00	900.00	0.00	53.13
1000.00	1000.00	0.00	53.13
1100.00	1100.00	0.00	53.13
1200.00	1200.00	0.00	53.13
1300.00	1300.00	0.00	53.13
1400.00	1400.00	0.00	53.13
1500.00	1500.00	0.00	53.13
1600.00	1600.00	0.00	53.13
1700.00	1700.00	0.00	53.13
1800.00	1800.00	0.00	53.13
1900.00	1900.00	0.00	53.13
2000.00	2000.00	0.00	53.13

Figura 7.3: Tela do programa mostrando os resultados da trajetória vertical

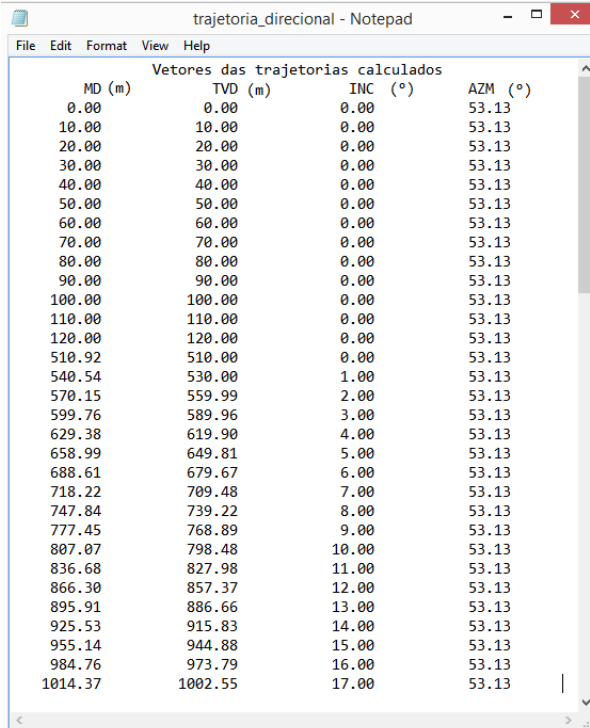
=====			
Raio de curvatura	1718.87	(m)	
Afastamento	500	(m)	
Tal	39.0967	(°)	
Omega	62.7886	(°)	
Angulo maximo	23.6919	(°)	
Afastamento (EOB)	144.868	(m)	
Profundidade (EOB)	1190.67	(m)	
Comprimento da secao build	710.757	(m)	
Comprimento da secao slant	883.814	(m)	
MD Final	2094.57	(m)	
=====			
MD (m)	TVD (m)	INC (°)	AZM (°)
0.00	0.00	0.00	53.13
100.00	100.00	0.00	53.13
200.00	200.00	0.00	53.13
300.00	300.00	0.00	53.13
400.00	400.00	0.00	53.13
500.00	500.00	0.00	53.13
510.92	510.00	0.00	53.13
540.54	530.00	1.00	53.13
570.15	559.99	2.00	53.13
599.76	589.96	3.00	53.13
629.38	619.90	4.00	53.13
658.99	649.81	5.00	53.13
688.61	679.67	6.00	53.13
718.22	709.48	7.00	53.13
747.84	739.22	8.00	53.13
777.45	768.89	9.00	53.13
807.07	798.48	10.00	53.13
836.68	827.98	11.00	53.13
866.30	857.37	12.00	53.13
895.91	886.66	13.00	53.13
925.53	915.83	14.00	53.13
955.14	944.88	15.00	53.13
984.76	973.79	16.00	53.13
1014.37	1002.55	17.00	53.13
1043.99	1031.16	18.00	53.13
1073.60	1059.61	19.00	53.13
1103.22	1087.89	20.00	53.13
1132.83	1115.99	21.00	53.13
1162.45	1143.90	22.00	53.13
1192.06	1171.62	23.00	53.13
1221.68	1200.67	23.69	53.13
1318.19	1289.06	23.69	53.13
1414.71	1377.44	23.69	53.13
1511.22	1465.82	23.69	53.13
1607.74	1554.20	23.69	53.13
1704.26	1642.58	23.69	53.13
1800.77	1730.96	23.69	53.13
1897.29	1819.34	23.69	53.13
1993.80	1907.73	23.69	53.13
2090.32	1996.11	23.69	53.13
2094.57	2000.00	23.69	53.13

