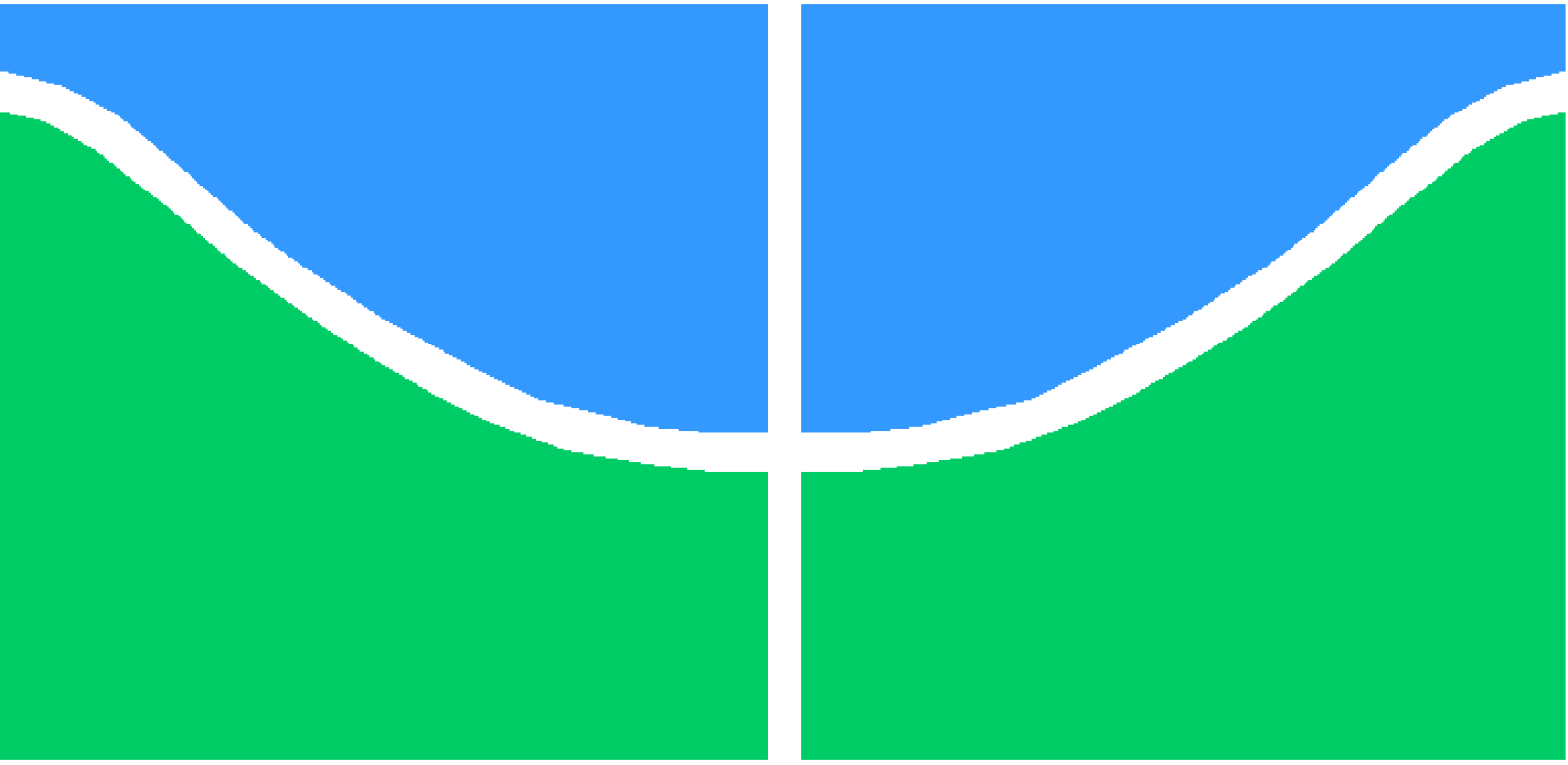


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia Eletrônica

Desenvolvimento de metodologias de quantização espacial para o processamento de sinais de retroespalhamento acústico e batimetria adquiridos por sonares multifeixe

Autores: Haroldo Júnio dos Santos e Bruno Mota de Souza

Orientador: Dr. Luciano Neves Fonseca, PhD

Brasília, DF

2018

Haroldo Júnio dos Santos e Bruno Mota de Souza

Desenvolvimento de metodologias de quantização espacial para o processamento de sinais de retroespalhamento acústico e batimetria adquiridos por sonares multifeixe

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Luciano Neves Fonseca, PhD

Brasília, DF

2018

Resumo

O fundo marinho pode fornecer informações importantes que podem ser úteis para aplicações comerciais, acadêmicas e militares. Ondas acústicas são empregadas para a realização de coleta remota de informações do fundo marinho com a geração de sinais acústicos transdutores eletroacústicos de sonares, de onde se propagam pela coluna d'água em direção ao fundo do mar. Ao alcançarem o fundo, estes sinais interagem com os sedimentos lá encontrados e retornam para aos transdutores por meio de um fenômeno conhecido como retroespalhamento. O retorno é registrado nos sonares, que armazenam todas as informações recebidas na forma de datagramas assíncronos. Para o processamento, primeiramente é feita a leitura e interpretação dos datagramas para que sejam feitas as devidas correções dos feixes que voltam ao sonar. Estas correções resultam em um fluxo complexo de processamento com múltiplas intervenções. No entanto, partindo-se da hipótese de que os dados sonográficos são superamostrados espacialmente, pode-se simplificar o processamento a partir de um mapeamento adaptativo dos feixes acústicos em células de quantização espacial (*bins*), organizadas, por exemplo, em grades regulares. Todos os feixes de um levantamento sonográficos são então mapeados no conjunto espacial de células. Após o mapeamento, os feixes que incidem nos limites espaciais uma célula são guardados em uma lista específica associada a esta célula. Tanto um mosaico batimétrico quanto um mosaico de retroespalhamento podem ser obtidos diretamente das listas associadas às células de quantização espacial, a partir de uma simples filtragem de mediana. Os mosaicos finais resultantes deste processamento mais simples apresentam baixo nível de ruído e são comparáveis aos obtidos a partir de um processamento tradicional feito em um software comercial dedicado.

Palavras-chaves: Quantização espacial. Sonar Multifeixe. Batimetria. Retroespalhamento.

Abstract

The seabed has been thoroughly investigated since the beginning of navigation, mainly due to the need for a precise knowledge of the seabed bathymetric surface. However, the seabed can provide other information, which can be important for applications such as mineral resource exploration, underwater habitat mapping, terrain stability for offshore installations, among others. Due to mechanical characteristics of the aqueous medium, which make the propagation of electromagnetic waves extremely difficult, acoustic waves are used for the remote acquisition of seabed information. For that, acoustic signals are generated by electroacoustic transducers assembled in sonars, from where they propagate by the water column towards the seabottom. Then, these signals interact with bottom sediments and return to the transducers through a phenomenon known as backscatter. The return signal is recorded by the sonars, which store all received information in the form of asynchronous datagrams, which are specific to each sonar manufacturer. These records are extremely useful both for obtaining bathymetric models, which depict the relief of the bottom, and for the remote survey of geoacoustic features of the seabed. For that, first we need to read and interpreted the datagrams, in order to apply the necessary geometric and radiometric corrections for each beam returning to the sonar. These corrections result in a complex processing flowwith multiple interventions. However, based on the hypothesis that multibeam data are spatially super-sampled, we can simplify the processing by applying an adaptive mapping of the acoustic beams into spatial data bins, which can be organized, for instance, into a regular grid. All beams from the survey are then mapped into the spatial bins. After the mapping, all beams that fall into the boundaries of a bin are stored in a specific list associated with each that bin. A bathymetric mosaic and a backscatter mosaic are obtained directly from the lists associated with the bins, by the use of a simple median filtering. The final mosaics obtained from this simple processing show a low noise level and are comparable to those obtained from traditional processing pipeline, implemented in a dedicated commercial software.

Key-words: Data binning. Multibeam echosounder. Bathymetry. Backscatter.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Sonar	*Sound Navigation and Ranging*
Hz	Hertz
UTM	Universal Transverse Mercator
IQR	Interquartile range

Sumário

1 Introdução

O solo marinho tem sido alvo de investigações detalhadas desde o início das navegações. Seus sedimentos podem abrigar informações especiais em suas composições e características, o que é essencial para a indústria petrolífera dentre outras aplicações que incluem exploração mineral, caracterização de habitats marinhos com estudos biológicos, além de dados geológicos que podem fornecer, por exemplo, um histórico de formação geológica e uma avaliação das mudanças climáticas e físicas do nosso planeta.

No entanto, o fundo do mar pode se tornar bastante inacessível ao considerar sua profundidade, que pode passar dos 4 mil metros. Assim, o estudo desses sedimentos presentes no solo marinho necessita da utilização de técnicas de coleta remota de dados.

Para o mapeamento remoto, comumente são aplicadas estratégias de obtenção de imagens contendo informações nos mais variados espectros eletromagnéticos, como o visível, micro-ondas, infravermelho e ultravioleta. Ondas eletromagnéticas são extremamente úteis e fáceis de serem produzidas e recebidas, provendo facilidade. Porém, ao adentrar no contexto de mapeamento e caracterização de solo marinho, esse tipo de radiação é pouco eficaz, visto que é altamente atenuada em ambiente aquoso, particularmente na água salgada. Isso se deve por conta de propriedades específicas da água, como sua densidade, viscosidade e elasticidade.

Para meio aquoso, ondas acústicas apresentam melhor penetração, propagando-se facilmente e alcançando o solo do oceano. Para o emprego desse tipo de onda mecânica, faz-se o uso de transdutores eletroacústicos presentes em sonares, equipamentos que revolucionaram a estimação da profundidade dos oceanos.

O sonar, junto com seu transdutor eletroacústico acoplado, permite a emissão de ondas sonoras em forma de pulsos direcionados ao solo marinho. Esses pulsos, ao atingirem o piso, retornam retroespalhados ao navio. O tempo de ida e volta é calculado com precisão e permite, juntamente com o conhecimento do perfil de velocidade do som no meio de propagação, o cálculo da profundidade, ou batimetria (MOUSTIER; MATSUMOTO, 1993). Além disso, o processamento do sinal retornado ainda permite a caracterização da composição do sedimento do piso marinho (FONSECA; MAYER, 2007).

O sonar provê duas características importantes para o entendimento de seu funcionamento:

- *Batimetria:* É a informação relacionada à profundidade em cada ponto do solo marinho. A partir da batimetria são ajustados modelos estatísticos para a geração de mapas 3-d de profundidade. A batimetria está relacionada ao tempo de retorno do

sinal emitido.

- *Retroespalhamento:* É a informação relacionada à estrutura física em cada ponto do solo marinho. A partir do retroespalhamento é ajustado um modelo referente à composição do solo marinho. O retroespalhamento está relacionado à intensidade do sinal que retorna ao sonar.

Existem vários tipos de sonares que são utilizados para essa tarefa. Os mais comuns são o sonar de feixe único, o sonar de imageamento lateral e o sonar multifeixe (PENROSE et al., 2005), presentes na Figura 1. O sonar de feixe único é caracterizado pela emissão de feixes verticais, possuindo dois ecos, o principal onde se encontra a informação de tempo de retorno, que se utiliza para a geração da batimetria, e informações de retroespalhamento, e o secundário onde se encontra mais informações de retroespalhamento. O sonar de imageamento lateral é geralmente um veículo submerso rebocado por um navio e é caracterizado por emitir dois feixes em leque com amplo alcance de cobertura, com a desvantagem de, geralmente, só se adquirir a informação de retroespalhamento. Já o sonar multifeixe é caracterizado pela emissão de vários feixes individuais em formato de leque, podendo ser adquiridas informações de batimetria e retroespalhamento simultaneamente (LURTON et al., 2015).

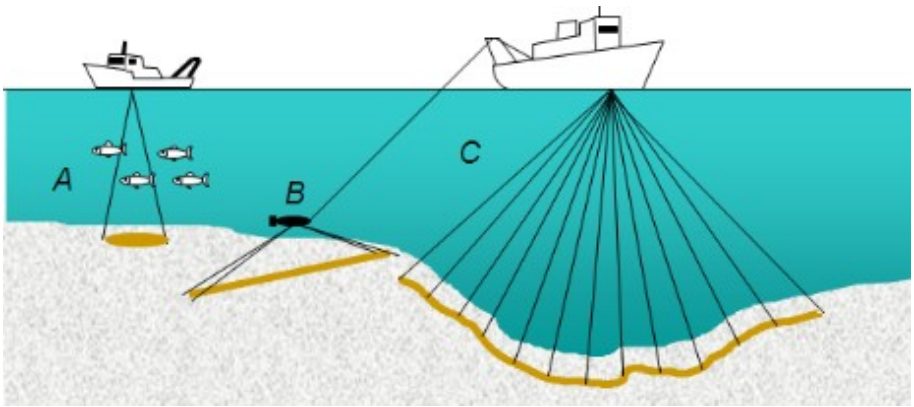


Figura 1 – Representação dos três principais tipos de sonares utilizados na caracterização do solo marinho. (A: sonar de feixe único; B: sonar de imageamento lateral; C: sonar multifeixe). Fonte: LURTON et al. (2015)

O sonar multifeixe possibilitou um avanço tecnológico importante na década de 1960, quando a Marinha Norte Estadunidense começou a desenvolver métodos de batimetria com vários feixes de som em cooperação com a General Instrument, um sistema com transdutores organizados em linha que envia ao mesmo tempo múltiplos pulsos, que experimentam retroespalhamentos individuais e interferências múltiplas entre si e o ambiente, todos eles retornando em tempos diferentes. Isso permite uma área muito maior de varredura e uma alta resolução de mapeamento. Contudo, os dados possuem maior complexidade e um tempo maior de processamento é necessário.

Com o desenvolvimento de plataformas de processamento mais produtivas e avançadas, pôde-se colocar um início na aplicação de dados obtidos por sonares com feixes múltiplos. Ao mensurar a energia retornante dos múltiplos feixes refletidos e combinar esses dados com parâmetros físicos coletados no ambiente de estudo, abriu-se a possibilidade de realização de mapeamento que apresente características básicas do solo marinho.

Tendo em vista a importância da possibilidade de caracterização do fundo do mar, essa técnica começou a ser amplamente desenvolvida e penetrou cada vez mais esferas comerciais, sendo uma importante ferramenta para o mercado naval em geral. No entanto, mesmo sendo algo resultante de um processamento complexo, sua batimetria e retroespalhamento resultantes podem apresentar perda de detalhes devido à própria quantização espacial e processos gerais. Para a resolução dessa perda em detalhamento, o presente estudo apresenta uma contribuição à evolução da caracterização do solo marinho com uma rotina em linguagem de programação C para a leitura e pré-processamento de dados de sonar com abordagem de quantização espacial e ainda apresenta a melhoria de processamento de detalhes com o desenvolvimento de script em linguagem de programação Python para aplicação como complemento ao programa QGIS, uma completa aplicação de sistema de informações geográficas com capacidade para processamento de projeções e manipulações de mapas em várias camadas.

Por meio do processamento de dados de sonar Simrad EM1002 da empresa Kongsberg, chegou-se a um mapeamento de retroespalhamento acústico e batimetria quantizados espacialmente e com a nova técnica desenvolvida adquiriram-se os mapas semelhantes porém em detalhamento maior. Todos os mapas demonstram de maneira fiel as características do solo estudado, contudo, para mais detalhamento, foi necessária a perda da coerência espacial entre linhas quando formado um mosaico batimétrico, o que torna os mapas mais detalhados úteis não mais para navegação e sim para aplicações militares, de prospecção e arqueológicas.

2 Aspectos Gerais

2.1 Batimetria e restroespalhamento

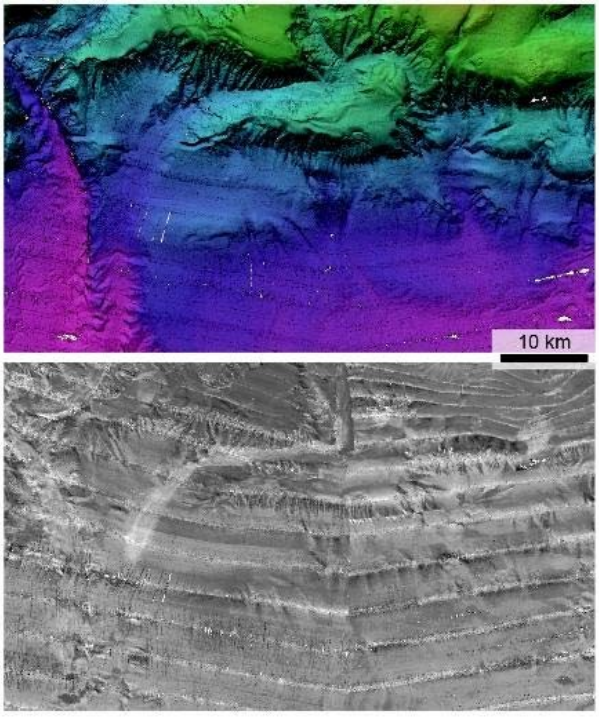


Figura 2 – Batimetria (acima) e retroespalhamento (abaixo) de uma mesma área (Estreito de Cook, Nova Zelândia). Fonte: LURTON et al. (2015)

2.1.1 Batimetria

É a medição de profundidade de um meio aquático. A geração de mapas batimétricos tem um papel muito importante na segurança das navegações, principalmente em localidades de águas rasas. Em localidades de águas profundas, entretanto, possui importante caráter exploratório. (CHAKRABORTY; FERNANDES, 2012) Um desses mapas batimétricos pode ser visto na parte superior da Figura 2.

Até o século XIX os instrumentos batimétricos eram baseados em métodos de *linha de prumo*, nos quais era realizada a batimetria a partir de um prumo, que era descido até tocar no fundo e então medida a linha que o sustenta. Só a partir da década de 1920 que este método foi substituído pela técnica atual de uso de ecobatímetros, primeiro com o sonar de feixe único, que só se popularizou na década de 1950 com o avanço da tecnologia de transdutores. Na década de 1970 surgiu o sonar multifeixe, capaz de mapear o solo aquático com alta resolução e alta densidade batimétrica de uma extensa faixa com alta acurácia, que o torna economicamente viável. (CHAKRABORTY; FERNANDES, 2012)

Em sistemas baseados em sonares a batimetria é obtida de uma forma bastante direta a partir da medição de tempo do eco de um sinal emitido. Entretanto outros fatores devem ser levados em consideração para uma mensuração acurada. (LURTON et al., 2015)

2.1.2 Retroespalhamento

É o retorno de ondas dispersadas ao serem refletidas por um objeto. A intensidade desse retorno depende das características do objeto, assim como a distância, o ângulo e as propriedades físicas do meio (como temperatura e densidade). (LURTON et al., 2015) O espalhamento pode ser visto na Figura 3, o retroespalhamento são as ondas espalhadas que voltam para a fonte.

Figura 3 – Reflexão e espalhamento de uma onda. Fonte: Adaptado de LURTON et al. (2015)

Para a caracterização do solo marinho é necessária a aquisição de dados de propriedades físicas, como tamanho de grão, rugosidade, dureza e composição sedimentar. Porém estas informações normalmente não podem ser medidas remotamente de forma direta, é necessário então se medir de forma indireta. (FONSECA; MAYER, 2007)

A partir do uso de sonares é possível se fazer essa caracterização com a utilização de técnicas relacionadas ao retroespalhamento acústico. Dados de retroespalhamento são adquiridos e as propriedades físicas do solo marinho a partir de modelos empíricos e teóricos. (FONSECA; MAYER, 2007)

Um mapa com as intensidades de retroespalhamento pode ser visto na parte inferior da Figura 2.

2.2 Sonares

Por haver maior atenuação de ondas eletromagnéticas em meio aquático que ondas sonoras, o som se torna a melhor maneira de se realizar sensoreamento remoto deste meio. O instrumento que utiliza ondas sonoras para detecção, localização e classificação de estruturas submersas é denominado sonar (STERGIOPOULOS, 2000).

Sonares são frequentemente tratados de forma similar a radares (*radio detection and ranging*). O que os diferencia, além do tipo de onda utilizada, são as frequências dessas

ondas. Radares utilizam ondas eletromagnéticas com frequências de operações na ordem de GHz e larguras de banda da ordem de MHz. Já sonares utilizam ondas sonoras (mecânicas) com frequências de operações e larguras de banda na ordem de kHz (CREASEY, 1990).

Sonares são divididos entre sistemas passivos e ativos:

- *Passivos*: são sonares que recebem as ondas sonoras produzidas pelo objeto a ser detectado. São utilizados para localização de embarcações, seres marinhos.
- *Ativos*: são sonares que recebem ondas sonoras produzidas por eles mesmos e refletidas pelo objeto a ser detectado. São os mais utilizados, podendo ser utilizados na detecção de qualquer objeto.

Para a caracterização do solo marinho sonares ativos são empregados.

2.2.1 Classificação de sonares

Os sonares mais comumente utilizados para a caracterização do solo marinho são três: o sonar de feixe único, o sonar de imageamento lateral e o sonar multifeixe. O funcionamento desses três sonares é apresentado na Figura 1.