Figura 7.4: Tela do programa mostrando os resultados da trajetória direcional



MD (m)	TVD (m)	INC (°)	AZM (°)
0.00	0.00	0.00	53.13
10.00	10.00	0.00	53.13
20.00	20.00	0.00	53.13
30.00	30.00	0.00	53.13
40.00	40.00	0.00	53.13
50.00	50.00	0.00	53.13
60.00	60.00	0.00	53.13
70.00	70.00	0.00	53.13
80.00	80.00	0.00	53.13
90.00	90.00	0.00	53.13
100.00	100.00	0.00	53.13
110.00	110.00	0.00	53.13
120.00	120.00	0.00	53.13
130.00	130.00	0.00	53.13
140.00	140.00	0.00	53.13
150.00	150.00	0.00	53.13
160.00	160.00	0.00	53.13
170.00	170.00	0.00	53.13
180.00	180.00	0.00	53.13
190.00	190.00	0.00	53.13
200.00	200.00	0.00	53.13
210.00	210.00	0.00	53.13
220.00	220.00	0.00	53.13
230.00	230.00	0.00	53.13
240.00	240.00	0.00	53.13
250.00	250.00	0.00	53.13
260.00	260.00	0.00	53.13
270.00	270.00	0.00	53.13
280.00	280.00	0.00	53.13
290.00	290.00	0.00	53.13
300.00	300.00	0.00	53.13

Figura 7.5: Resultado salvo no disco no formato “.txt” do poço vertical



MD (m)	TVD (m)	INC (°)	AZM (°)
0.00	0.00	0.00	53.13
10.00	10.00	0.00	53.13
20.00	20.00	0.00	53.13
30.00	30.00	0.00	53.13
40.00	40.00	0.00	53.13
50.00	50.00	0.00	53.13
60.00	60.00	0.00	53.13
70.00	70.00	0.00	53.13
80.00	80.00	0.00	53.13
90.00	90.00	0.00	53.13
100.00	100.00	0.00	53.13
110.00	110.00	0.00	53.13
120.00	120.00	0.00	53.13
510.92	510.00	0.00	53.13
540.54	530.00	1.00	53.13
570.15	559.99	2.00	53.13
599.76	589.96	3.00	53.13
629.38	619.90	4.00	53.13
658.99	649.81	5.00	53.13
688.61	679.67	6.00	53.13
718.22	709.48	7.00	53.13
747.84	739.22	8.00	53.13
777.45	768.89	9.00	53.13
807.07	798.48	10.00	53.13
836.68	827.98	11.00	53.13
866.30	857.37	12.00	53.13
895.91	886.66	13.00	53.13
925.53	915.83	14.00	53.13
955.14	944.88	15.00	53.13
984.76	973.79	16.00	53.13
1014.37	1002.55	17.00	53.13

Figura 7.6: Resultado salvo no disco no formato “.txt” do poço direcional

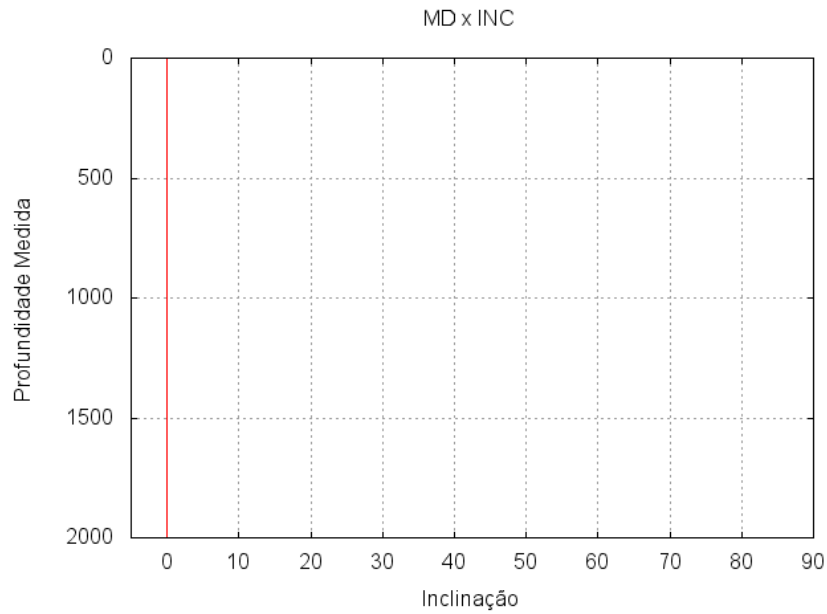


Figura 7.7: Gráfico exibindo a relação entre a profundidade medida e a inclinação do poço vertical.

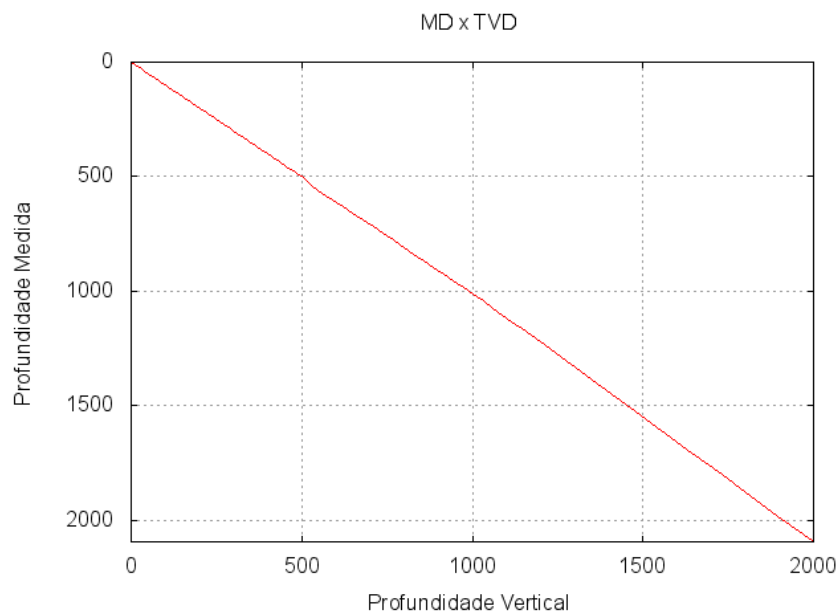


Figura 7.8: Gráfico exibindo a relação entre a profundidade medida e a profundidade vertical do poço direcional.