2.2.1.1 Sonar de feixe único

É o sonar mais simples existente, constituído por um transdutor que emite um único feixe acoplado no casco da embarcação. O retorno adquirido por este tipo de sonar é tipicamente no formato visto na Figura 4. Primeiro há uma reverberação devido ao uso do transdutor para se emitir o sinal, após isso há o retorno do primeiro eco e do segundo eco.

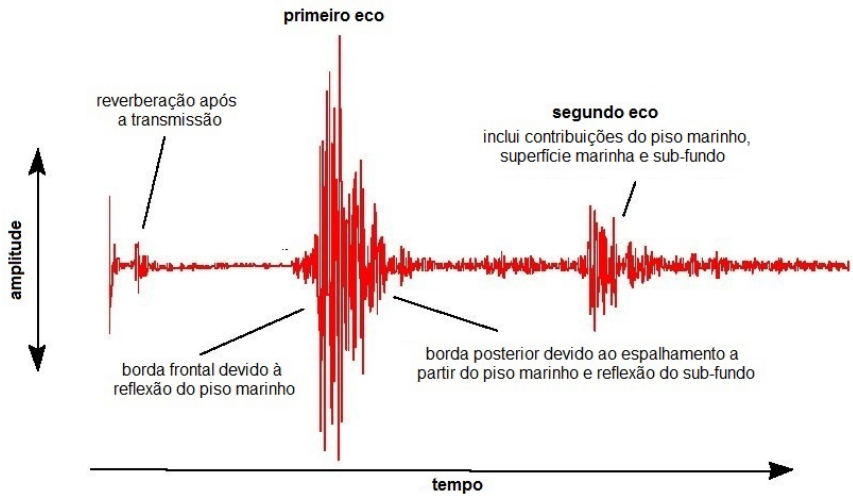


Figura 4 – Retorno típico de um sonar de feixe único. Fonte: Adaptado de PENROSE et al. (2005)

A varredura é feita com uma visão vertical a partir da embarcação. O retorno adquirido pelo transdutor possui dois ecos principais, o primeiro (chamado E1) referente à reflexão do feixe emitido e ao seu retroespalhamento e o segundo (chamado E2) referente a reflexão do primeiro eco na superfície marítima - E2 é a reflexão do feixe, duas vezes pelo piso marinho e uma vez pela superfície marinha. De maneira geral E2 tem aproximadamente o dobro de atraso em relação a E1 (PENROSE et al., 2005).

O sistema que utiliza os dados de E1 e E2 para classificação sedimentar do piso marinho é o RoxAnn, desenvolvido pela Marine Micro Systems of Aberdeen, Escócia, Reino Unido. E1 é considerado com informações predominantes de rugosidade acústica e E2 de dureza acústica. Para cada tipo de sedimento há uma relação entre essas duas informações (PENROSE et al., 2005).

2.2.1.2 Sonar de imageamento lateral

É um sonar no qual existem dois transdutores, um de cada lado da embarcação. Tipicamente é utilizado um corpo rebocado, chamado de *peixe*, como visto na Figura 5. Cada transdutor emite um fino feixe em forma de leque na direção perpendicular à navegação. Como a maior parte da energia é refletida pelo solo marinho somente o retroespalhamento é adquirido (PENROSE et al., 2005).

Figura 5 – Corpo rebocado do sonar PulSAR (ao centro). Fonte: KONGSBERG (2015a)

A maior vantagem deste tipo de sonar é a capacidade de formar imagens bem realísticas do piso marinho, um exemplo pode ser visto na Figura 6. Após a formação de mosaicos as estruturas físicas são facilmente reconhecidas. Porém a amplitude do sinal adquirido é muito variável, fazendo com que uma área com o mesmo sedimento possa aparentar possuir diferentes tipos de sedimento (KENNY et al., 2003).

Para se adquirir batimetria a partir de um sonar de imageamento lateral são necessárias adaptações para se fazer uso da interferometria. Adicionando receptores paralelos ao transmissor e comparando as séries temporais dos sinais recebidos pode-se calcular a posição no piso marinho a partir dos ângulos de chegada desses sinais. Esses sonares também são chamados de sonares batimétricos (BLONDEL, 2010).

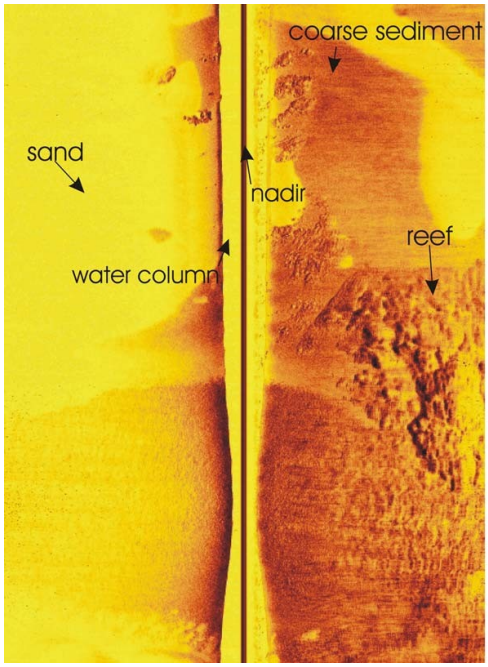


Figura 6 – Exemplo de uma imagem bruta adquirida por um sonar de imageamento lateral. Fonte: PENROSE et al. (2005)

2.2.1.3 Sonar multifeixe

É a tecnologia de sonar mais recente e mais avançada para sensoriamento e caracterização remota do solo marinho. Dados de batimetria e de retroespalhamento podem ser adquiridos simultaneamente e com alta qualidade com essa tecnologia (PENROSE et al., 2005; KENNY et al., 2003) .

Figura 7 – Funcionamento típico de um sonar multifeixe. Fonte: MAYER; LI; MELVIN (2002)

É constituído de dois arranjos de transdutores (Figura 8), geralmente retângulos, um para emissão, montado no sentido transversal, e outro para recepção, montado no sentido longitudinal. Seu funcionamento (Figura 7) é a emissão de um arranjo de feixes em leque largo na direção transversal e estreito na direção longitudinal. O sinal recebido é a formação de vários feixes direcionados perpendicularmente a partir do arranjo emitido que é largo na direção longitudinal e estreito na direção transversal (MAYER; LI; MELVIN, 2002; PENROSE et al., 2005).

Figura 8 – Sonar multifeixe EM 2040 da Kongsberg. Fonte: KONGSBERG (2013)

2.2.2 Equações do sonar

- *Nível de eco:* O nível de eco (EL) é derivado da *equação do sonar* (LURTON et al., 2015), e é descrito, em dB, como:

$$EL = SL - 2TL + TS \tag{2.1}$$

onde, SL é a intensidade da fonte, TL a perda de transmissão e TS a força do alvo.

- *Perda de transmissão:* A perda de transmissão (TL) é dada por uma componente de espalhamento esférico e por uma componente de absorção pela água (HAMMERSTAD, 2000). É descrita em dB por:

$$2TL = 2\alpha R + 40log(R) \tag{2.2}$$

onde, R é a distância e α o coeficiente de absorção.

- *Força do alvo:* A força do alvo (TS) descreve o quanto da energia é redirecionada em relação com o quanto de energia atingiu o alvo (LURTON et al., 2015). É dada em dB por:

$$TS = 10log\left(\frac{I_s}{I_i}\right) \tag{2.3}$$

onde, I_i é a intensidade incidente e I_s a intensidade espalhada.

Essas equações são importantes para se realizar as correções angulares de dependência do retroespalhamento.

2.3 Datagramas

Os dados lidos pelos sonares da série EM da Kongsberg, sendo o *EM1002* utilizado neste trabalho, são organizados e armazenados em datagramas assíncronos (KONGSBERG, 2015b). Todos os datagramas possuem 3 partes principais: um cabeçalho (*header*), um corpo e um rodapé (*footer*). Cada campo de dado possui 1, 2 ou 4 bytes nos formatos *unsigned*, *signed* ou *float*.

No cabeçalho, Figura 9, há um identificador de início (*STX*, sempre 0x02) e é onde estão as informações de tipo e tamanho do datagrama, modelo e número de série do sonar, a data e o tempo após a meia noite. No corpo estão todas as informações únicas de cada tipo de datagrama, como por exemplo latitude e longitude em um datagrama de posição, ou retroespalhamento e batimetria em um datagrama *XYZ*. No rodapé, Figura 10, há um identificador de fim (*ETX*, sempre 0x03) e o *check sum* do datagrama.

Núm. de bytes	STX	Tipo	Modelo	Data	Tempo

Figura 9 – Cabeçalho dos datagramas. Fonte: Autoria própria

ETX	Check sum

Figura 10 – Rodapé dos datagramas

Os datagramas utilizados neste trabalho são o datagrama de posição, o datagrama de profundidade e o datagrama de retroespalhamento.

Toda a navegação é divida em várias *linhas de navegação*. Os datagramas referentes a cada uma dessas *linhas* são concatenados e armazenados em um arquivo com extensão *.all*. Como a posição está armazenada em um tipo de datagrama, a profundidade em outro e o retroespalhamento em outro, para se adquirir essas últimas informações em função da posição é então necessário se fazer uma interpolação entre estes datagramas utilizando o tempo como referência.

2.3.1 Datagrama de posição

É um datagrama que possui como identificador o valor 0x58. É nele que estão contidas as informações de latitude e longitude adquiridas ao decorrer da navegação por um GPS presente na embarcação, Figura 11.

Neste datagrama também estão contidos os dados de velocidade, curso e *heading* da embarcação.

Latitude	Longitude	Velocidade	Curso	Heading

Figura 11 – Datagrama de posição

2.3.2 Datagrama de profundidade

É um datagrama que possui como identificador o valor 0x44. Este datagrama possui os dados aferidos por cada feixe do sonar, então há uma repetição de um ciclo de dados para cada batimento (*ping*), Figura 12.

Neste ciclo há, para cada feixe (*beam*), as informações de profundidade (eixo Z), distância transversal (eixo X) e distância longitudinal (eixo Y), todas as três em relação ao transdutor e retroespalhamento (amostra central).

Neste datagrama também estão contidas as informações de velocidade do som no transdutor, número de detecções válidas, profundidade do transdutor, *heading* da embarcação e frequência de amostragem.

Heading	Velocidade do som	Profundidade do transdutor
Detecções válidas = N	Resolução Z	Resolução X e Y
Taxa de amostragem	**Repetição de N ciclos:**	Profundidade
Distância Y	Distância X	Restroespalhamento

Figura 12 – Datagrama de profundidade

2.3.3 Datagrama de retroespalhamento

É um datagrama que possui como identificador o valor 0x53. É onde estão armazenados todos os valores de amostras de retroespalhamento (série temporal), Figura 13.

Este datagrama também possui as informações de coeficiente de absorção, largura do pulso, distância para a incidência normal, incidência normal de retroespalhamento (*BSN*), incidência obíqua de retroespalhamento (*BSO*) e ângulo de cruzamento da lei de ganho variável no tempo (*TVG law*), que são utilizadas na inversão do modelo.

Coeficiente de absorção	Largura do pulso	Distância à normal
BSN	BSO	Ângulo de cruzamento TVG
Detecções válidas = N	**Repetição de N ciclos:**	Número de amostras = Ns
Núm. amostra central	**Repetição de $\sum$Ns ciclos:**	Restroespalhamento

Figura 13 – Datagrama de retroespalhamento

2.4 Pré-processamento

2.4.1 Ajuste de ângulo de retroespalhamento

O retroespalhamento possui uma dependência com o ângulo de incidência dos feixes emitidos pelo sonar (LURTON et al., 2015). Para se tratar esse tipo de dado é necessário uma compensação angular, que remova essa dependência, vista na Figura 14.

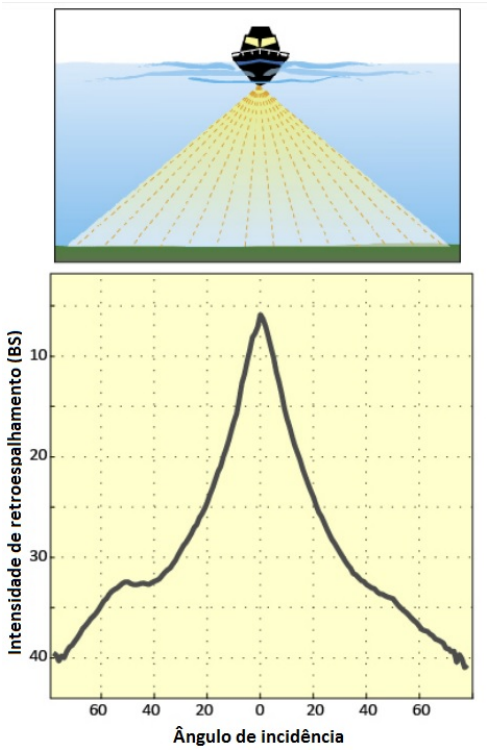


Figura 14 – Dependência angular da intensidade de retroespalhamento, com ângulos em módulo e piso marinho não simétrico. Fonte: Adaptado de LURTON et al. (2015)

Uma primeira tentativa é a de utilizar a dependência angular com um $cos(\theta)$. Juntamente com um comportamento de espalhamento local também com um $cos(\theta)$. As contribuições resultam em uma dependência angular de um $cos^2(\theta)$ (LURTON et al., 2015). Essa dependência é conhecida como *Lei de Lambert* e é expressa em dB por:

$$BS(\theta) = BS_0 + 20log(cos(\theta)) \tag{2.4}$$

A dependência angular possui boa aproximação pela *Lei de Lambert* em ângulos acima de 25° (em módulo), onde se pode assumir um piso marinho plano (HAMMERSTAD, 2000).

Para inclusão de ângulos menores que 25° (em módulo), assumindo que o coeficiente de retroespalhamento varia linearmente com o ângulo de incidência, e considerando que ângulo de incidência de 25° é aproximadamente $R = 1.1R_N$, sendo R_N o *range* para

a incidência normal, o modelo completo se encontra na Equação 2.5.

$$\begin{aligned} BS &= BS_N + 10log(\theta_x \theta_y R^2), R \leq R_N \\ BS &= BS_O - 5log\left(\frac{R}{R_N}\right)^2 \left[\left(\frac{R}{R_N}\right)^2 - 1\right] + 10log\left(\frac{c\tau}{2}\theta_x R\right), R \geq R_N \\ BS &= BS_N + 3.162\sqrt{\frac{R}{R_N} - 1}(BS_O - BS_N) - 5log\left(\frac{R}{R_N}\right)^2 \left[\left(\frac{R}{R_N}\right)^2 - 1\right] \\ &\quad + 10log\left(\frac{c\tau}{2}\theta_x R\right), R_N < R < 1.1R_N \end{aligned} \tag{2.5}$$

sendo BS_N o retroespalhamento normal, BS_O o retroespalhamento oblíquo, c a velocidade do som no meio, τ a largura do pulso transmitido e θ_x e θ_y os ângulos de abertura do pulso nas direções longitudinal e transversal, respectivamente.

Alguns sonares utilizam lei de ganho variável no tempo (*TVG law*), em que o ângulo de 25° é substituído pelo ângulo de cruzamento .

O sonar utilizado já possui todo o pré-processamento de ajuste de correção angular implementado, já armazenando os valores de retroespalhamento corrigidos, sendo essas equações necessárias caso se desejasse a inversão do modelo.

2.4.2 Quantização espacial de dados (*Data Binning*)

O processo de quantização ou agrupamento espacial (*binning*) nada mais é que uma discretização por categorias, onde o dado a ser analisado é processado e adicionado a uma lista de ocorrências. No caso de imagens, o *binning* em duas dimensões permite a criação de uma matriz com essas ocorrências e cada um de seus elementos de configuram como uma lista (KRZYWINSKI, 2016).

No processo de *binning*, os dados analisados passam por uma grade e são interpretados. Cada célula resultará em um *bin* dos dados ali anteriormente presentes. Assim, tem-se em cada elemento dessa matriz de duas dimensões uma análise estatística dos valores (WICKLIN, 2013).

Como uma tentativa de simplificação do processamento dos dados adquiridos, o uso do *binning* faz-se justificado visto que o mesmo permite a redução significativa da quantidade de processos e seu resultado apresenta uma filtragem adequada de *outliers* (SAYAD, 2011). Quando processados seus *bins* em análises de quartis, o valor a ser substituído em uma imagem resultante será embasado em decisões de característica mediana, muitas vezes não sendo necessário aplicações de filtros posteriores.

Por conta de sua característica natural de demonstrar ocorrências de valores, o *binning* permite a construção de imagens que demonstram a fidelidade e acurácia do

resultado com os dados processados por reduzir o ruído por conta da não linearidade atenuada. Essas imagens são construídas com análises interquartis dos histogramas gerados anteriormente.

3 Processamento de dados de batimetria e retroespalhamento

3.1 Leitura dos datagramas

Foi proposta a criação de um algoritmo responsável pela leitura e pré-processamento dos datagramas. Os dados são lidos, sendo que os que são relevantes são armazenados em um banco de dados. Após isso é necessária a realização de interpolação para se adquirir os dados de profundidade e retroespalhamento em função da posição. Por último é realizado um processamento estatístico (*binning*) para se preencher uma grade removendo *outliers*.

3.1.1 Rotina em C

Para a leitura e processamento dos arquivos contendo os datagramas primeiramente foi criada uma rotina em C divida em 9 arquivos: estruturas (*structs.h*), *main* (*main.c*), entradas/saidas (*io.c*), interpretação (*interpretation.c*), leitura dos datagramas (*read_datagrams.c*), interpolação (*interpolation.c*), grade (*grid.c*), *byteswap* (*byteswap.c*) e ângulo (*angle.c*).

A rotina pode ser dividida em 3 partes: *leitura dos datagramas*, *interpolação* e *criação das grades*. Com seu fluxograma presente na Figura 15.

A rotina constitui de uma estrutura de dados principal onde são armazenadas todas as informações relevantes de batimetria e retroespalhamento (amostras centrais).

A grade é salva como um arquivo ASCII, sendo um conjunto de 4 dados, as coordenadas X e Y, a profundidade Z e a amostra central de retroespalhamento BS. Os dados são então importados no software *QGIS*.

3.1.1.1 Leitura dos datagramas

Primeiramente são lidos os arquivos .all presentes numa pasta definida na rotina, chamada *inputs*, e seus caminhos completos salvos na estutura de dados.

É chamada então a função de leitura dos dados dos datagramas de posição e profundidade. Todos os arquivos da lista são lidos, e os dados dos datagramas são salvos na estrutura de dados.

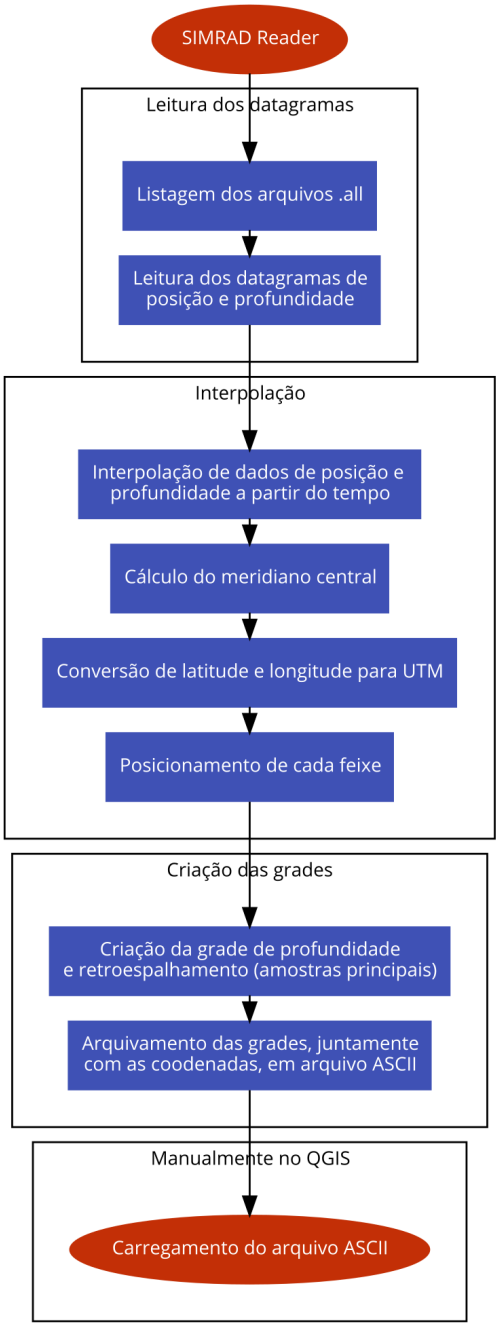


Figura 15 – Fluxograma do código em C

3.1.1.2 Interpolação

Os dados de profundidade são posicionados a partir da interpolação, pelo tempo, entre os dados dos datagramas de profundidade e posição. Para isso os dados são ordenados pelo tempo e os dados repetidos descartados.

Cada feixe é então posicionado a partir das ditâncias X (longitudinal) e Y (transversal), juntamente com a posição do navio. As posições são convertidas de latitude e longitude para UTM, sendo que a zona UTM é armazenada.

3.1.1.3 Criação das grades

Com todos os dados posicionados a grid é criada. Para tal se adquire as extremidades da navegação e se calcula o tamanho da grid a partir das dimensões da navegação e do tamanho do *bin*. Cada valor de profundidade, e retroespalhamento, é inserido no *bin* correspondente à sua localização.

Ao serem inseridos todos os valores é realizada a quantização espacial, onde os *bins* que possuem mais de 5 valores tem a mediana tomada, atribuindo este valor ao *bin*. Nos que não possuem valores suficientes é atribuído *valor inválido*.

Para se carregar os valores no software QGIS, os dados válidos são salvos em um arquivo ASCII, com as coordenadas dos dados e os dados de batimetria e retroespalhamento (amostra central). Na primeira linha do arquivo está a zona UTM e na segunda a informação de qual dado está presente em cada coluna (X, Y, Z e BS). O arquivo de saída é chamado *stats.bs* e se encontra na pasta *Outputs*.

Para se carregar o arquivo ASCII no QGIS se deve abrir um projeto, clicar em *Camada* > *Adicionar camada* > *A partir de um texto delimitado*. Seleciona-se o arquivo *stats.bs*, escolhe-se *Delimitadores personalizados* em *Formato do arquivo*, marcando a opção *Tabulação*. Em *Definição da geometria* selecionar o *Geometry CRS* correspondente à zona UTM presente na primeira linha do arquivo ASCII. Por fim, em *Record and field options* deve-se atribuir valor 1 no campo *Número de linhas de cabeçalho a descartar*. Adiciona-se a camada clicando no botão *Adicionar*.

Vários pontos serão adicionados. Para se vizualizar os dados deve-se clicar com o botão direito sobre o nome da camada (*stats*) no painel *Camadas* e sem seguida em *Properties...*.

Na janela aberta deve-se clicar na aba *Simbologia*, selecionando *Graduated* no menu suspenso na parte mais superior da janela. No menu suspenso *Coluna* escolhe-se o dado a ser visualizado, *Z* ou *BS*. No menu suspenso *Gradiente de cores* escolhe-se ou cria-se este. Em *Classes* atribui-se o número de classes. O tamanho do simbolo que representa cada valor pode ser alterado no botão *Símbolo*, aparecendo uma janela *Symbol selector* com o campo *Tamanho*.

Para melhor visualização o marcador deve ter sua cor de borda alterada, para isso, na janela *Symbol selector* deve-se clicar em *Simple marker* abaixo de *Marcador* e a *Cor do traço* alterada para *Transparent stroke*. Clica-se em *Ok* na duas janelas abertas e os dados poderão ser visualizados.

Para-se visualizar o tipo de dado restante basta ir novamente em *Properties...*, clicando com o botão direito sobre a camada, e seleciona-lo no menu suspenso *Coluna* na aba *Simbologia*.

3.1.2 Script *QGIS* em Python

Para realização do referente trabalho optou-se por se criar um script para o software QGIS na linguagem Python, versão 3.6. A linguagem já vem inclusa na instalação padrão do QGIS e sua forma de programação integrada ao QGIS é conhecida como PyQGIS. Toda a documentação está disponível no site oficial (PYQGIS. . . , 2018).

O script tem como entrada a lista de arquivos .all, selecionados pelo usuário a partir de uma janela de abertura de arquivos, e como saída imagens georreferenciadas de batimetria e retroespalhamento. Sua estrutura básica possui o mesmo princípio da rotina em C.

Todos os dados de entrada são lidos e processados. São geradas grades preenchidas com os dados de batimetria e retroespalhamento salvas como arquivos TIFF georreferenciados e esses arquivos são carregados no QGIS de forma automatizada.

O script é um único arquivo com extensão *py* (*plugin_SIMRAD_QGIS_3_2.py*), que pode ser dividido em 3 partes básicas: *leitura dos dados de batimetria*, *leitura dos dados de retroespalhamento* e *carregar no qgis*. O fluxograma do plugin está na Figura 16.

Para a criação da GUI (interface gráfica do usuário) foi utilizada a bliblioteca *PyQt5*.

O usuário tem a possibilidade de inserir os tamanhos de pixel, em metros, para as grades de batimetria e retroespalhamento, assim como se deseja ou não interpolar os dados para se preencher lacunas. Ao se abrir os arquivos .all que se deseja processar é gerada uma lista com os caminhos completos dos arquivos, sendo que cada arquivo corresponde a uma *linha de navegação*.

Para se executar a interpolação se utilizou da técnica de fechamento de morfologia matemática. Nesta técnica há o uso da técnica de dilatação, seguida do uso da técnica de erosão. Na dilatação lacunas menores que o elemento estruturante são preenchidas e a borda é alargada. Na erosão a borda retoma seu formato original com poucas alterações. O fechamento serve para se suavizar o contorno, preechendo as lacunas (GONZALEZ; WOODS, 2009). Após isso é realizada a interpolação por *spline* cúbica nos pixels presentes na resultante do fechamento.

3.1.2.1 Leitura dos dados de batimetria

Em um *loop* todos os datagramas de posição e profundidade são lidos e armazenados. Os datagramas de profundidade tem suas posições estimadas a partir da interpolação com o tempo com os datagramas de posição. São adquiridos o meridiano central UTM e o código EPSG da navegação e as posições são então convertidas de coordenadas geográficas para UTM e todos os feixes são posicionados a partir das distâncias X e Y.

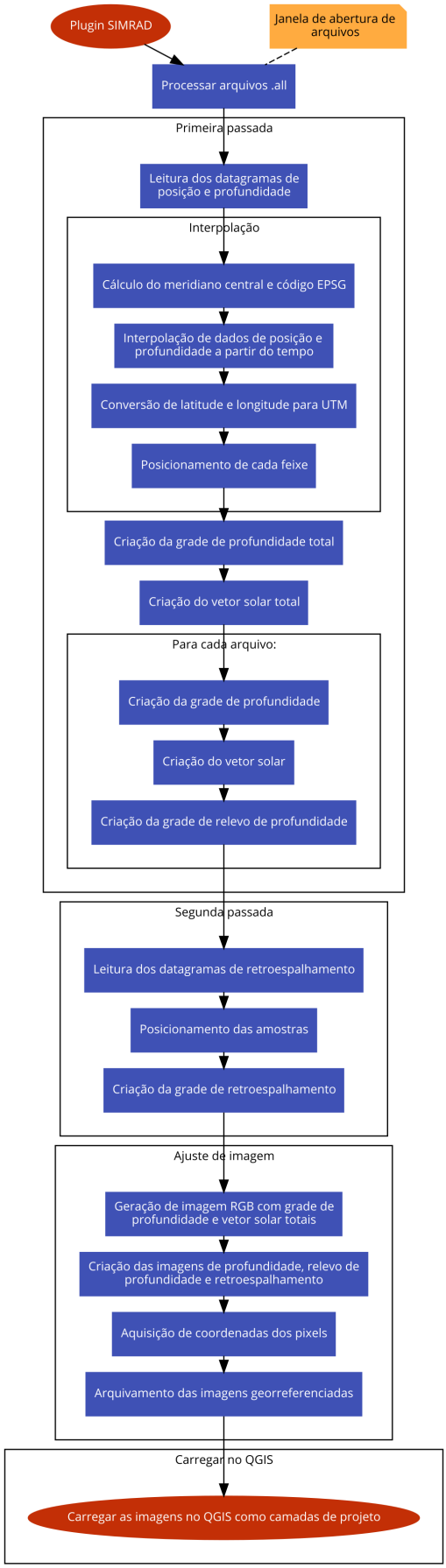


Figura 16 – Fluxograma do script em Python.

Com todos os dados de batimetria posicionados são criadas grades, uma de toda a navegação e uma para cada *linha*, onde são inseridos todos os dados de profundidade referentes às posições correspondentes a cada *bin*. Em cada *bin* é realizado o processamento de mediana, para os *bins* que possuem 3 ou mais valores. Adicionalmente a grade gerada pode ser interpolada para o preenchimento de lacunas.

Para cada grade batimétrica é gerada uma grade de relevo por um processamento de **vetorização solar**, no qual se calcula a projeção do vetor normal à superfície da grade batimétrica em relação a um vetor solar. O resultado da projeção é um fator de luminosidade, e utilizando a batimetria como cor (*hue*) e a projeção como luminância, se cria uma imagem RGB a partir do sistema de cores HSL, com saturação constante.

Nesta etapa, cada grade, tanto de batimetria quanto de relevo batimétrico, é salva como um arquivo de imagem TIFF em ponto flutuante.

3.1.2.2 Leitura dos dados de retroespalhamento

Em um *loop* todos os datagramas de retroespalhamento são lidos e armazenados, mas de uma forma diferente dos dados da primeira passada. Como os dados das amostras de retroespalhamento representam aproximadamente metade de todos os dados presentes no arquivo .all (KONGSBERG, 2002), preferiu-se processar cada arquivo separadamente. São então geradas várias grades, todas com o tamanho de toda a navegação, uma para cada *linha de navegação*.

Para cada uma dessas grades é realizado o processamento de mediana, de forma análoga às grades batimétricas geradas. Todas essas grades são unidas, onde os dados referentes às mesmas posições são unidos em um único *bin*, realizando o processamento de mediana novamente, com a diferença de ser realizado mesmo havendo apenas um valor válido. Adicionalmente a grade gerada pode ser interpolada para o preenchimento de lacunas.

Para o posicionamento das amostras foram utilizados os dados dos datagramas de profundidade, já com as posições, em que a amostra central é posicionada com o feixe correspondente no datagrama de profundidade, e o restante das amostras posicionadas na reta correspondida entre o navio e o feixe, distanciadas pelo produto entre taxa de amostragem e velocidade do som.

A grade gerada é salva em um arquivo de imagem TIFF em ponto flutuante.

3.1.2.3 Carregar no QGIS

Por fim, os arquivos TIFF são georreferenciados e salvos como GeoTIFF em ponto flutuante. Após isso são carregados no QGIS, de uma forma que os valores inválidos não sejam mostrados na camada de projeto.

3.1.3 Instruções de utilização do script como plugin

O script desenvolvido é um plugin do programa QGIS, uma completa ferramenta de manipulação e processamento de dados geográficos para diversos fins. Durante a realização desse estudo, a versão utilizada era a 3.2.3 Bonn executada em um computador com sistema operacional Microsoft Windows 10 Pro.

Para a inserção de scripts Python no QGIS, deve-se habilitar a visualização da caixa de ferramentas de processamento como demonstra a figura 17.

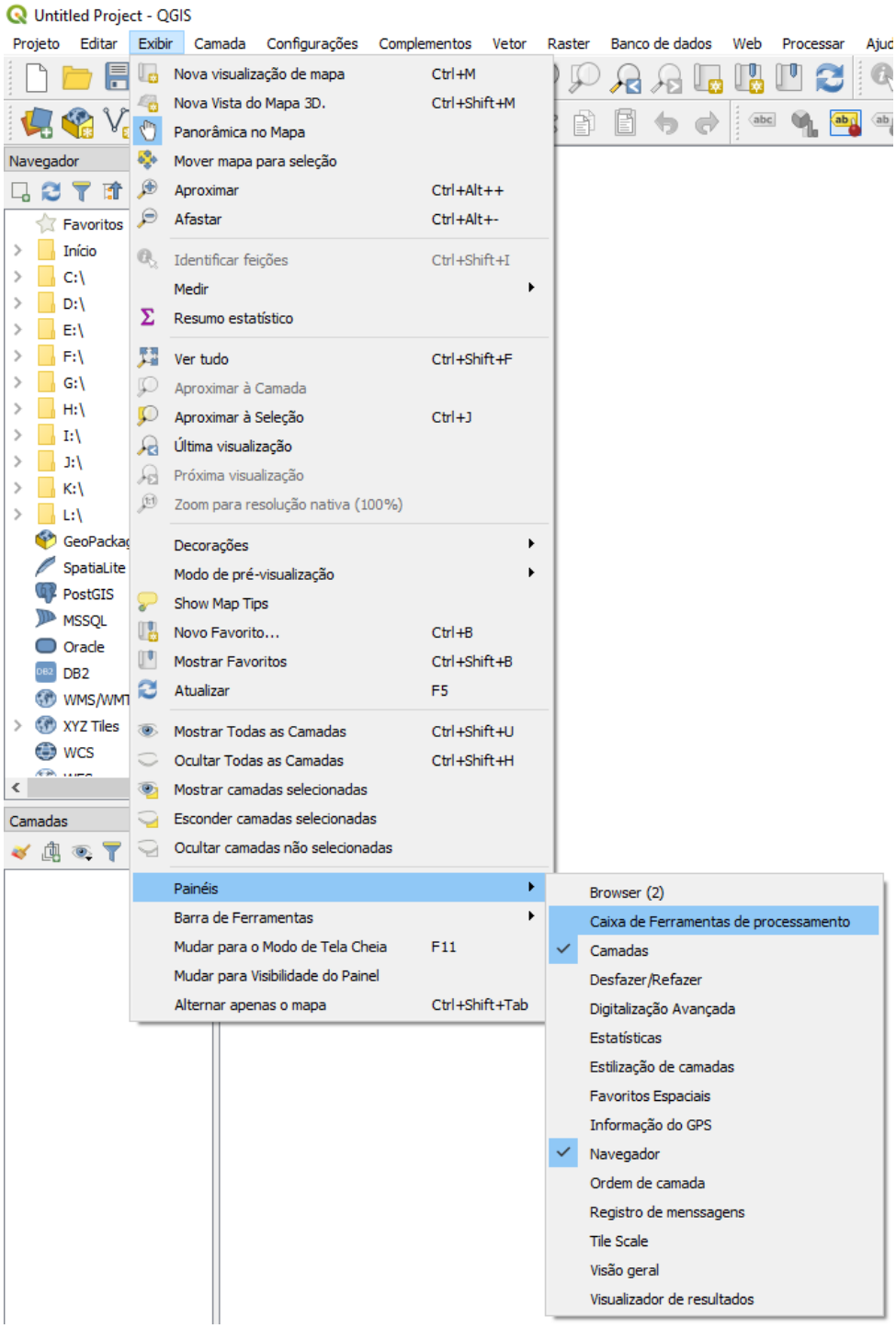


Figura 17 – Imagem de instrução para exibição de painel com caixa de ferramentas

Na caixa de ferramentas de processamento, deve-se localizar o botão com a logo Python e selecionar a opção *Add Script to Toolbox* como visto na figura 18.

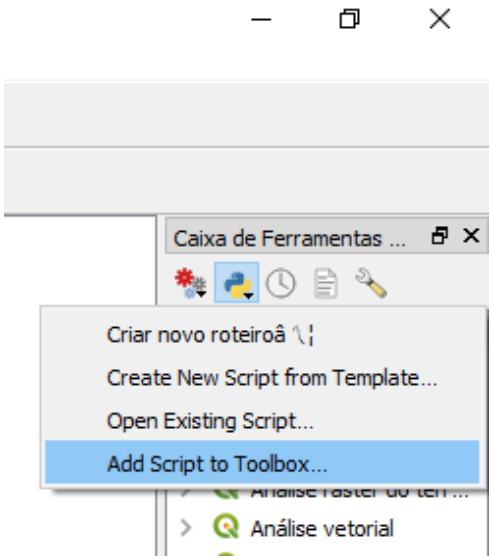


Figura 18 – Imagem de instrução para escolha de ferramenta em menu

Ao selecionar a opção anterior no menu, uma janela se abrirá para seleção do arquivo *.py* com o script. Ao seleciona-lo, o mesmo se encontrará na parte inferior da caixa de ferramenta de processamento anexado ao menu *Scripts* conforme a figura 19.

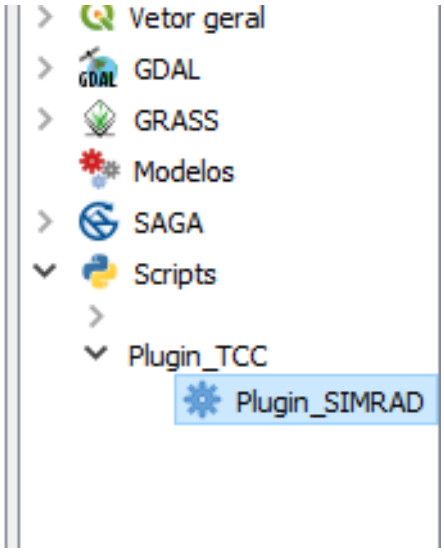


Figura 19 – Imagem de demonstração da localização do script na caixa de ferramentas

Para a inicialização do script, um clique duplo sobre o item no menu demonstrado anteriormente faz-se necessário. Quando o plugin é executado, sua janela de interface com o usuário é adicionada à tela e, como é visto na figura 20, possui algumas opções inerentes ao processamento que sera feito. A primeira opção ao usuário é a escolha do tamanho de pixel em metros para batimetria e retroespalhamento. Logo após, uma caixa de seleção demonstra a opção de dilatação e interpolação ao mapa gerado. Um botão denominado *Processar arquivos .all* é a seguinte opção e, quando ativado, abre uma janela de seleção de

arquivos para o processamento. Uma janela de logs é vista após o botão de processamento e a mesma contém informações úteis sobre as iterações realizadas no script. Por fim, um botão de limpar a janela de logs e um botão para fechar o plugin estão localizados na parte inferior da interface.

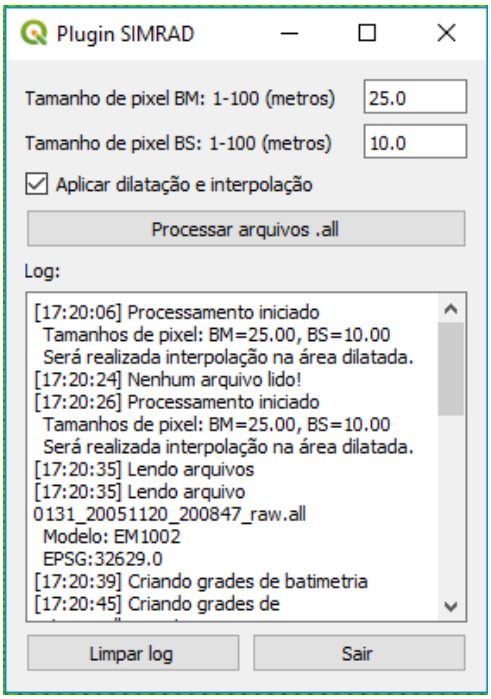


Figura 20 – Imagem de demonstração da interface do plugin desenvolvido

Após o processamento, os mapas de batimetria e retroespalhamento gerados são posicionados no QGIS em forma de camadas e qualquer tipo de manipulação posterior é possível.

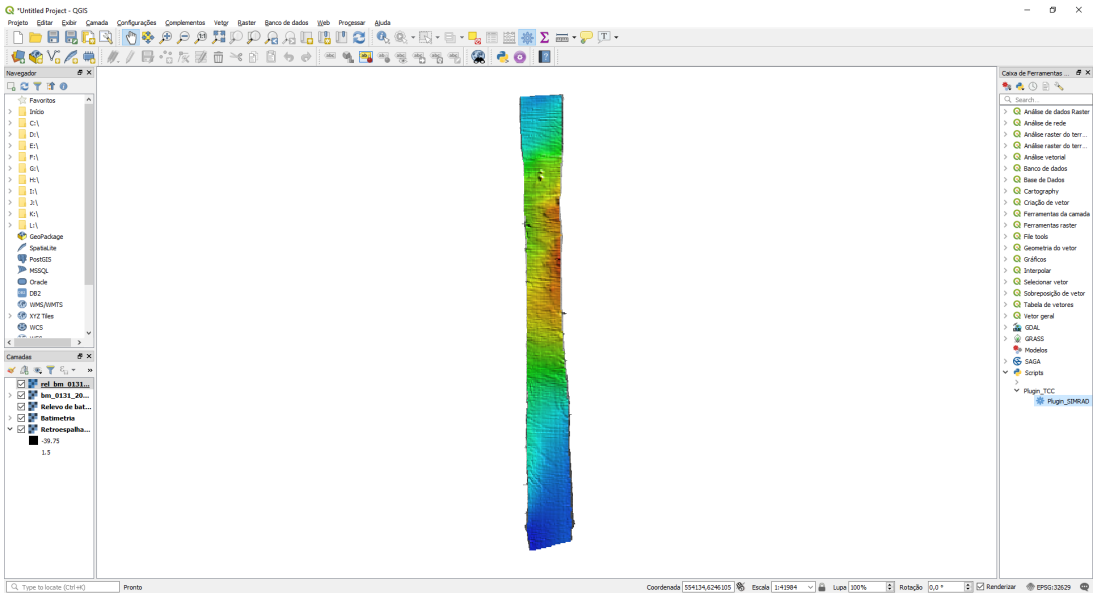


Figura 21 – Imagem de demonstração do resultado em camadas no QGIS

4 Resultados

Os dados analisados foram coletados no Oceano Atlântico, ao norte da Irlanda e a oeste da Escócia, Figura 22, por um sonar multifeixe EM 1002 da Kongsberg durante os dias 20 e 21 de novembro de 2005.