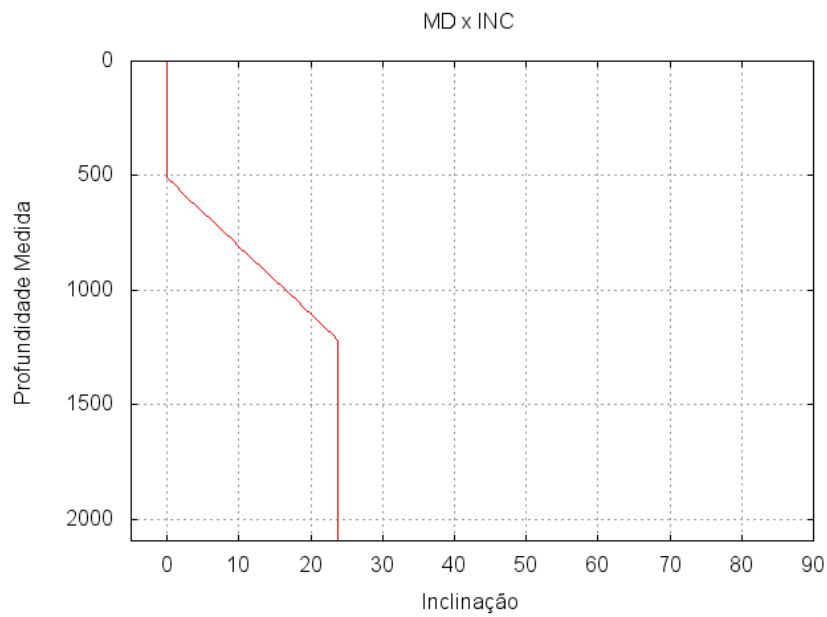


Figura 7.9: Gráfico exibindo a relação entre a profundidade medida e a inclinação do poço direcional.

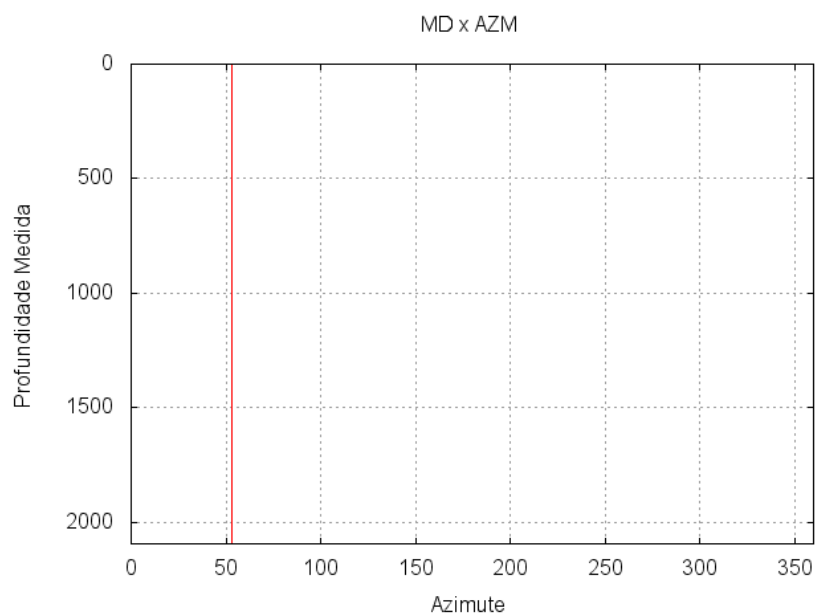


Figura 7.10: Gráfico exibindo a relação entre a profundidade medida e o azimuth do poço direcional.

Capítulo 8

Documentação

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo a documentação de uso do "software Cálculo de Trajetória Direcional para Perfuração de Poços de Petróleo". Esta documentação tem o formato de uma apostila que explica passo a passo como usar o software.

8.1 Documentação do usuário

Descreve-se aqui o manual do usuário, um guia que explica, passo a passo a forma de instalação e uso do software desenvolvido.

8.1.1 Como instalar o software

Abra o terminal, vá para o diretório onde está o projeto, compile o programa e, depois o execute. Logo após, siga os seguintes passos:

1. Entre com “v” para poço vertical e “d” para direcional;
2. Entre com os seguintes dados:
 - (a) Caso for vertical:
 - i. Coordenadas (x,y) da cabeça do poço, em UTM;
 - ii. Coordenadas (x,y) do alvo geológico, em UTM;
 - iii. Profundidade final do poço (TVD), em m.
 - (b) Caso for direcional:
 - i. Coordenadas (x,y) da cabeça do poço, em UTM;
 - ii. Coordenadas (x,y) do alvo geológico, em UTM;
 - iii. Profundidade final do poço (TVD), em m;
 - iv. Profundidade inicial do ganho de ângulo (KOP), em m;
 - v. Taxa de ganho de ângulo (BUR), em °/30m;

3. Após isso serão mostrados no console os resultados, e serão salvos o “.txt” com o resultados e os gráficos em “.png”.

8.2 Documentação para desenvolvedor

Apresenta-se nesta seção a documentação para o desenvolvedor, isto é, informações para usuários que queiram modificar, aperfeiçoar ou ampliar este software.