Figura 22 – Localização dos dados coletados. Fonte: GOOGLE (2018)

Para uma análise inicial dos dados de batimetria e retroespalhamento foi utilizado o software de código aberto, disponível para plataformas Linux e MacOS, *MB-System Seafloor Mapping Software*, versão 5.5, a partir de um terminal *bash* na plataforma Ubuntu, versão 16.04 *LTS*.

As instruções de instalação podem ser encontradas na página oficial do software (HOW. . . , 2017), sendo utilizada a instalação a partir do repositório *UbuntuGIS*.

Para se vizualizar os dados foi utilizado o software de código aberto *QGIS*, disponível em várias plataformas, incluindo Linux, Windows e MacOS, versão 3.2, na plataforma Windows 10.

A instalação foi feita a partir do executável encontrado na página oficial do software (QGIS. . . , 2018).

4.1 Dados processados no MB-System

Para se processar os dados no software MB-System é necessário se abrir um terminal na pasta onde estão os arquivos *.all* (ou localizar o diretório usando o comando *cd*). Após isso as seguintes linhas de comando devem ser inseridas no terminal:

```
ls -1 | grep all$ > tmplist
mbdatalist -F-1 -I tmplist > datalist-1
mbdatalist -F-1 -I tmplist -N
```

A primeira linha de comando cria uma arquivo temporário com a listagem de todos os arquivos *.all* do diretório. A segunda linha de comando gera um segundo arquivo que possui a listagem do primeiro mas com o formato e comprimento da grade após cada arquivo. A terceira linha de comando extrai as informações de cada linha de navegação em arquivos auxiliares de *fast navigation* (*.fnv*), *fast bathymetry* (*.fbt*) e *metadados* (*.inf*). Estes arquivos são utilizados na plotagem dos dados (SCHMIDT; CHAYES; CARESS, 2006; FERREIRA, 2015; CARESS; CHAYES; FERREIRA, 2018).

Para se gerar a plotagem dos dados é utilizado o macro *mbm_plot*. Ele possui dois argumentos importantes. O primeiro é se deseja que a navegação seja mostrada no plot, para isso é adicionado *-N* na linha de comando do macro. A segunda é se deseja plotar algum dado de batimetria ou retroespalhamento e qual tipo de dado é esse. Para isso é adicionado *-G*, para se plotar o dado, seguido de um número, que determina o dado. Os dados possíveis estão na tabela 1 (CARESS; CHAYES; FERREIRA, 2018).

Tabela 1 – Dados possíveis de serem plotados a partir do macro *mbm_plot*

Dado a ser plotado	**Valor na opção *-G***
Preenchimento de cores de dados de batimetria	1
Sombreamento de relevos de dados de batimetria	2
Sombreamento de batimetria usando dados de amplitude	3
Preenchimento de escala de cinza de dados de amplitude	4
Preenchimento de escala de cinza de imageamento lateral	5

Fonte: Adaptado de SCHMIDT; CHAYES; CARESS (2006)

4.1.1 Batimetria

São gerados dois plots batimétricos, um com preenchimento de cores e outro com sombreamento de relevos.

Para a geração do primeiro plot devem ser inseridas as seguintes linhas de comando no terminal:

```
mbm_plot -F-1 -I datalist-1 -G1
./ datalist -1.cmd
```

A primeira linha de comando gera um *script* a partir da listagem dos arquivos que será responsável por gerar um plot com a batimetria em um arquivo vetorizado *.ps*. A segunda linha de comando executa esse *script*. A imagem gerada na plotagem é a figura 24a.

Já para o segundo plot são inseridas as seguintes linhas de comando no terminal:

```
mbm_plot -F-1 -I datalist-1 -G2
./ datalist -1.cmd
```

Troca-se apenas a opção *-G1* pela opção *-G2*, gerando a imagem da figura 25a.

4.1.2 Retroespalhamento

São gerados dois plots de retroespalhamento, um utilizando os dados de amplitude e outro utilizando os dados de imageamento lateral.

Para a geração do primeiro plot devem ser inseridas as seguintes linhas de comando no terminal:

```
mbm_plot -F-1 -I datalist-1 -G4
./ datalist -1.cmd
```

As linhas inseridas no terminal são de forma análoga às inseridas para o plot de navegação. A única diferença é o uso da opção *-G4* no lugar da opção *-G1*. A imagem gerada na plotagem é a figura 23a.

4.2 Dados processados na rotina em C e no script em Python

Com a interpretação e separação dos dados dos datagramas da leitura de mapeamento realizado ao norte da Irlanda pelo sonar multifeixe Kongsberg EM 1002, obteve-se arquivos de navegação e arquivos de interpolação de batimetria e retroespalhamento.

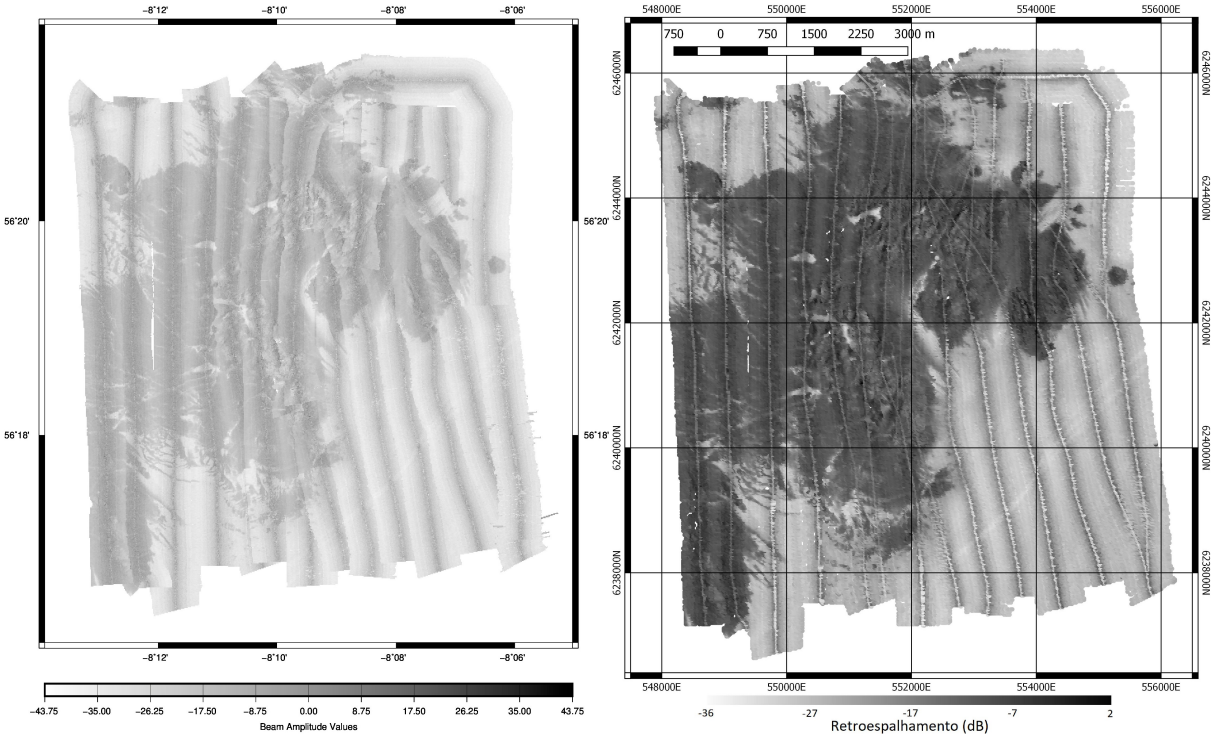
As seguintes imagens demonstram os resultados obtidos pelo *binning* para batimetria (Figura 24b) e retroespalhamento (Figura 23b) com o uso de programa em C anteriormente desenvolvido. Pode-se observar em ambas as imagens uma variação suave nos valores atribuídos no processo de *binning*.

Por comparação, pode-se observar que os resultados gerados pelo processo de *binning* no processamento com o uso do código em C desenvolvido anteriormente são muito similares ao resultados alcançados pelo *MB-System*. Com relação ao mapa de batimetria e batimetria com relevo gerados pelo script Python desenvolvido, observa-se que os resultados obtidos são semelhantes aos anteriores no que diz respeito a características gerais

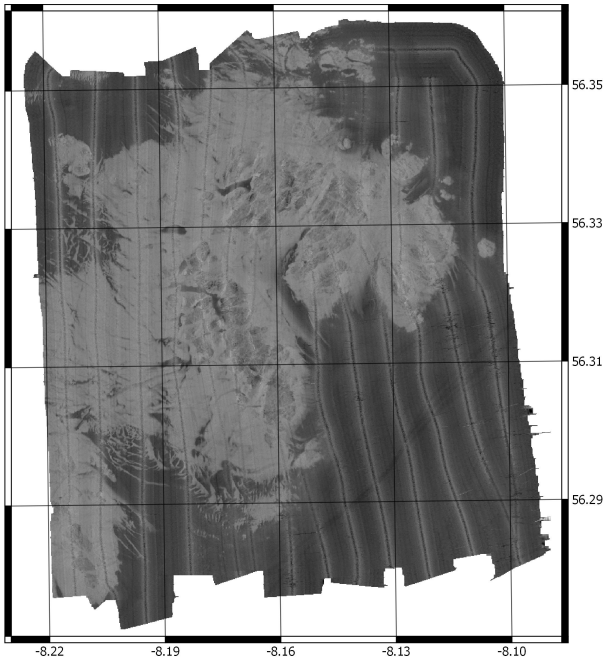
do solo analisado. Ao analisar as linhas individuais, é possível a observação mais clara de características do solo com uma maior resolução.

O mapa de retroespalhamento resultante em C e no *MB-System* demonstram-se bastante similares. Já o mesmo mapa gerado pelo script Python possui uma resolução e exibição de detalhes superiores aos anteriores por conta do uso de série temporal.

Com o processamento pelo *script* em Python desenvolvido, o resultado pode ser demonstrado nas Figuras 24c, 25b e 23c.

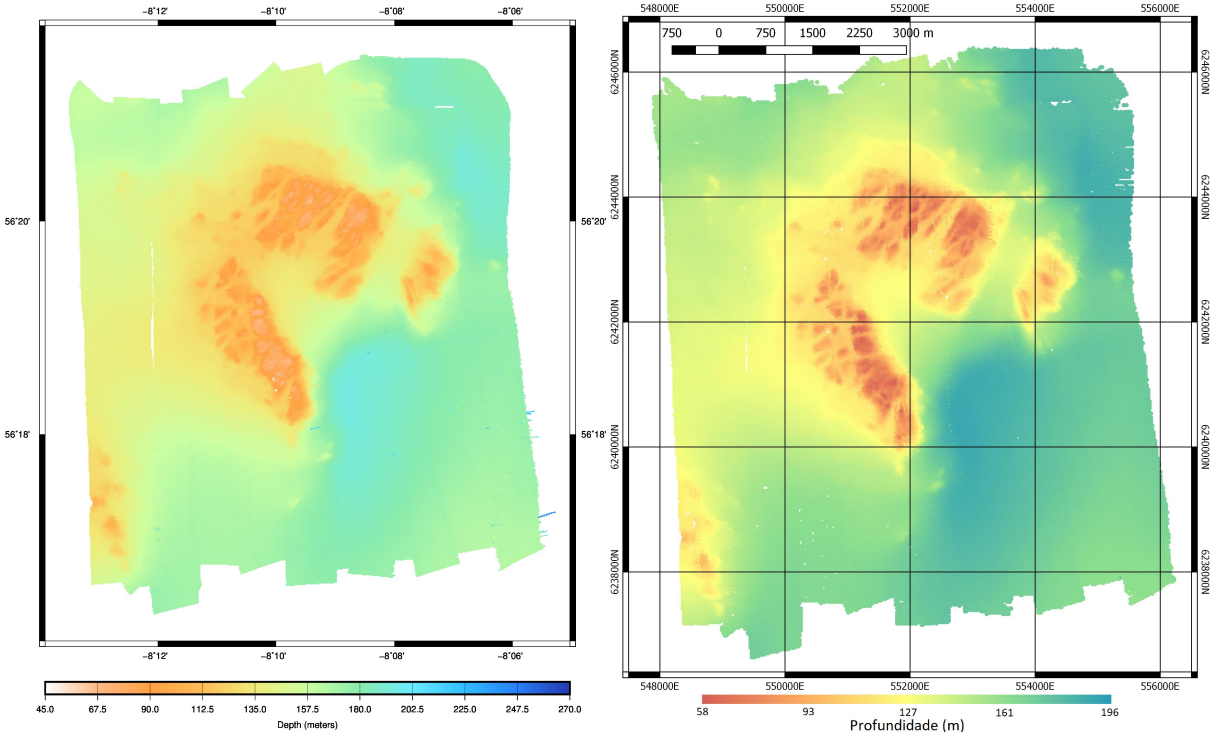


(a) *MB-System* (b) Rotina em C

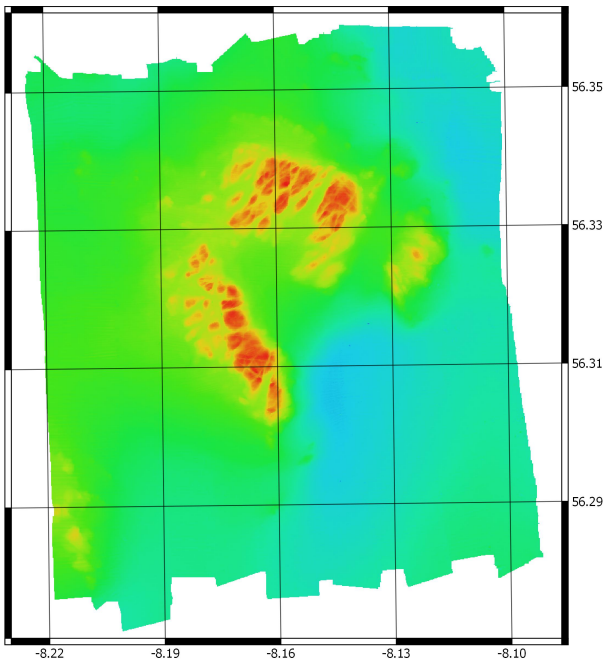


(c) Script em Python

Figura 23 – Resultados de retroespalhamento

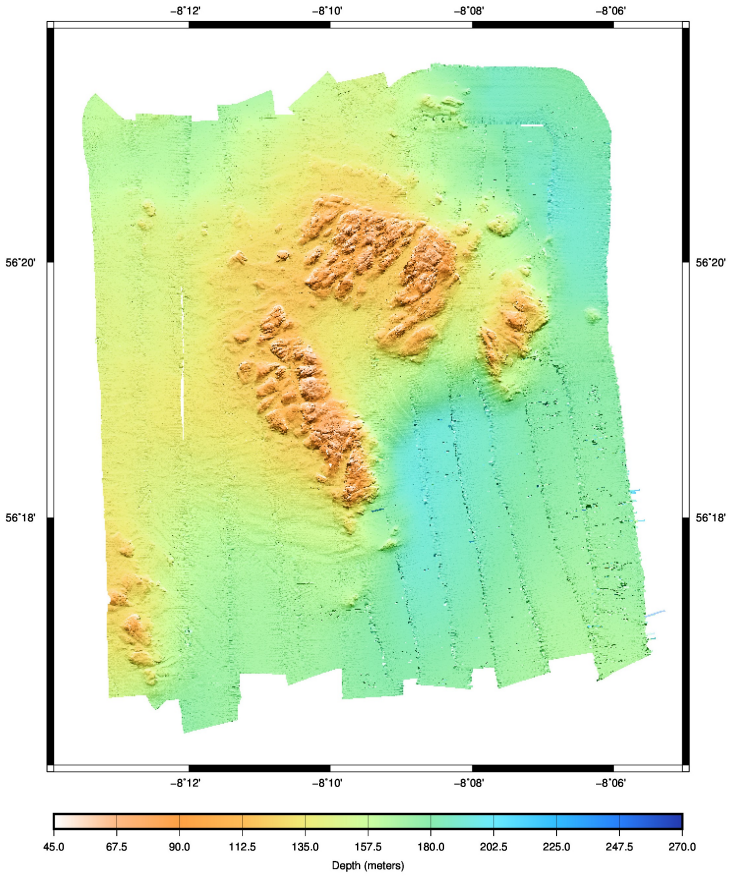


(a) *MB-System* (b) Rotina em C

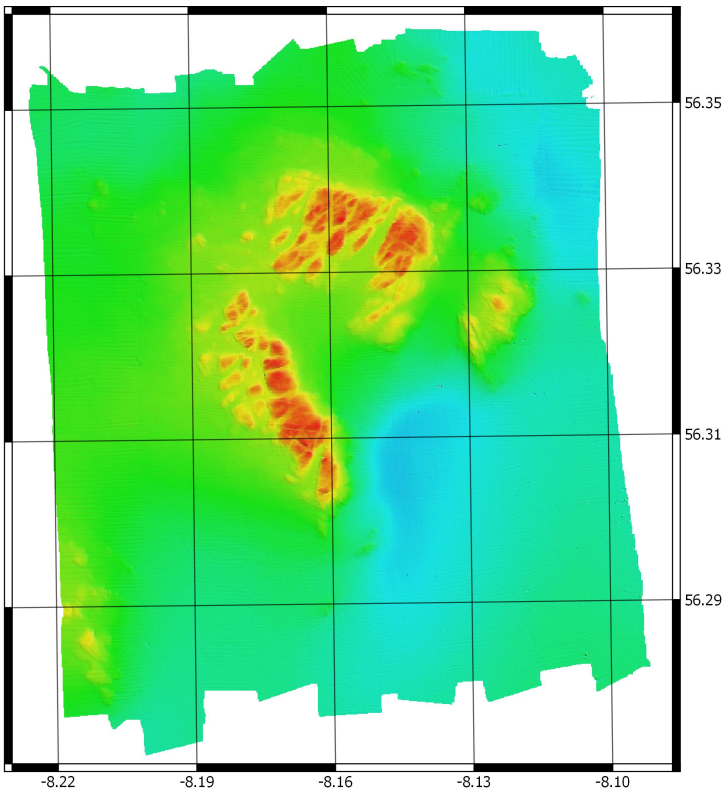


(c) Script em Python

Figura 24 – Resultados de batimetria



(a) *MB-System*



(b) Script em Python

Figura 25 – Resultados de relevo de batimetria

5 Conclusão

O estudo desenvolvido proporcionou a correta leitura e interpretação de arquivos em formato de datagrama relativos a dados obtidos de sonares multifeixes, além de apresentar uma abordagem simples e estatística dos dados interpretados com um pré-processamento utilizando *binning* de dados em linguagem C e paralelamente em linguagem Python.

Ao observar os resultados do pré-processamento desenvolvido, percebe-se que tanto a técnica de *binning* em C, com análise estatística dos dados, quanto as correções desenvolvidas demonstraram-se bastante úteis e as imagens obtidas apresentam ruído mínimo e qualidade satisfatória. No entanto, o resultado do retroespalhamento obtido com essa rotina não é otimizado devido ao número de amostras por feixe, visto que no pré-processamento somente uma amostra foi utilizada. Logo, com a redução dos dados superamostrados pelo processo de *binning* e análises de característica mediana das incidências mapeadas, pode-se concluir que essa primeira abordagem com rotina em linguagem C pôde demonstrar uma metodologia única e eficaz para um desfecho adequado.

A respeito do script em linguagem Python desenvolvido, o resultado de sua utilização juntamente com o programa QGIS foi o processamento de retroespalhamento utilizando a série temporal, permitindo uma maior resolução por haver um número consideravelmente maior de amostras. Quanto à batimetria, foi realizado a criação de um modelo batimétrico com efeito tridimensional com a utilização de um artefato de iluminação por manipulação vetorial com um vetor solar e os vetores normais ao relevo, gerando um mapa muito mais detalhado e preciso quanto a aspectos de forma do solo analisado. O mapa batimétrico gerado com o plugin desenvolvido apresenta potencial de aplicação em áreas de estudo mais específicos que exijam um detalhamento maior, como é o caso de diversas situações na área militar quando faz-se necessário a identificação de objetos metálicos no fundo do mar que possam ser de elevada importância ou em áreas comerciais e de estudo acadêmico, já que esse tipo de processamento pode demonstrar embarcações afundadas, aberturas de poços e particularidades de rochas, bem como falhas geológicas pequenas que quando analisadas juntamente com o mapa de retroespalhamento na mesma localização, podem fornecer ricas informações para se chegar a conclusões a respeito de sua origem, evolução e influência macroscópica na região em que se encontra.

Referências

BLONDEL, P. *The handbook of sidescan sonar*. [S.l.]: Springer Science & Business Media, 2010. Citado na página 16.

CARESS, D.; CHAYES, D. N.; FERREIRA, C. *List of MB-System Programs*. 2018. Disponível em: <http://www3.mbari.org/products/mbsystem/html/mbsystem_man_list.html>. Acesso em: 27 de maio de 2018. Citado na página 34.

CHAKRABORTY, B.; FERNANDES, W. Bathymetric techniques and indian ocean applications. In: BLONDEL, P. (Ed.). *Bathymetry and Its Applications*. Rijeka: InTech, 2012. cap. 1. Disponível em: <https://doi.org/10.5772/35790>. Citado na página 13.

CREASEY, D. Similarities and differences in signal processing for radar and sonar. In: IET. *Digital Signal Processing for Radar and Sonar Applications, Tutorial Meeting on*. [S.l.], 1990. p. 1–1. Citado na página 15.

FERREIRA, C. Mb-system training course. v. 1, 2015. Citado na página 34.

FONSECA, L.; MAYER, L. Remote estimation of surficial seafloor properties through the application angular range analysis to multibeam sonar data. *Marine Geophysical Researches*, Springer, v. 28, n. 2, p. 119–126, 2007. Citado 2 vezes nas páginas 10 e 14.

GONZALEZ, R. C.; WOODS, R. E. *Processamento Digital de Imagens*. 3. ed. [S.l.]: Pearson, 2009. ISBN 978-8576054016. Citado na página 27.

GOOGLE. *Google Maps*. 2018. Disponível em: <https://www.google.com.br/maps/>. Acesso em: 27 de maio de 2018. Citado na página 33.

HAMMERSTAD, E. *EM Technical Note: Backscattering and Seabed Image Reflectivity*. [S.l.]: Kongsberg Maritime AS, 2000. Citado 2 vezes nas páginas 18 e 21.

HOW to Download and Install MB-System. 2017. Disponível em: <https://www.mbari.org/products/research-software/mb-system/how-to-download-and-install-mb-system/>. Acesso em: 28 de julho de 2018. Citado na página 33.

KENNY, A. et al. An overview of seabed-mapping technologies in the context of marine habitat classification. *ICES Journal of Marine Science*, Oxford University Press, v. 60, n. 2, p. 411–418, 2003. Citado 2 vezes nas páginas 16 e 17.

KONGSBERG. *Operator manual: EM 1002, hydrography multibeam echo sounder*. Kongsberg Maritime AS, 2002. Disponível em: <https://www.km.kongsberg.com/ks/web/nokbg0397.nsf/AllWeb/7158703C4046C49FC1257F5B0036BD44/$file/160977-EM1002-Operator-manual.pdf>. Citado na página 29.

KONGSBERG. *Aplication note: Discovering the redefined EM multibeam series*. Kongsberg Maritime AS, 2013. Disponível em: <https://www.km.kongsberg.com/ks/web/nokbg0397.nsf/AllWeb/1601A6EC456EEFD9C1257C75004B6EC4/$file/Kongsberg_Application_Note_Discovering-the-redefined-EM-series.pdf>. Citado na página 18.

KONGSBERG. *Aplication note: PulSAR - high resolution sidescan sonar*. Kongsberg Maritime AS, 2015. Disponível em: <https://www.km.kongsberg.com/ks/web/nokbg0397.nsf/AllWeb/AA92795D4BB19721C1257EF8003DB311/$file/Kongsberg-PulSAR-applications.pdf>. Citado na página 16.

KONGSBERG. *Series Multibeam echo sounders datagram formats*. [S.l.]: Kongsberg Maritime AS, 2015. Citado na página 19.

KRZYWINSKI, M. Binning high-resolution data. v. 13, 2016. Citado na página 22.

LURTON, X. et al. Backscatter measurements by seafloor-mapping sonars: guidelines and recommendations. *A collective report by members of the GeoHab Backscatter Working Group*, n. May, p. 1–200, 2015. Citado 6 vezes nas páginas 11, 13, 14, 18, 20 e 21.

MAYER, L.; LI, Y.; MELVIN, G. 3d visualization for pelagic fisheries research and assessment. v. 59, p. 216–225, 01 2002. Citado na página 17.

MOUSTIER, C. de; MATSUMOTO, H. Seafloor acoustic remote sensing with miltibeam echo-sounders and bathymetric sidescan sonar systems. *Marine Geophysical Research*, v. 15, p. 411–418, 1993. Citado na página 10.

PENROSE, J. et al. Acoustic techniques for seabed classification. *Cooperative Research Centre for Coastal Zone Estuary and Waterway Management, Technical Report*, v. 32, 2005. Citado 4 vezes nas páginas 11, 15, 16 e 17.

PYQGIS Developer Cookbook. 2018. Disponível em: <https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/>. Acesso em: 28 de julho de 2018. Citado na página 27.

QGIS Um Sistema de Informação Geográfica livre e aberto. 2018. Disponível em: <https://www.qgis.org/pt_BR/site/>. Acesso em: 28 de julho de 2018. Citado na página 33.

SAYAD, S. *Binning*. 2011. Disponível em: <http://www.saedsayad.com/binning.htm>. Acesso em: 09 de junho de 2018. Citado na página 22.

SCHMIDT, V.; CHAYES, D.; CARESS, D. The mb-system cookbook. v. 1, 2006. Citado na página 34.

STERGIOPOULOS, S. *Advanced signal processing handbook: theory and implementation for radar, sonar, and medical imaging real time systems*. [S.l.]: CRC press, 2000. Citado na página 14.

WICKLIN, R. *A simple implementation of two-dimensional binning*. 2013. Disponível em: <https://blogs.sas.com/content/iml/2013/07/17/2d-binning.html>. Acesso em: 15 de junho de 2018. Citado na página 22.