8.2.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- No sistema operacional GNU/Linux:
 - Instalar o compilador g++ da GNU disponível em <http://gcc.gnu.org>.
 - Para instalar no GNU/Linux use o comando `yum install gcc`.
- O software `gnuplot`, disponível no endereço <http://www.gnuplot.info/>, deve estar instalado.
 - É possível que haja necessidade de setar o caminho para execução do `gnuplot`.
 - Biblioteca `CGnuplot`; os arquivos para acesso a biblioteca `CGnuplot` devem estar no diretório com os códigos do software;
- No sistema operacional Windows:
 - Instalar o compilador apropriado;
 - Recomenda-se o CodeBlocks 16.01, disponível no endereço <http://www.codeblocks.org/dow>

8.2.2 Como gerar a documentação usando doxygen

A documentação do código do software deve ser feita usando o padrão JAVADOC, conforme apresentada no Capítulo - Documentação, do livro texto da disciplina. Depois de documentar o código, use o software `doxygen` para gerar a documentação do desenvolvedor no formato html. O software `doxygen` lê os arquivos com os códigos (*.h e *.cpp) e gera uma documentação muito útil e de fácil navegação no formato html. Segue exemplo da documentação gerada, Figura 8.1 e segue o diagrama gerado pelo software (Figura 8.2).

←

→

↺

file:///G:/01ModeloDocumento-ProjetoEngenharia/listagens/html/annotated.html

☆

≡

🔍

⌵

TrajectoriaDirecional

Main Page

Classes ▾

Files ▾

Q Search

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

C CParametrosDirecional	
C CSimulador	O SIMULADOR REUNIE TODAS AS CLASSES E ACESSA TODAS PARA CHAMAR OS RESULTADOS
C CTrajDirecional	PARA ISSO PRECISAMOS INCLUIR TODAS AS CLASSES
C CTrajDirecionalBuildUp	UTILIZAR OS PARÂMETROS DIRECIONAIS CALCULADOS NA CLASSE "CPARAMETROSDIRECIONAL"
C CTrajDirecionalSiant	
C CTrajVertical	