Apêndices

APÊNDICE A – *Resultados do script em Python*

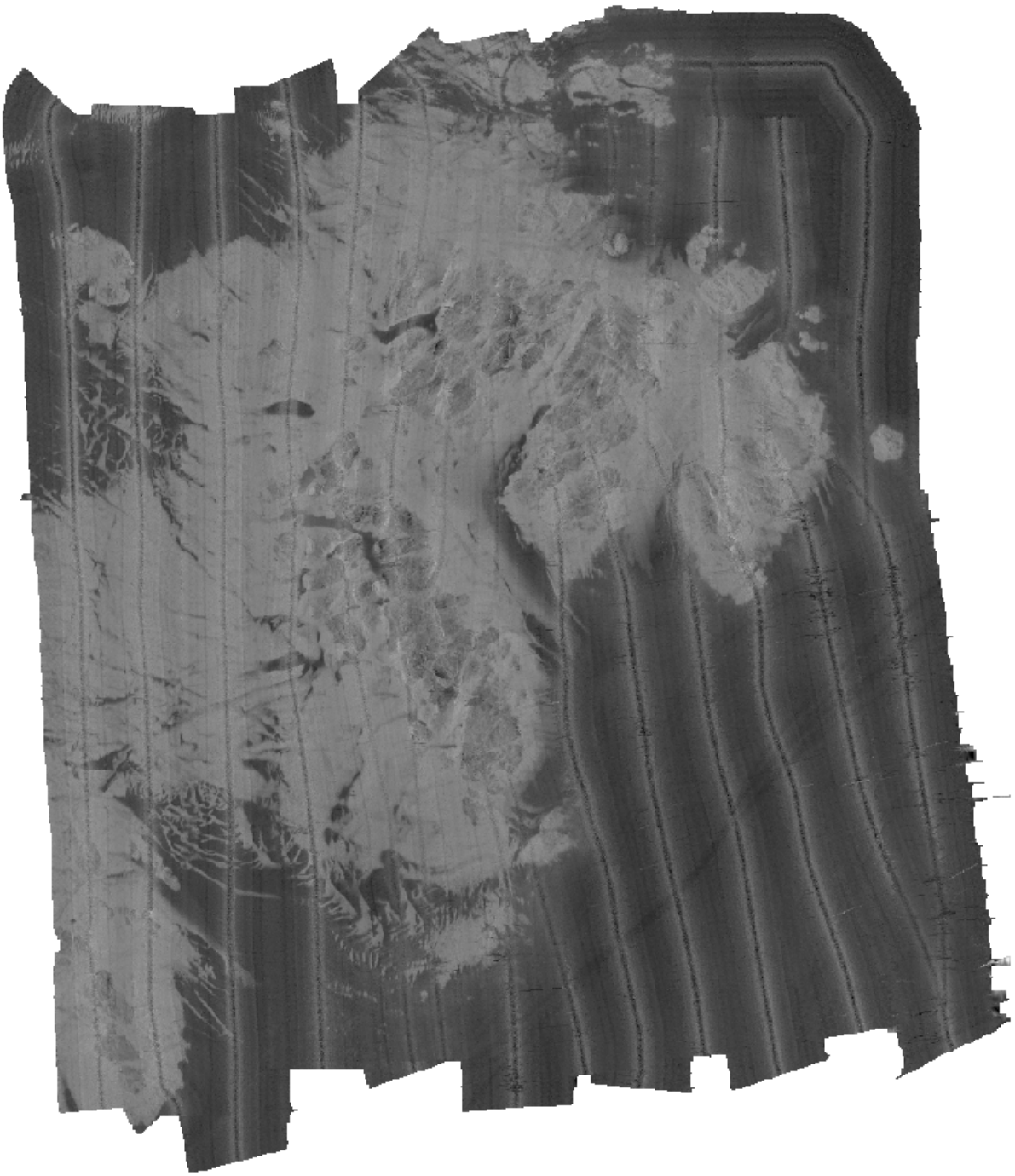


Figura 26 – Retroespalhamento de toda a navegação

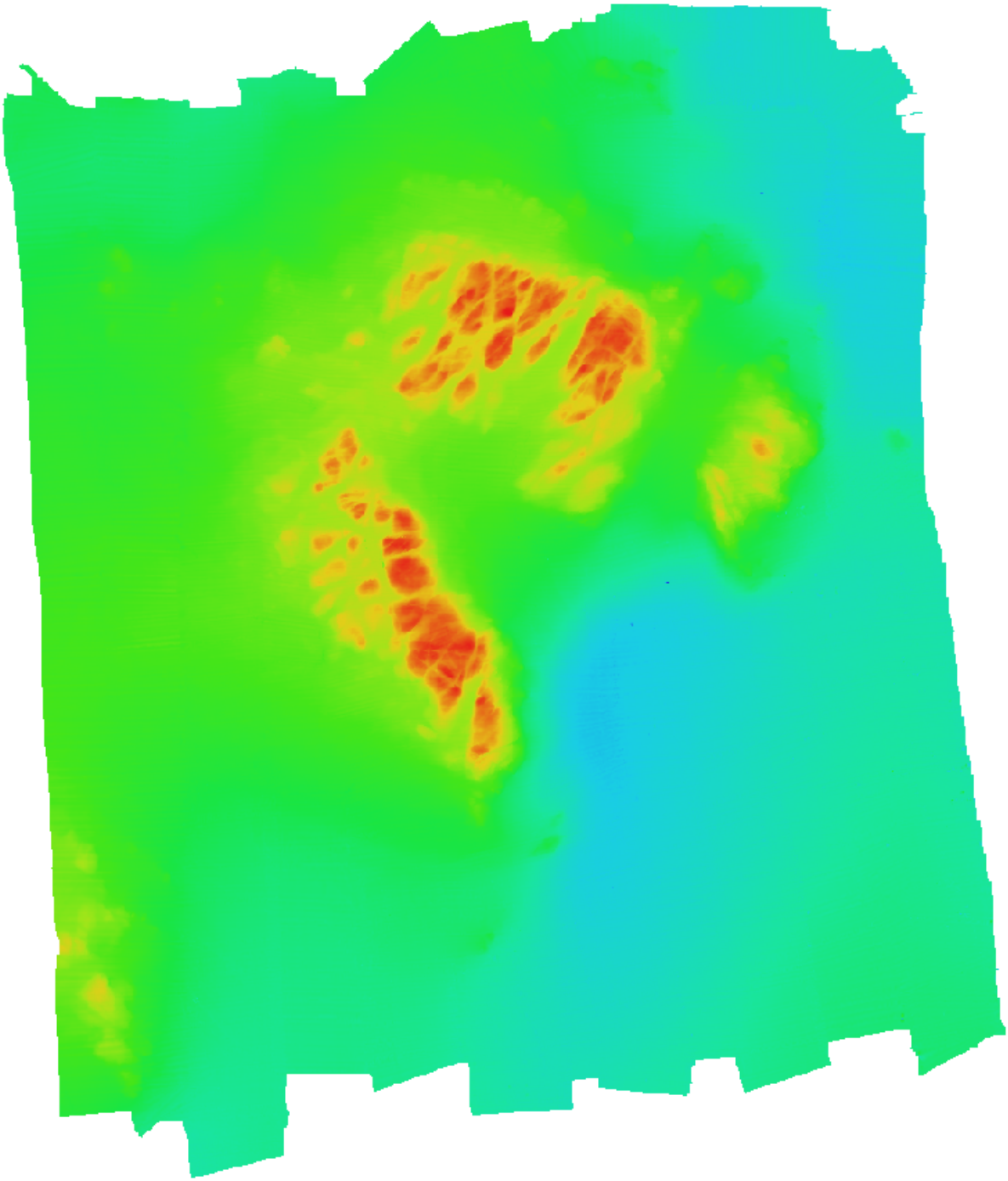


Figura 27 – Batimetria de toda a navegação

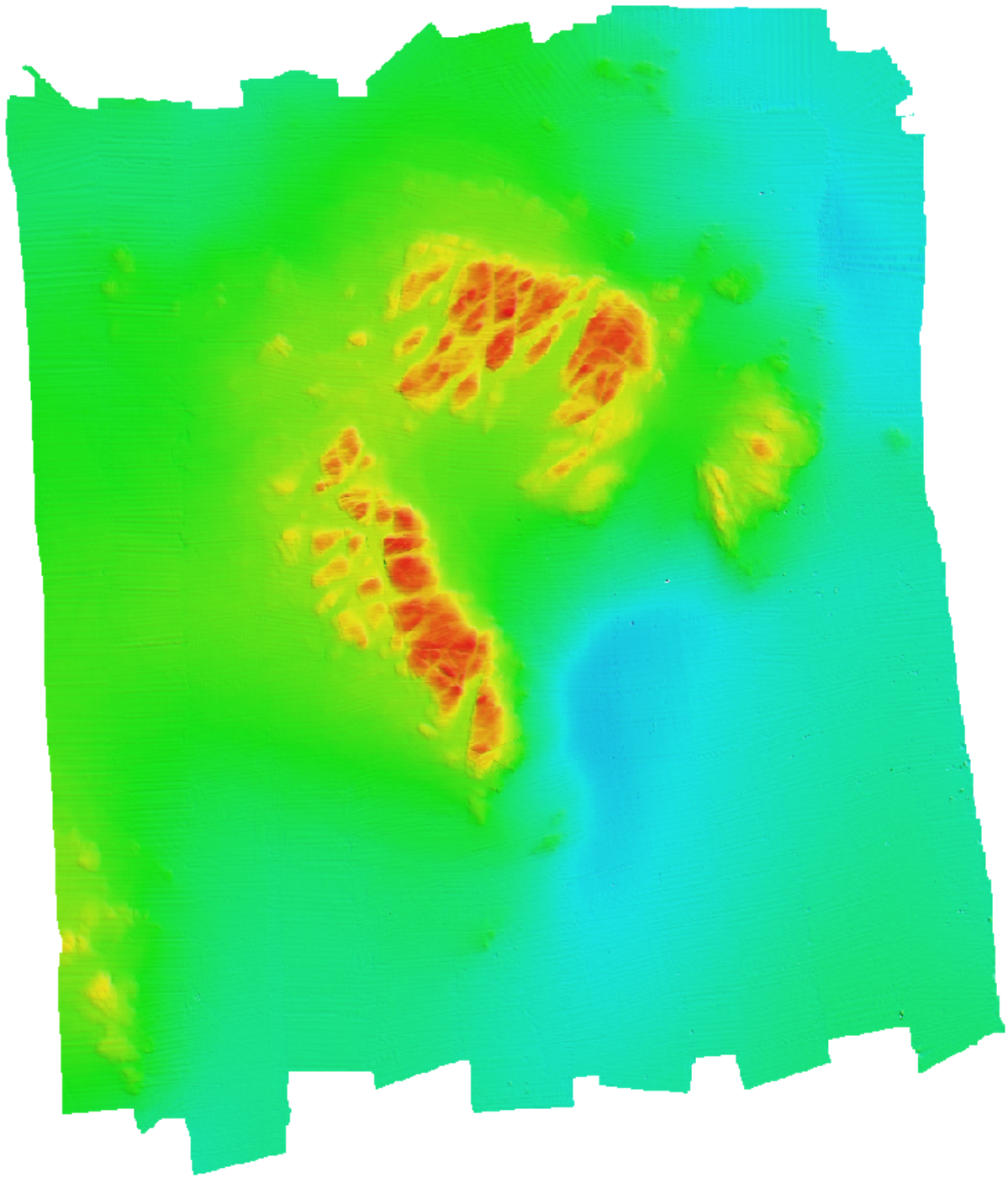


Figura 28 – Relevo de batimetria de toda a navegação

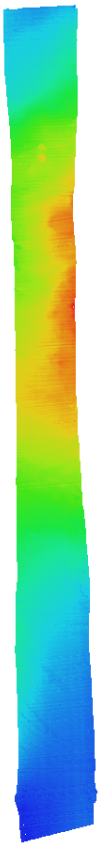


Figura 29 – Batimetria
0131_20051120_200847_raw.all

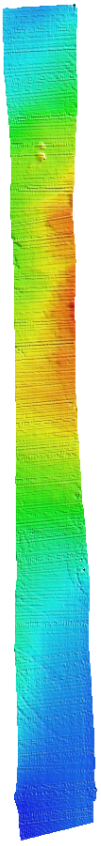


Figura 30 – Relevo
0131_20051120_200847_raw.all

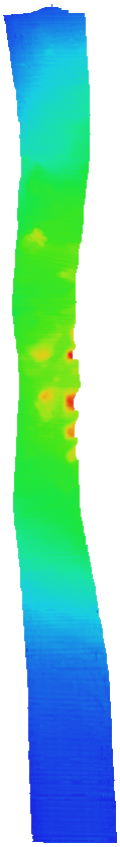


Figura 31 – Batimetria
0132_20051120_204947_raw.all

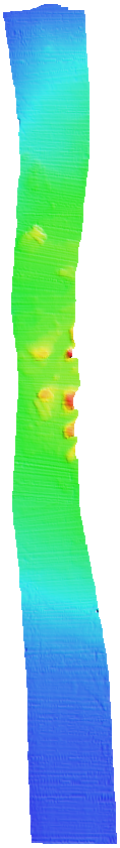


Figura 32 – Relevo
0132_20051120_204947_raw.all

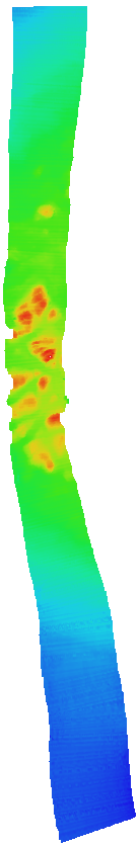


Figura 33 – Batimetria
0133_20051120_212619_raw.all

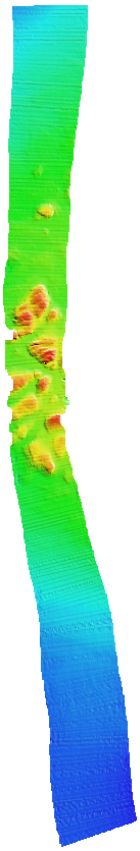


Figura 34 – Relevo
0133_20051120_212619_raw.all

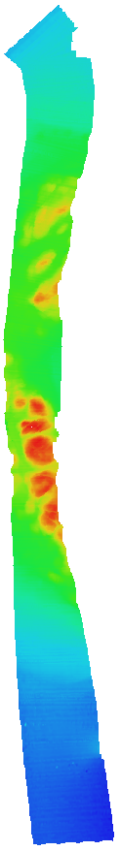


Figura 35 – Batimetria
0134_20051120_215934_raw.all

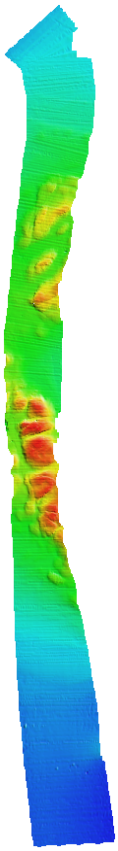


Figura 36 – Relevo
0134_20051120_215934_raw.all

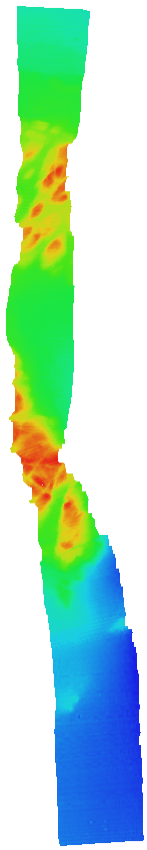


Figura 37 – Batimetria
0135_20051120_223400_raw.all

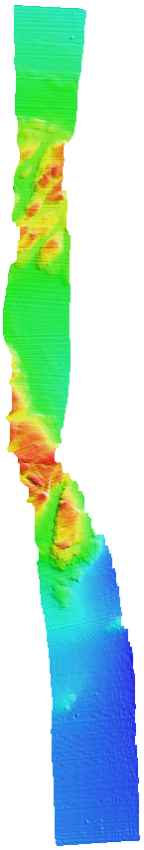


Figura 38 – Relevo
0135_20051120_223400_raw.all

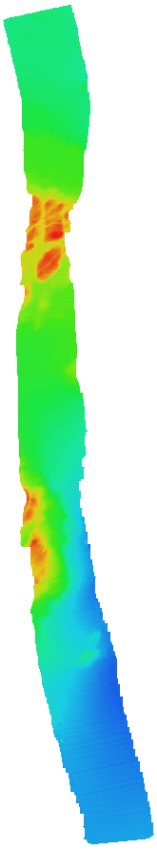


Figura 39 – Batimetria
0136_20051120_230803_raw.all

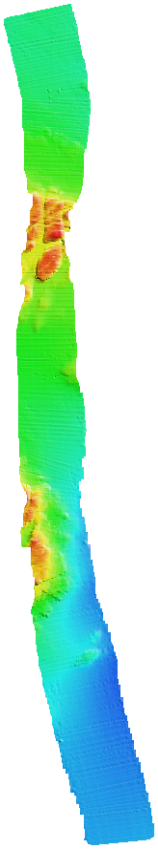


Figura 40 – Relevo
0136_20051120_230803_raw.all

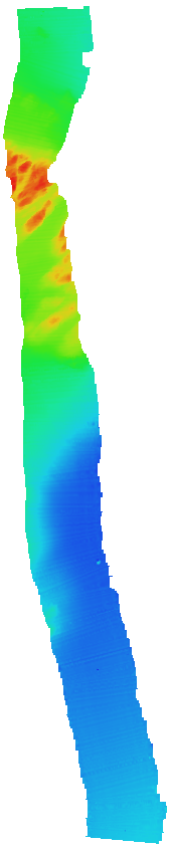


Figura 41 – Batimetria
0137_20051120_234236_raw.all

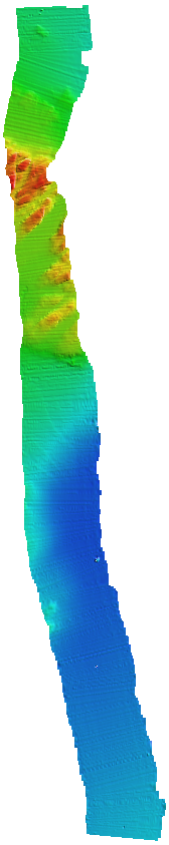


Figura 42 – Relevo
0137_20051120_234236_raw.all

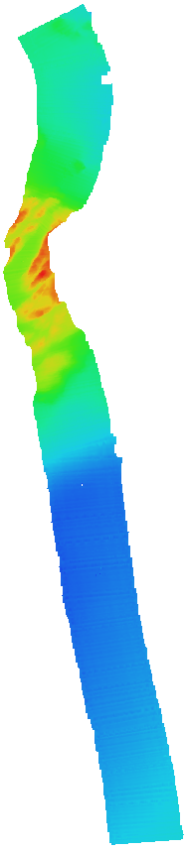


Figura 43 – Batimetria
0138_20051121_001414_raw.all

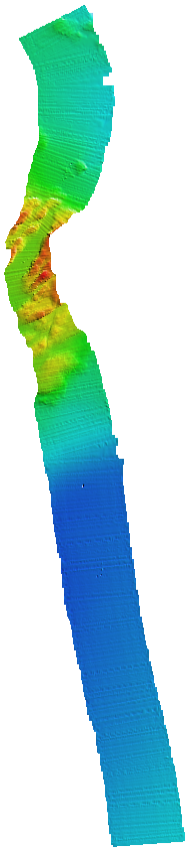


Figura 44 – Relevo
0138_20051121_001414_raw.all

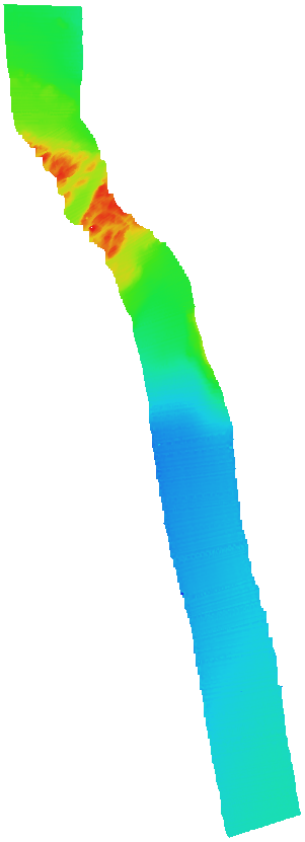


Figura 45 – Batimetria
0139_20051121_004754_raw.all

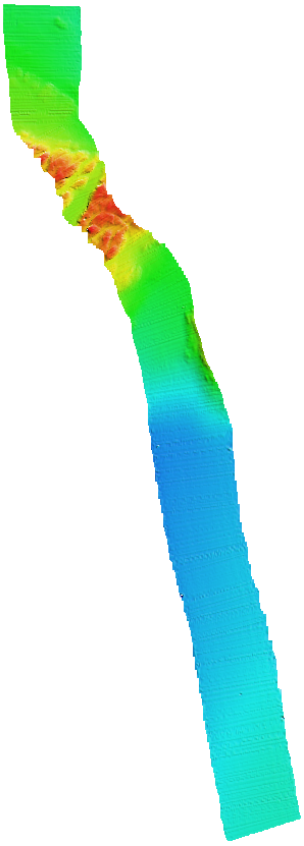


Figura 46 – Relevo
0139_20051121_004754_raw.all

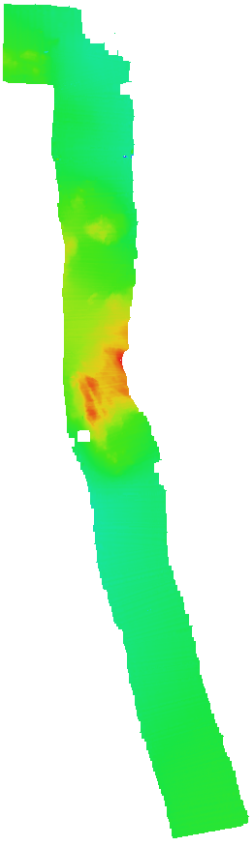


Figura 47 – Batimetria
0140_20051121_012138_raw.all

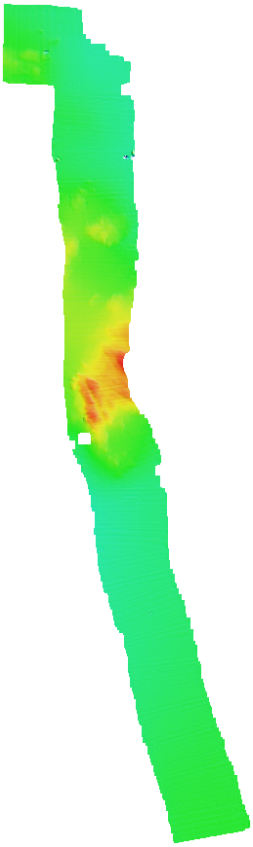


Figura 48 – Relevo
0140_20051121_012138_raw.all

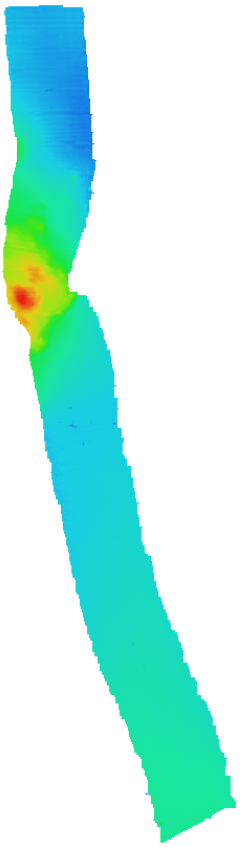


Figura 49 – Batimetria
0141_20051121_015708_raw.all

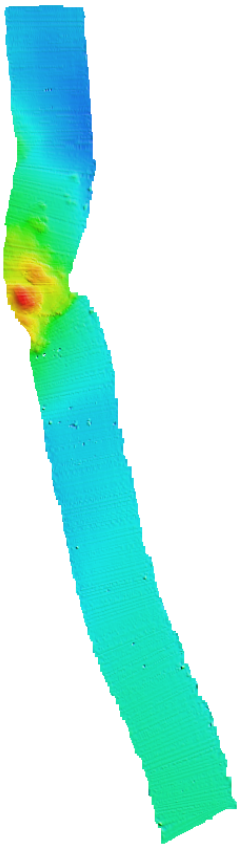


Figura 50 – Relevo
0141_20051121_015708_raw.all

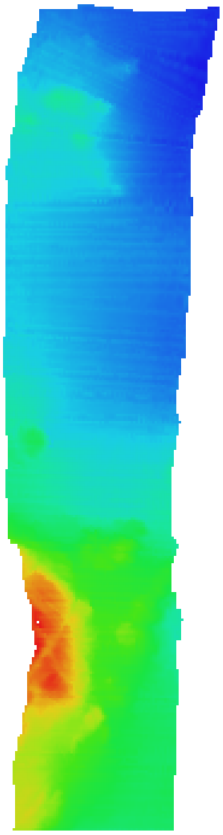


Figura 51 – Batimetria
0142_20051121_023317_raw.all

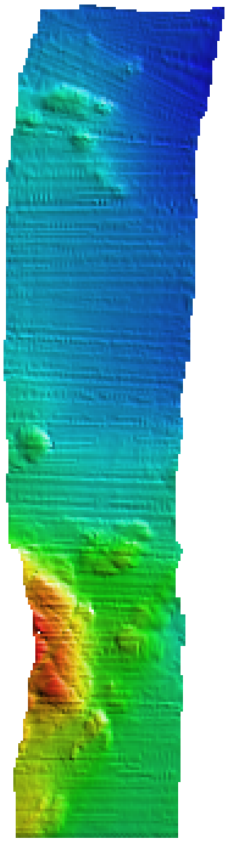


Figura 52 – Relevo
0142_20051121_023317_raw.all

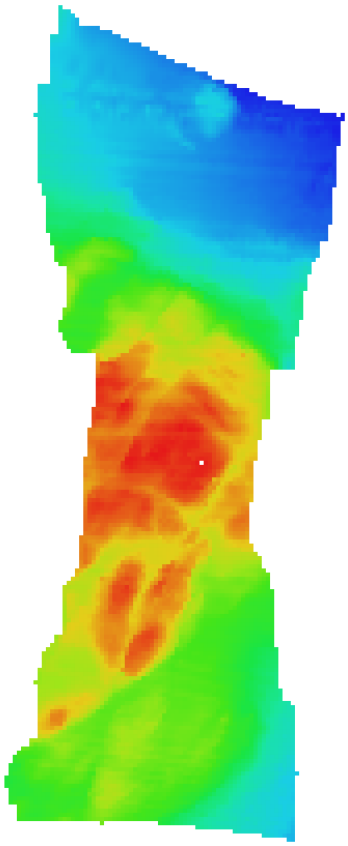


Figura 53 – Batimetria
0143_20051121_024838_raw.all

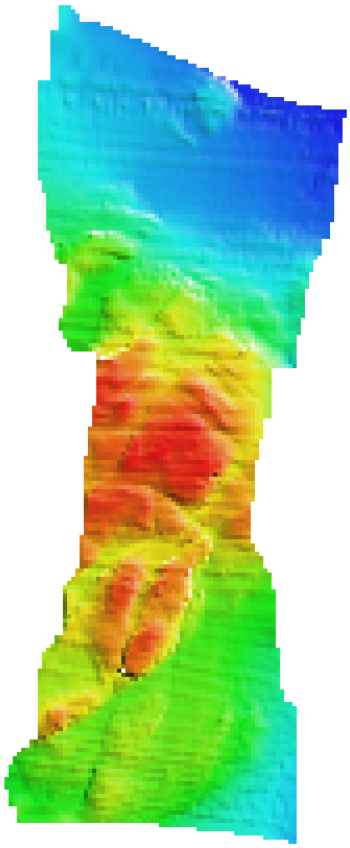


Figura 54 – Relevo
0143_20051121_024838_raw.all

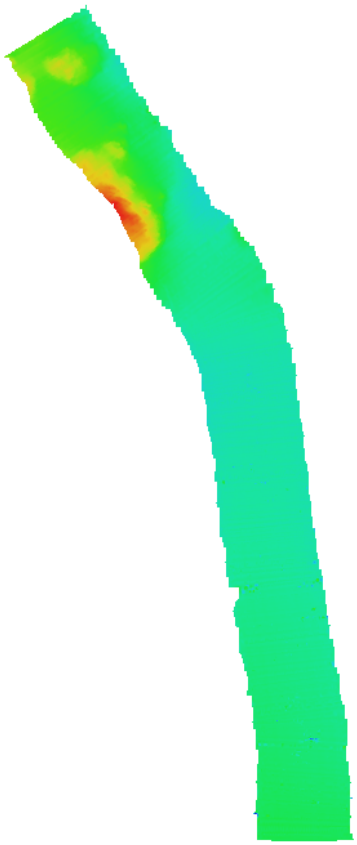


Figura 55 – Batimetria
0144_20051121_030037_raw.all

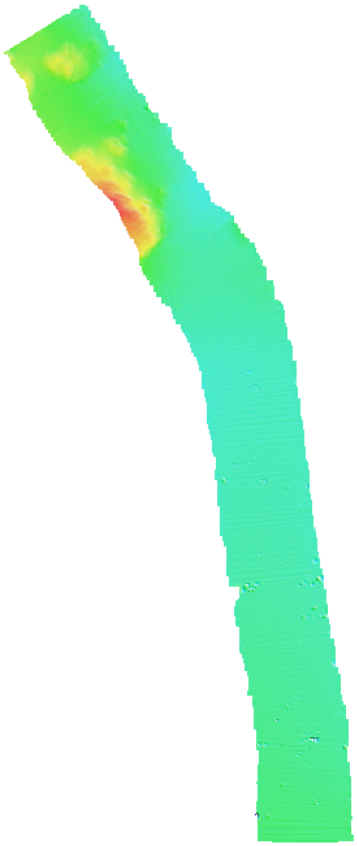


Figura 56 – Relevo
0144_20051121_030037_raw.all

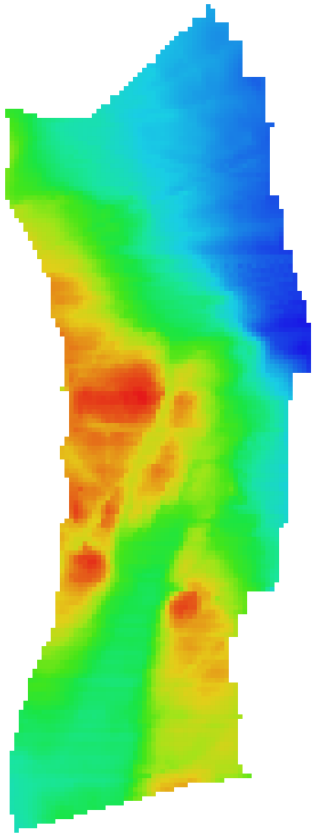


Figura 57 – Batimetria
0146_20051121_210230_raw.all

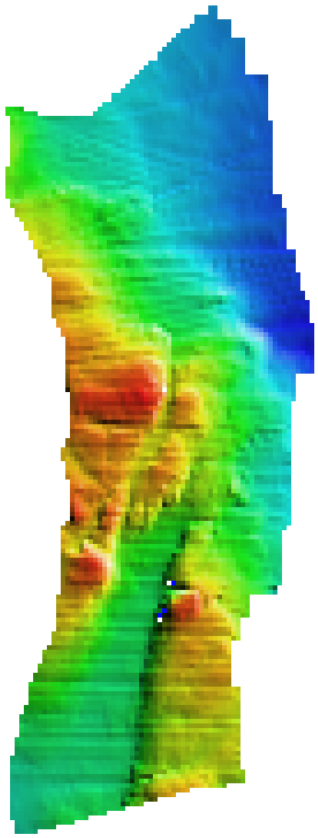


Figura 58 – Relevo
0146_20051121_210230_raw.all

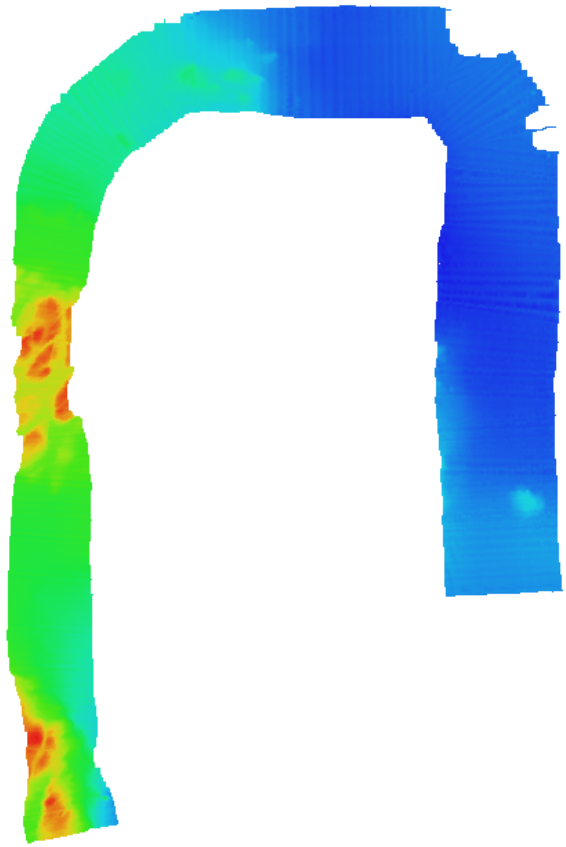


Figura 59 – Batimetria
0147_20051121_211231_raw.all

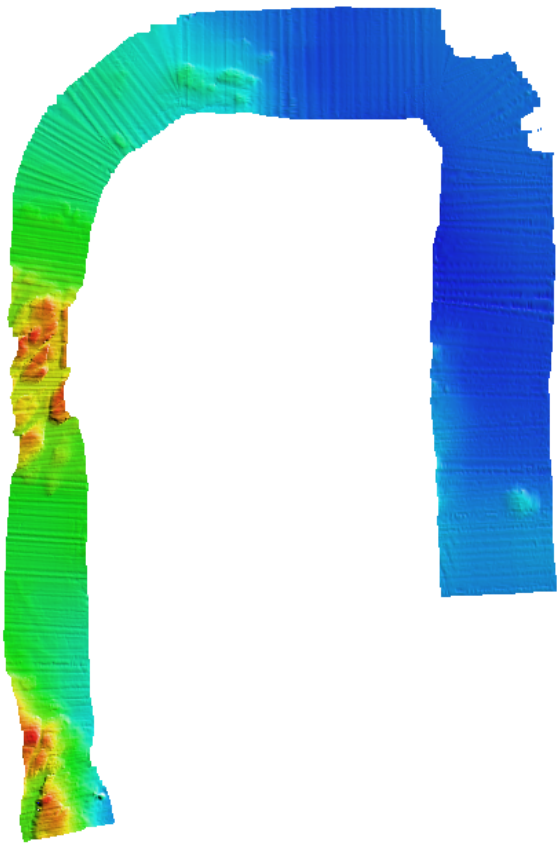


Figura 60 – Relevo
0147_20051121_211231_raw.all

Figura 61 – Batimetria
0149_20051121_225547_raw.all

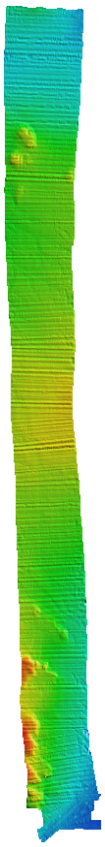


Figura 62 – Relevo
0149_20051121_225547_raw.all

Figura 63 – Batimetria
0150_20051121_233208_raw.all

Figura 64 – Relevo
0150_20051121_233208_raw.all

APÊNDICE B – *Rotina em C*

Listagem B.1 – *structs.c*

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>


#define TRUE 1
#define FALSE 0
#define PI 3.14159265359
#define IN_FDR "Inputs/" //Pasta de entrada
#define OUT_FDR "Outputs/" //Pasta de saída
#define INV_BIN -9999 //Valor atribuído a um bin inválido

typedef enum { //Definição dos modelos de sonares
  EM1002 = 1002,
  EM120 = 120,
  EM300 = 300,
  EM710 = 710,
  EM2000 = 2000,
  EM3000 = 3000,
  EM3002 = 3020,
  EM302 = 302,
  EM122= 122,
  EM121A = 121,
  ME70BO = 850,
  EM2040 =2040,
  EM2040C =2045
} model_id;

typedef enum { //Definição dos tipos de datagramas
  SIMRAD96_ISVP = 0x76,
  SIMRAD96_REMOTE_IPARAM = 0x70,
  SIMRAD96_NETATTITUDE = 0x6e,
  SIMRAD96_WATERCOLUMNI = 0x6b,
  SIMRAD96_WATERCOLUMNH = 0x6b,
  SIMRAD96_STOP_IPARAM = 0x69,
  SIMRAD96_STOP_IPARAMPREA = 0x69,
  SIMRAD96_HEIGHT = 0x68,
  SIMRAD96_RAWRANGE = 0x66,
  SIMRAD96_RANGE = 0x65,
  SIMRAD96_BACKSCATTER = 0x59,
  SIMRAD96_XYZ = 0x58,
  SIMRAD96_SVP = 0x56,
  SIMRAD96_HRSVP = 0x55,
  SIMRAD96_TIDE = 0x54,
  SIMRAD96_IMAGERYF = 0x53,
  SIMRAD96_IMAGERYB = 0x53,
  SIMRAD96_IMAGERYPREA = 0x53,
  SIMRAD96_RUNTIMEM = 0x52,
  SIMRAD96_RUNTIMEI = 0x52,
  SIMRAD96_RUNTIMEF = 0x52,
  SIMRAD96_RUNTIMEB = 0x52,
  SIMRAD96_RUNTIMEA = 0x52,
  SIMRAD96_RUNTIMEPREA = 0x52,
```