Generated by

doxygen

1.8.13

Figura 8.1: Documentação do projeto gerada pelo Doxygen

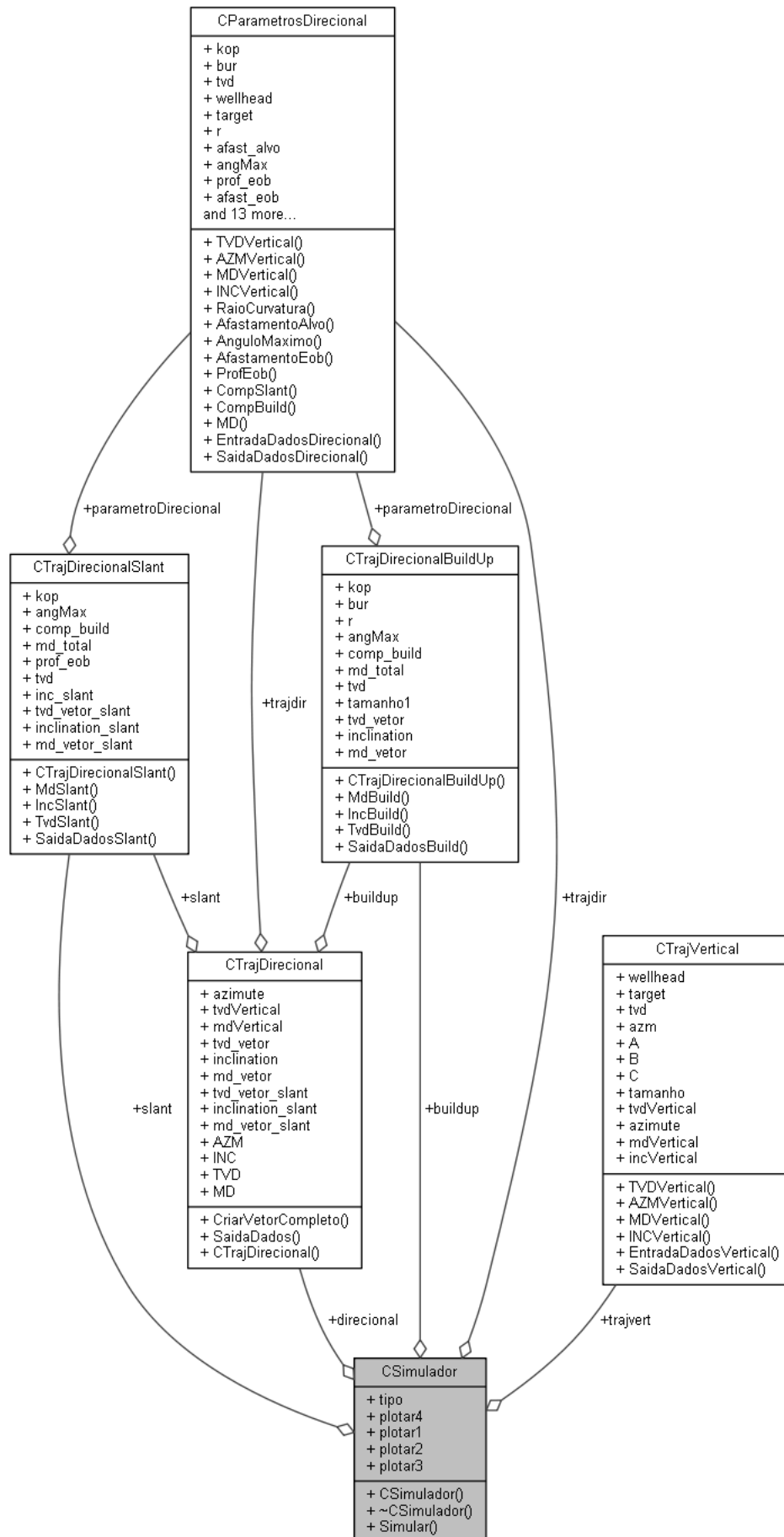


Figura 8.2: Diagrama de classes gerada pelo Doxygen

Referências Bibliográficas

- [1] ROCHA, L. A. S. et al. Perfuracao direcional. *Rio de Janeiro: Editora Interciência*, 2006.
- [2] BOURGOYNE, A. T. et al. Applied drilling engineering. Society of Petroleum Engineers Richardson^ eTX TX, 1986.
- [3] THOMAS, J. E. *Fundamentos de engenharia de petroleo*. [S.l.]: Interciencia, 2001.
- [4] BUENO, A. Programacao orientada a objeto com c++, 2^a ed. *Novatec Editora Ltda*, 2007.
- [5] FARAH, F. O. et al. Directional well design, trajectory and survey calculations, with a case study in fiale, asal rift, djibouti. United Nations University, 2014.

Índice Remissivo

A

Análise orientada a objeto, 17
AOO, 17

C

Casos de uso, 5
Cenário, 4
colaboração, 20
comunicação, 20
Concepção, 3
Controle, 23

D

Diagrama de colaboração, 20
Diagrama de componentes, 24
Diagrama de execução, 25
Diagrama de máquina de estado, 20
Diagrama de sequência, 19

E

Efeitos do projeto nas heranças, 24
Efeitos do projeto nos métodos, 24
Elaboração, 6
especificação, 3
estado, 20
Eventos, 19

H

Heranças, 24
heranças, 24

I

Implementação, 26

M

Mensagens, 19

métodos, 24

modelo, 23, 24

P

Plataformas, 23
POO, 23
Projeto do sistema, 22
Projeto orientado a objeto, 23
Protocolos, 22

R

Recursos, 22