```
  SIMRAD96_POSITION = 0x50,
  SIMRAD96_POSITIONPREA = 0x50,
  SIMRAD96_RAWRANGE78 = 0x4e,
  SIMRAD96_ECHOGRAMS = 0x4b,
  SIMRAD96_MECHTILT = 0x4a,
  SIMRAD96_START_IPARAM = 0x49,
  SIMRAD96_START_IPARAMPREA = 0x49,
  SIMRAD96_HEADING = 0x48,
  SIMRAD96_SURFSS = 0x47,
  SIMRAD96_RAWRT = 0x46,
  SIMRAD96_SINGLEBEAM = 0x45,
  SIMRAD96_DEPTH = 0x44,
  SIMRAD96_DEPTHPREA = 0x44,
  SIMRAD96_CLOCK = 0x43,
  SIMRAD96_BIST = 0x42,
  SIMRAD96_ATTITUDE = 0x41,
  SIMRAD96_ATTITUDEPREA = 0x41,
  SIMRAD96_EXTRA_PARAMS = 0x33,
  SIMRAD96_DATA_OUT_ON = 0x32,
  SIMRAD96_PU_STATUSJ = 0x31,
  SIMRAD96_PU_STATUSH = 0x31,
  SIMRAD96_DATA_OUT_OFF = 0x31,
  SIMRAD96_PU_BROADCAST = 0x30
} simrad96_id;


//BUFFERS

typedef struct header_def {
  uint32_t n_bytes; //Number of bytes in datagram
  uint8_t start; //Start identifier = STX (Always 02h)
  uint8_t datagram; //Type of datagram
  uint16_t model; //EM model number (Example: EM 3000 = 3000)
  uint32_t date; //Date = year*10000 + month*100 + day (Example: Feb 26, 1995 = 19950226)
  uint32_t time; //Time since midnight in milliseconds (Example: 08 : 12 : 51.234 = 29570234)
}header_t;

typedef struct footer_def {
  uint8_t end; //End identifier = ETX (Always 03h)
  uint16_t check_sum; //Check sum of data between STX and ETX
} footer_t;

typedef struct position_datagram_def { //Type of datagram = P(osition data) (Always 050h)
  uint16_t counter; //Position counter (sequential counter)
  uint16_t serial_number; //System serial number
  int32_t latitude; //Latitude in decimal degrees*20000000 (negative if southern hemisphere) (
      Example: 32°34' S = -651333333)
  int32_t longitude; //Longitude in decimal degrees*10000000 (negative if western hemisphere) (
      Example: 110.25° E = 1102500000)
  uint16_t fix_quality; //Measure of position fix quality in cm
  uint16_t vessel_speed; //Speed of vessel over ground in cm/s
  uint16_t vessel_course; //Course of vessel over ground in 0.01°
  uint16_t vessel_heading; //Heading of vessel in 0.01°
  uint8_t position_system; //Position system descriptor
  uint8_t number_bytes; //Number of bytes in input datagram
} position_datagram_t;

typedef struct depth_datagram_beams_def {
  int16_t z; //Depth (z) from transmit transducer (unsigned for EM 120 and EM 300)
  uint16_t uz; //Depth (z) from transmit transducer (unsigned for EM 120 and EM 300)
  int16_t y; //Acrosstrack distance (y)
  int16_t x; //Alongtrack distance (x)
  int16_t depression_angle; //Beam depression angle in 0.01°
  uint16_t azimuth_angle; //Beam azimuth angle in 0.01°
```

```
  uint16_t range; // Range (one - way travel time)
  uint8_t quality_factor; //Quality factor
  uint8_t detection_window; // Length of detection window (samples/4)
  int8_t bs; // Reflectivity (BS) in 0.5 dB resolution) (Example: -20 dB = 216)
  uint8_t beam_number; //Beam number
} depth_datagram_beams_t;

typedef struct depth_datagram_def { //Type of datagram = D(epth data) (Always 44h)
  uint16_t ping_counter; //Ping counter (sequential counter)
  uint16_t serial_number; //System serial number
  uint16_t heading; //Heading of vessel in 0.01°
  uint16_t sound_speed; //Sound speed at transducer in dm/s
  uint16_t tx_transducer_depth; //Transmit transducer depth re water level at time of ping in cm
  uint8_t maximum_beams; //Maximum number of beams possible
  uint8_t valid_beams; //Number of valid beams = N
  uint8_t z_resolution; //z resolution in cm
  uint8_t x_y_resolution; //x and y resolution in cm
  uint16_t sampling_frequency; //x and y resolution in cm or Depth difference between sonar heads
      in the EM 3000D (int16_t)
  depth_datagram_beams_t depth_beam; //Repeat cycle - N entries of :
  int8_t transducer_depth; // Transducer depth offset multiplier
} depth_datagram_t;

typedef struct xyz_datagram_beams_def {
  float z; //Depth (z) from transmit transducer in m
  float y; //Acrosstrack distance (y) in m
  float x; //Alongtrack distance (x) in m
  uint16_t window_lenght; //Detection window length in samples
  uint8_t quality_factor; //Quality factor
  int8_t angle_adjustment; //Beam incidence angle adjustment (IBA) in 0.1 deg
  uint8_t detection_info; //Detection information
  int8_t cleaning; //Real time cleaning information
  int16_t bs; //Reflectivity (BS) in 0.1 dB resolution (Example: -20.1 dB = FF37h = 65335)
} xyz_datagram_beams_t;

typedef struct xyz_datagram_def { //Type of datagram = X (58h, 88d)
  uint16_t ping_counter; //Ping counter (sequential counter)
  uint16_t serial_number; //System serial number
  uint16_t heading; //Heading of vessel (at TX time) in 0.01°
  uint16_t sound_speed; //Sound speed at transducer in dm/s
  float tx_transducer_depth; //Transmit transducer depth in m re water level at time of ping
  uint16_t number_beams; //Number of beams in datagram = N
  uint16_t valid_detections; //Number of valid detections
  float sampling_frequency; //Sampling frequency in Hz
  uint8_t scanning_info; //Scanning info.
  uint8_t spare[3]; //Spare
  xyz_datagram_beams_t xyz_beam; //Repeat cycle - N entries of :
  uint8_t spare_1; //Spare (always 0)
} xyz_datagram_t;


//ESTRUTURA DE DADOS PRINCIPAL
typedef struct stat_cell_def { //Estrutura da célula estatística da grade
  double median;
  double quartile_1;
  double quartile_3;
  double iqr;
} stat_cell_t;

typedef struct cell_def { //Estrutura da célula da grade
  double *z;
  double *bs;
  double *angle;
```

```
  int  cell_size;
  int  cell_buf;
  stat_cell_t z_stat_cell;
  stat_cell_t bs_stat_cell;
  stat_cell_t angle_stat_cell;
} cell_t;

typedef struct grid_def { //Estrutura da grade
  cell_t **cell;
  double min_x;
  double max_x;
  double min_y;
  double max_y;
  double dx;
  double dy;
  double mid_along;
  int n_lins;
  int n_cols;
  double size_x;
  double size_y;
} grid_t;

typedef struct line_list_def { //Estrutura de listagem de arquivos de entrada
  FILE *directory;
  FILE *position_file;
  FILE *interpolated_backscatter_file;
  FILE *grid_file;
  char ** list;
  int n_lines;
  int n_buf_lines;
} line_list_t;

typedef struct sonar_beam_def { //Estrutura dos feixes
  double x;
  double y;
  double z;
  double bs;
  double angle;
  double range; //1-way
  double lat;
  double lon;
  double utmx;
  double utmy;
  double length;
  double azimuth;
} sonar_beam_t;

typedef struct sonar_ping_def { //Estrutura dos pings
  int ping_number;
  double epoch;
  double lon;
  double lat;
  double utmx;
  double utmy;
  double heading;
  double sound_speed;
  double tx_transducer_depth;
  int n_beams;
  int valid_detections;
  double sampling_frequency;
  sonar_beam_t *sonar_beam;
} sonar_ping_t;

typedef struct nav_fix_def { //Estrutura da navegação
  double epoch;
```

```
  double lon;
  double lat;
  double utmx;
  double utmy;
} nav_fix_t;

typedef struct line_def { //Estrutura dos dados da linha de navegação
  FILE *line_in;
  header_t header;
  uint8_t byteswap;
  int model;
  int n_fix;
  int n_buf_nav;
  int n_pings;
  int n_buf_pings;
  sonar_ping_t *pings;
  nav_fix_t *navigation;
} line_t;

typedef struct survey_def { //Estrutura principal onde vão todos os dados relevantes
  line_list_t line_list; //Lista de linhas de entrada
  line_t *line_data; //Dados das linhas de entrada
  grid_t grid; //Grade com os dados estatísticamente pré-processados
  int current_line;
  int utm_zone;
  int utm_central_meridian;
} survey_t;
```

Listagem B.2 – *main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "structs.h"
#include "io.h"
#include "read_datagrams.h"
#include "interpretation.h"
#include "interpolation.h"
#include "byteswap.h"
#include "angle.h"
#include "grid.h"


int main()
{
  survey_t survey;

  open_inputs(&survey); //Abertura dos arquivos

  //Leitura dos fix de navegação e de pings
  interpretation(&survey); //Leitura dos datagramas em cada linha de navegação

  //Pré-processamento dos dados lidos
  interpol(&survey); //Interpolação dos dados de batimetria e backscatter com os dados de posição

  create_grid(&survey); //Criação da grade

  output_files(&survey); //Escrita em arquivos de saída

  return 0;
}
```

Listagem B.3 – *io.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "structs.h"


void open_inputs(survey_t *survey_p) {
  char buffer[100];
  survey_p->line_list.directory = _popen("cd Inputs && dir *.all /b", "r");
  survey_p->line_list.n_lines = 0;
  survey_p->line_list.n_buf_lines = 1;
  survey_p->line_list.list = calloc(survey_p->line_list.n_buf_lines, sizeof(char *));
  if (survey_p->line_list.list == NULL) {
    printf("linha. list  sem memoria alocada\n"); //Mensagem de erro, sem memória
  }
  else {
    for (int linha_nav = 0; linha_nav < survey_p->line_list.n_buf_lines; linha_nav++) {
      survey_p->line_list.list[linha_nav] = calloc(100, sizeof(char));
    }
  }


  while (fgets(buffer, 100, survey_p->line_list.directory) != NULL) {
    buffer[strcspn(buffer, "\n")] = 0; //Remove quebra de linha se existir
    if (survey_p->line_list.n_lines >= survey_p->line_list.n_buf_lines) {
      char **line_list_p;
      survey_p->line_list.n_buf_lines *= 2;
      line_list_p = realloc(survey_p->line_list.list, survey_p->line_list.n_buf_lines * sizeof(char
          *));
      if (line_list_p == NULL) {
        printf("temp.list  sem memoria alocada\n"); //Mensagem de erro, sem memória
      }
      for (int line_nav = survey_p->line_list.n_buf_lines / 2; line_nav < survey_p->line_list.
          n_buf_lines; line_nav++) {
        line_list_p[line_nav] = calloc(100, sizeof(char));
        if (line_list_p[line_nav] == NULL) {
          printf("temp.list[%d] sem memoria alocada!\n", line_nav);
        }
      }
      survey_p->line_list.list = line_list_p;
    }

    strcpy(survey_p->line_list.list[survey_p->line_list.n_lines], buffer);

    survey_p->line_list.n_lines++;
  }

  for (int linha = 0; linha < survey_p->line_list.n_lines; linha++) {
    strcpy(buffer, survey_p->line_list.list[linha]);
    printf("%s\n", buffer);
  }

  survey_p->line_data = calloc(survey_p->line_list.n_lines, sizeof(line_t));

  _pclose(survey_p->line_list.directory);
}

void output_files(survey_t *survey_p) {
```

```
char buffer[100];
for (int line = 0; line < survey_p->line_list.n_lines; line++) {
  line_t *line_p = &survey_p->line_data[line];
  // Escreve navegação em arquivo de saída
  strcpy(buffer, OUT_FDR); //Pasta do arquivo de saída
  strcat(buffer, survey_p->line_list.list[line]); //Adiciona nome do arquivo de entrada no
      arquivo de saída
  buffer[strcspn(buffer, ".")] = 0; //Remove a extensão .all
  strcat(buffer, ".nav"); //Adiciona extensão .nav no arquivo de navegação
  survey_p->line_list.position_file = fopen(buffer, "w");
  fprintf(survey_p->line_list.position_file, "epoch\tX\tY\tLON\tLAT\n");
  for (int fix = 0; fix < line_p->n_fix; fix++) {
    fprintf(survey_p->line_list.position_file, "%.3lf\t%.3lf\t%.3lf\t%.3lf\t%.3lf\n", line_p->
        navigation[fix].epoch,
      line_p->navigation[fix].utmx, line_p->navigation[fix].utmy, line_p->navigation[fix].lon,
          line_p->navigation[fix].lat);
  }
  fclose(survey_p->line_list.position_file);

  //Escreve backscatter interpolado com latitude e longitude
  buffer[strcspn(buffer, ".")] = 0; //Remove a extensão anterior
  strcat(buffer, ".bs"); //Adiciona extensão .bs no arquivo de interpolação
  survey_p->line_list.interpolated_backscatter_file = fopen(buffer, "w");
  fprintf(survey_p->line_list.interpolated_backscatter_file, "X\tY\tZ\tBS\n");
  for (int ping = 0; ping < survey_p->line_data[line].n_pings; ping++) {
    for (int beam = 0; beam < line_p->pings[ping].n_beams; beam++) {
      fprintf(survey_p->line_list.interpolated_backscatter_file, "%.3lf\t%.3lf\t%.3lf\t%.1lf\n",
        line_p->pings[ping].sonar_beam[beam].utmx, line_p->pings[ping].sonar_beam[beam].
            utmy,
        line_p->pings[ping].sonar_beam[beam].z, line_p->pings[ping].sonar_beam[beam].corr_bs
            );
    }
  }

  fclose(survey_p->line_list.interpolated_backscatter_file);
}

//Escreve binning
strcpy(buffer, OUT_FDR); //Pasta do arquivo de saída
strcat(buffer, "grid.txt");
survey_p->line_list.grid_file = fopen(buffer, "w");
if (survey_p->utm_zone > 0) {
    int utm_zone = survey_p->utm_zone;
    fprintf(survey_p->line_list.grid_file, "UTM Zone: %dN\n", utm_zone);
}
else {
    int utm_zone = survey_p->utm_zone * -1;
    fprintf(survey_p->line_list.grid_file, "UTM Zone: %dS\n", utm_zone);
}
fprintf(survey_p->line_list.grid_file, "X\tY\tZ\tBS\n");
for (int lin = 0; lin < survey_p->grid.n_lins; lin++) {
  for (int col = 0; col < survey_p->grid.n_cols; col++) {
    if (survey_p->grid.cell[col][lin].bs_stat_cell.median != INV_BIN) {
      double x = ((double)col / (double)grid_p->n_cols)*grid_p->dx + grid_p->min_x;
      double y = ((double)lin / (double)grid_p->n_lins)*grid_p->dy + grid_p->min_y;
      fprintf(survey_p->line_list.grid_file, "%.3lf\t%.3lf\t%.3lf\t%.3lf\n", x, y,
        survey_p->grid.cell[col][lin].z_stat_cell.median, survey_p->grid.cell[col][lin].
            bs_stat_cell.median);
    }

  }
}
fclose(survey_p->line_list.grid_file);
```

```
}
```

Listagem B.4 – *interpretation.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "structs.h"
#include "byteswap.h"
#include "read_datagrams.h"


static void read_header(line_t *line_p) {
  fread(&line_p->header, sizeof(header_t), 1, line_p->line_in);

  if (line_p->byteswap) {
    line_p->header.n_bytes = byteswap4u(line_p->header.n_bytes);
    line_p->header.model = byteswap2u(line_p->header.model);
    line_p->header.date = byteswap4u(line_p->header.date);
    line_p->header.time = byteswap4u(line_p->header.time);
  }
}

static void dtg_interpretation(survey_t *survey_p) {
  line_t *line_p = &survey_p->line_data[survey_p->current_line];
  unsigned int buffer_size;
  unsigned char *buffer;
  buffer_size = line_p->header.n_bytes - sizeof(header_t) + 4; //+ 4 pois não écontabilizado o
      tamanho do campo número de bytes
  buffer = malloc(buffer_size);

  if (buffer == NULL) {
    printf("buffer de interpretacao sem memoria alocada\n"); //Mensagem de erro, sem memória
  }

  if (line_p->header.datagram == SIMRAD96_POSITION) {
    read_position_datagram(line_p);
  }
  else if (line_p->header.datagram == SIMRAD96_DEPTH) {
    read_depth_datagram(line_p);
  }
  else if (line_p->header.datagram == SIMRAD96_XYZ) {
    read_xyz_datagram(line_p);
  }
  else {
    fread(buffer, buffer_size, 1, line_p->line_in);
  }

  free(buffer);
}

void interpretation(survey_t *survey_p) {
  for (survey_p->current_line = 0; survey_p->current_line < survey_p->line_list.n_lines;
      survey_p->current_line++) {
    line_t *line_p = &survey_p->line_data[survey_p->current_line];

    //Sequência de abertura dos arquivos de entrada
    char buffer[100];
    strcpy(buffer, IN_FDR);
    strcat(buffer, survey_p->line_list.list[survey_p->current_line]);
```

```
    line_p->line_in = fopen(buffer, "rb");

    byteswapvalidation(line_p); //Verificação de se realizar byteswap

    if (line_p->model) { //Se modelo válido, realizar leitura dos datagramas
      while (!feof(line_p->line_in)) { //Leitura da header e interpretação
        read_header(line_p);
        dtg_interpretation(survey_p);
      }
    }
    else {
      printf("Erro de leitura na linha %s\n", buffer);
    }
    fclose(line_p->line_in);
  }
}
```

Listagem B.5 – *read_datagrams.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <math.h>

#include "structs.h"
#include "interpolation.h"
#include "byteswap.h"

void read_position_datagram(line_t *line_p) {
  position_datagram_t dtg;
  footer_t footer;
  unsigned char *position_input; //Position input datagram as received

  fread(&dtg.counter, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.serial_number, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.latitude, sizeof(uint32_t), 1, line_p->line_in);
  fread(&dtg.longitude, sizeof(uint32_t), 1, line_p->line_in);
  fread(&dtg.fix_quality, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.vessel_speed, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.vessel_course, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.vessel_heading, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.position_system, sizeof(uint8_t), 1, line_p->line_in);
  fread(&dtg.number_bytes, sizeof(uint8_t), 1, line_p->line_in);

  position_input = malloc(dtg.number_bytes + 1);
  fread(position_input, dtg.number_bytes, 1, line_p->line_in);
  fread(&footer.end, sizeof(uint8_t), 1, line_p->line_in);
  if (footer.end == 0) {
    fread(&footer.end, sizeof(uint8_t), 1, line_p->line_in);
  }
  fread(&footer.check_sum, sizeof(uint16_t), 1, line_p->line_in);

  free(position_input);

  if (line_p->n_fix == 0) { //Alocação inicial
    line_p->n_buf_nav = 512;
    line_p->navigation = calloc(line_p->n_buf_nav, sizeof(nav_fix_t));
    if (line_p->navigation == NULL) {
      printf("linha.navigation sem memoria alocada\n"); //Mensagem de erro, sem memória
    }
  }
```

```
  if (line_p->n_fix >= line_p->n_buf_nav) {
    nav_fix_t *temp_nav_p;
    line_p->n_buf_nav += 512;
    temp_nav_p = (nav_fix_t *)realloc(line_p->navigation, line_p->n_buf_nav*(sizeof(
        nav_fix_t)));
    if (temp_nav_p == NULL) {
      printf("temp.nav sem memoria alocada\n"); //Mensagem de erro, sem memória
    }
    line_p->navigation = temp_nav_p;
  }

  if (line_p->byteswap) {
    line_p->navigation[line_p->n_fix].epoch = read_time(line_p->header.time, line_p->header.
        date);
    line_p->navigation[line_p->n_fix].lat = byteswap4(dtg.latitude) / 20000000.0;
    line_p->navigation[line_p->n_fix].lon = byteswap4(dtg.longitude) / 10000000.0;
  }
  else {
    line_p->navigation[line_p->n_fix].epoch = read_time(line_p->header.time, line_p->header.
        date);
    line_p->navigation[line_p->n_fix].lat = dtg.latitude / 20000000.0;
    line_p->navigation[line_p->n_fix].lon = dtg.longitude / 10000000.0;
  }

  line_p->n_fix++;
}

void read_depth_datagram(line_t *line_p) {
  depth_datagram_t dtg;
  footer_t footer;

  fread(&dtg.ping_counter, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.serial_number, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.heading, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.sound_speed, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.tx_transducer_depth, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.maximum_beams, sizeof(uint8_t), 1, line_p->line_in);
  fread(&dtg.valid_beams, sizeof(uint8_t), 1, line_p->line_in);
  fread(&dtg.z_resolution, sizeof(uint8_t), 1, line_p->line_in);
  fread(&dtg.x_y_resolution, sizeof(uint8_t), 1, line_p->line_in);
  fread(&dtg.sampling_frequency, sizeof(uint16_t), 1, line_p->line_in);

  if (line_p->n_pings == 0) { //Alocação inicial
    line_p->n_buf_pings = 512;
    line_p->pings = calloc(line_p->n_buf_pings, sizeof(sonar_ping_t));
    if (line_p->pings == NULL) {
      printf("linha.pings sem memoria alocada\n"); //Mensagem de erro, sem memória
    }
  }

  if (line_p->n_pings >= line_p->n_buf_pings) {
    sonar_ping_t *sonar_ping_p;
    line_p->n_buf_pings += 512;
    sonar_ping_p = (sonar_ping_t *)realloc(line_p->pings, line_p->n_buf_pings*(sizeof(
        sonar_ping_t)));
    if (sonar_ping_p == NULL) {
      printf("temp.pings sem memoria alocada\n");//Mensagem de erro, sem memória
    }
    line_p->pings = sonar_ping_p;
  }

  if (line_p->byteswap) {
```

```
    line_p->pings[line_p->n_pings].epoch = read_time(line_p->header.time, line_p->header.
        date);
    line_p->pings[line_p->n_pings].heading = byteswap2u(dtg.heading) * 0.01; //Em graus
    line_p->pings[line_p->n_pings].sound_speed = byteswap2u(dtg.sound_speed) / 10.0; //Em m
        /s
    line_p->pings[line_p->n_pings].tx_transducer_depth = byteswap2u(dtg.tx_transducer_depth)
        ; //Em m
    line_p->pings[line_p->n_pings].n_beams = dtg.valid_beams;
    line_p->pings[line_p->n_pings].valid_detections = dtg.valid_beams;
    line_p->pings[line_p->n_pings].sampling_frequency = byteswap2u(dtg.sampling_frequency);
        //Em Hz
}
else
{
    line_p->pings[line_p->n_pings].epoch = read_time(line_p->header.time, line_p->header.
        date);
    line_p->pings[line_p->n_pings].heading = dtg.heading *0.01; //Em graus
    line_p->pings[line_p->n_pings].sound_speed = dtg.sound_speed / 10.0; //Em m/s
    line_p->pings[line_p->n_pings].tx_transducer_depth = dtg.tx_transducer_depth; //Em m
    line_p->pings[line_p->n_pings].n_beams = dtg.valid_beams;
    line_p->pings[line_p->n_pings].valid_detections = dtg.valid_beams;
    line_p->pings[line_p->n_pings].sampling_frequency = dtg.sampling_frequency; //Em Hz
}

line_p->pings[line_p->n_pings].ping_number = ping_counter;

line_p->pings[line_p->n_pings].sonar_beam = calloc(line_p->pings[line_p->n_pings].
    n_beams, sizeof(sonar_beam_t));

if (line_p->pings[line_p->n_pings].sonar_beam == NULL)
{
    printf("linha.pings.sonar_beam sem memoria alocada\n"); //Mensagem de erro, sem memória
}

for (int beam = 0; beam < line_p->pings[line_p->n_pings].n_beams; beam++) {
    if (line_p->model == EM120 || line_p->model == EM300) {
        fread(&dtg.depth_beam.z, sizeof(int16_t), 1, line_p->line_in);
    }
    else {
        fread(&dtg.depth_beam.uz, sizeof(uint16_t), 1, line_p->line_in);
    }

    fread(&dtg.depth_beam.y, sizeof(int16_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.x, sizeof(int16_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.depression_angle, sizeof(int16_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.azimuth_angle, sizeof(uint16_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.range, sizeof(uint16_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.quality_factor, sizeof(uint8_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.detection_window, sizeof(uint8_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.bs, sizeof(int8_t), 1, line_p->line_in);
    fread(&dtg.depth_beam.beam_number, sizeof(uint8_t), 1, line_p->line_in);

    if (line_p->byteswap) {
        if (line_p->model == EM120 || line_p->model == EM300) {
            line_p->pings[line_p->n_pings].sonar_beam[beam].z = byteswap2(dtg.depth_beam.z) *
                dtg.z_resolution * 0.01; //Em m
        }
        else {
            line_p->pings[line_p->n_pings].sonar_beam[beam].z = byteswap2u(dtg.depth_beam.uz) *
                dtg.z_resolution * 0.01; //Em m
        }
        line_p->pings[line_p->n_pings].sonar_beam[beam].y = byteswap2(dtg.depth_beam.y) * dtg.
            x_y_resolution * 0.01; //Em m
```

```
      line_p->pings[line_p->n_pings].sonar_beam[beam].x = byteswap2(dtg.depth_beam.x) * dtg.
          x_y_resolution * 0.01; //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].bs = dtg.depth_beam.bs * 0.5; //Em dB
      line_p->pings[line_p->n_pings].sonar_beam[beam].range = byteswap2u(dtg.depth_beam.
          range) / (byteswap2u(dtg.sampling_frequency) * 4.0);
    }
    else {
      if (line_p->model == EM120 || line_p->model == EM300) {
        line_p->pings[line_p->n_pings].sonar_beam[beam].z = dtg.depth_beam.z * dtg.
            z_resolution * 0.01; //Em m
      }
      else {
        line_p->pings[line_p->n_pings].sonar_beam[beam].z = dtg.depth_beam.uz * dtg.
            z_resolution * 0.01; //Em m
      }
      line_p->pings[line_p->n_pings].sonar_beam[beam].y = dtg.depth_beam.y * dtg.
          x_y_resolution * 0.01; //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].x = dtg.depth_beam.x * dtg.
          x_y_resolution * 0.01; //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].bs = dtg.depth_beam.bs * 0.5; //Em dB
      line_p->pings[line_p->n_pings].sonar_beam[beam].range = dtg.depth_beam.range / (dtg.
          sampling_frequency * 4.0);
    }

  }

  fread(&dtg.transducer_depth, sizeof(int8_t), 1, line_p->line_in);
  fread(&footer.end, sizeof(uint8_t), 1, line_p->line_in);
  fread(&footer.check_sum, sizeof(uint16_t), 1, line_p->line_in);

    line_p->n_pings++;
}

void read_xyz_datagram(line_t *line_p) {
  xyz_datagram_t dtg;
  footer_t footer;

  fread(&dtg.ping_counter, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.serial_number, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.heading, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.sound_speed, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.tx_transducer_depth, sizeof(float), 1, line_p->line_in);
  fread(&dtg.number_beams, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.valid_detections, sizeof(uint16_t), 1, line_p->line_in);
  fread(&dtg.sampling_frequency, sizeof(float), 1, line_p->line_in);
  fread(&dtg.scanning_info, sizeof(uint8_t), 1, line_p->line_in);
  fread(&dtg.spare, sizeof(uint8_t), 3, line_p->line_in);

  if (line_p->n_pings == 0) { //Alocação inicial
    line_p->n_buf_pings = 512;
    line_p->pings = calloc(line_p->n_buf_pings, sizeof(sonar_ping_t));
    if (line_p->pings == NULL) {
      printf("linha.pings sem memoria alocada\n"); //Mensagem de erro, sem memória
    }
  }

  if (line_p->n_pings >= line_p->n_buf_pings) {
    sonar_ping_t *sonar_ping_p;
    line_p->n_buf_pings += 512;
    sonar_ping_p = (sonar_ping_t *)realloc(line_p->pings, line_p->n_buf_pings*(sizeof(
        sonar_ping_t)));
    if (sonar_ping_p == NULL) {
      printf("temp.pings sem memoria alocada\n");//Mensagem de erro, sem memória
```

```
    }
    line_p->pings = sonar_ping_p;
  }

  if (line_p->byteswap) {
    line_p->pings[line_p->n_pings].epoch = read_time(line_p->header.time, line_p->header.
        date);
    line_p->pings[line_p->n_pings].heading = byteswap2u(dtg.heading) * 0.01; //Em graus
    line_p->pings[line_p->n_pings].sound_speed = byteswap2u(dtg.sound_speed) * 0.1; //Em m/s
    line_p->pings[line_p->n_pings].tx_transducer_depth = byteswapf(dtg.tx_transducer_depth);
        //Em m
    line_p->pings[line_p->n_pings].n_beams = byteswap2u(dtg.number_beams);
    line_p->pings[line_p->n_pings].valid_detections = byteswap2u(dtg.valid_detections);
    line_p->pings[line_p->n_pings].sampling_frequency = byteswapf(dtg.sampling_frequency); //
        Em Hz
  }
  else
  {
    line_p->pings[line_p->n_pings].epoch = read_time(line_p->header.time, line_p->header.
        date);
    line_p->pings[line_p->n_pings].heading = dtg.heading * 0.01; //Em graus
    line_p->pings[line_p->n_pings].sound_speed = dtg.sound_speed * 0.1; //Em m/s
    line_p->pings[line_p->n_pings].tx_transducer_depth = dtg.tx_transducer_depth; //Em m
    line_p->pings[line_p->n_pings].n_beams = dtg.number_beams;
    line_p->pings[line_p->n_pings].valid_detections = dtg.valid_detections;
    line_p->pings[line_p->n_pings].sampling_frequency = dtg.sampling_frequency; //Em Hz
  }


line_p->pings[line_p->n_pings].sonar_beam = calloc(line_p->pings[line_p->n_pings].
    n_beams, sizeof(sonar_beam_t));

  if (line_p->pings[line_p->n_pings].sonar_beam == NULL)
  {
    printf("linha.pings.sonar_beam sem memoria alocada\n"); //Mensagem de erro, sem memória
  }

  for (int beam = 0; beam < line_p->pings[line_p->n_pings].n_beams; beam++) {
    fread(&dtg.xyz_beam.z, sizeof(float), 1, line_p->line_in);
    fread(&dtg.xyz_beam.y, sizeof(float), 1, line_p->line_in);
    fread(&dtg.xyz_beam.x, sizeof(float), 1, line_p->line_in);
    fread(&dtg.xyz_beam.window_lenght, sizeof(uint16_t), 1, line_p->line_in);
    fread(&dtg.xyz_beam.quality_factor, sizeof(uint8_t), 1, line_p->line_in);
    fread(&dtg.xyz_beam.angle_adjustment, sizeof(int8_t), 1, line_p->line_in);
    fread(&dtg.xyz_beam.detection_info, sizeof(uint8_t), 1, line_p->line_in);
    fread(&dtg.xyz_beam.cleaning, sizeof(int8_t), 1, line_p->line_in);
    fread(&dtg.xyz_beam.bs, sizeof(int16_t), 1, line_p->line_in);

    if (line_p->byteswap) {
      line_p->pings[line_p->n_pings].sonar_beam[beam].z = byteswapf(dtg.xyz_beam.z); //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].y = byteswapf(dtg.xyz_beam.y); //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].x = byteswapf(dtg.xyz_beam.x); //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].bs = byteswap2(dtg.xyz_beam.bs) * 0.1;
          //Em dB
    }
    else {
      line_p->pings[line_p->n_pings].sonar_beam[beam].z = dtg.xyz_beam.z; //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].y = dtg.xyz_beam.y; //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].x = dtg.xyz_beam.x; //Em m
      line_p->pings[line_p->n_pings].sonar_beam[beam].bs = dtg.xyz_beam.bs * 0.1; //Em dB
    }

  }
```

```
  fread(&dtg.spare_1, sizeof(uint8_t), 1, line_p->line_in);
  fread(&footer.end, sizeof(uint8_t), 1, line_p->line_in);
  fread(&footer.check_sum, sizeof(uint16_t), 1, line_p->line_in);

  line_p->n_pings++;
}
```

Listagem B.6 – *interpolation.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <float.h>
#include <string.h>

#include "structs.h"
#include "angle.h"


static int comp_epoch_ping(const void *a, const void *b)
{
  sonar_ping_t *a1 = (sonar_ping_t *)a;
  sonar_ping_t *a2 = (sonar_ping_t *)b;
  if ((a1->epoch)<(a2->epoch))return -1;
  else if ((a1->epoch)>(a2->epoch))return 1;
  else return 0;
}

static int comp_epoch_nav(const void *a, const void *b)
{
  nav_fix_t *a1 = (nav_fix_t *)a;
  nav_fix_t *a2 = (nav_fix_t *)b;
  if ((a1->epoch)<(a2->epoch))return -1;
  else if ((a1->epoch)>(a2->epoch))return 1;
  else return 0;
}

static int findNearestNeighbourIndex(double value, double *x, int len)
{
  double dist;
  int idx;
  int i;

  idx = -1;
  dist = DBL_MAX;
  if (value < x[0]) {
    return 0;
  }
  for (i = 0; i < len; i++) {
    double newDist = value - x[i];
    if (newDist > 0 && newDist < dist) {
      dist = newDist;
      idx = i;
    }
  }

  return idx;
}

static int linear(double *xi, double *yi, int nin, double *xo, double *yo, int nout)
```

```
{
  double dx, dy, *slope, *intercept;
  int i, indiceEnVector;

  slope = (double *)calloc(nin, sizeof(double));
  intercept = (double *)calloc(nin, sizeof(double));

  for (i = 0; i < nin; i++) {
    if (i<nin - 1) {
      dx = xi[i + 1] - xi[i];
      if (!dx) {
        return(-2);
      }
      dy = yi[i + 1] - yi[i];
      slope[i] = dy / dx;
      intercept[i] = yi[i] - xi[i] * slope[i];
    }
    else {
      slope[i] = slope[i - 1];
      intercept[i] = intercept[i - 1];
    }
  }

  for (i = 0; i < nout; i++) {
    indiceEnVector = findNearestNeighbourIndex(xo[i], xi, nin);
    if (indiceEnVector != -1)
      yo[i] = slope[indiceEnVector] * xo[i] + intercept[indiceEnVector];
    else
      yo[i] = DBL_MAX;
  }
  free(slope);
  free(intercept);
  return(0);
}

static void ConstructArc(double *lat_ptr, double *lon_ptr, double length, double azimuth)
{
  // CGeoReference::ConstructArc solves the direct problem in geodesy, that
  // is: given the coordinates of point p1, plus the azimuth and distance
  // to a second point -- compute the coordinates of the second point
  // the solution is iterated untill the change in sigma is less than DEL

  // Note the ugly coding and ungainly variable names -- due to direct
  // translation from Fortran -- You should've seen the original!!

  // REFERENCE T. VINCENTY - SURVEY REVIEW, APRIL 1975
  // Original Fortran source GLOB by Inge Nesbo, Jan 1976.

#define DEL 1.0e-12

  double sinAzim, cosAzim, sinLat, cosLat, diag, sinU1, cosU1, ac, bc;
  double sinSig1, cosSig1, sig, sig1, sAlf, uSq, dSig, dDSig, dSigM, sig0;
  double sinSig, cosSig, cosDSig, sinDLat, cosDLat, cSqAlf, c, ddL;
  short ik, stop;
  double To_z, To_lat, To_lon, From_z, From_lat, From_lon;
  double m_dA, m_dF, m_dB, m_dSqE, m_dE, m_dSqE2, m_dE2, bOnA, eSq2;

  m_dA = 6378137.;
  m_dF = 1. / 298.257223563;
  m_dB = m_dA * (1. - m_dF);
  //  m_dN = ( m_dA - m_dB) / ( m_dA + m_dB);
  m_dSqE = (2 * m_dF - m_dF * m_dF);
  m_dE = sqrt(m_dSqE);
  m_dSqE2 = (m_dA * m_dA - m_dB * m_dB) / (m_dB * m_dB);
```

```
m_dE2 = sqrt(m_dSqE2);
//  m_strSpheroid = "World Geodetic System 1984";
bOnA = m_dB / m_dA;
eSq2 = m_dE2 * m_dE2;

To_z = From_z = 0.0;
From_lat = *lat_ptr;
From_lon = *lon_ptr;
sinAzim = sin(azimuth);
cosAzim = cos(azimuth);

sinLat = sin(From_lat) * bOnA;
cosLat = cos(From_lat);

diag = sqrt(sinLat * sinLat + cosLat * cosLat);

sinU1 = sinLat / diag;
cosU1 = cosLat / diag;

sinSig1 = sinU1;
cosSig1 = cosU1 * cosAzim;
sig1 = atan2(sinSig1, cosSig1);
sAlf = cosU1 * sinAzim;

uSq = (1. - sAlf * sAlf) * eSq2;
ac = -768. + uSq * (320. - 175. * uSq);
ac = 1. + uSq * (4096. + uSq * ac) / 16384.;
bc = -128. + uSq * (74. - 47. * uSq);
bc = uSq * (256. + uSq * bc) / 1024.;

dDSig = 0.;
sig0 = length / m_dB / ac;
sig = sig0;

// Start iteration

// NOTE: the strange stop construct is used as I did not want to change
// the code too fundamentaly from the fortran code -- the fortran code
// used a goto statement to jump out of the !middle! of the 'while' loop

ik = TRUE;
stop = FALSE;

while (!stop) {
  if (!ik) stop = TRUE;
  dSig = dDSig;
  dSigM = 2. * sig1 + sig;
  sinSig = sin(sig);
  cosSig = cos(sig);
  cosDSig = cos(dSigM);
  if (ik) {
    dDSig = bc * cosDSig * (-3. + 4. * sinSig * sinSig) * (-3. + 4. * cosDSig * cosDSig) / 6.;
    dDSig = bc * sinSig * (cosDSig + bc * (cosSig * (-1. + 2. * cosDSig * cosDSig) - dDSig) / 4.);
    sig = sig0 + dDSig;
    if (fabs(dDSig - dSig) <= DEL) ik = FALSE;
  }
}

// compute lattitude
sinLat = sinU1 * cosSig + cosU1 * sinSig * cosAzim;
cosLat = sinU1 * sinSig - cosU1 * cosSig * cosAzim;
cosLat = (1. - m_dF) * sqrt(sAlf * sAlf + cosLat * cosLat);
To_lat = atan2(sinLat, cosLat);

// Compute longitude
```

```
  sinDLat = sinSig * sinAzim;
  cosDLat = cosU1 * cosSig - sinU1 * sinSig * cosAzim;
  cSqAlf = 1. - sAlf * sAlf;
  c = cSqAlf * (4. + m_dF * (4. - 3. * cSqAlf)) * m_dF / 16.;

  ddL = cosDSig + c * cosSig * (-1. + 2. * cosDSig * cosDSig);
  ddL = (1. - c) * m_dF * sAlf * (sig + c * sinSig * ddL);

  To_lon = From_lon + atan2(sinDLat, cosDLat) - ddL;

  *lat_ptr = To_lat;
  *lon_ptr = To_lon;
}


#define WGS84 1
#define WGS84_ECCEN 0.0818191908417579
#define WGS84_RADIUS 6378137                /* units are meters */

static void geo2utm(double phi, double lam, double clam, double *x, double *y)
{
  double k0 = 0.9996; /* central scale factor */
  double x0 = 500000.0;      /* x0 to be added in standard UTM */
  double y0 = 10000000.0;    /* y0 to be added in standard UTM for southern hemisphere */
  double e2, a;
  double e4;
  double e6;
  double epr2;
  double cosphi, sinphi, tanphi;
  double n, t, t2, c;
  double b, b2, b3, b4, b5, b6;
  double m, m1, m2, m3, m4;
  double dphi, dlam, dclam;

  e2 = WGS84_ECCEN * WGS84_ECCEN;
  a = WGS84_RADIUS;

  e4 = e2 * e2;
  e6 = e2 * e2 * e2;
  epr2 = e2 / (1.0 - e2);

  dphi = deg2rad(phi);
  dlam = deg2rad(lam);
  dclam = deg2rad(clam);
  cosphi = cos(dphi);
  sinphi = sin(dphi);
  tanphi = tan(dphi);

  n = a / sqrt(1.0 - e2 * sinphi * sinphi);
  t = tanphi * tanphi;
  t2 = t * t;
  c = epr2 * cosphi * cosphi;
  b = (dlam - dclam) * cosphi;      /* b is A in Snyder's formulas */
  b2 = b * b;
  b3 = b * b * b;
  b4 = b * b * b * b;
  b5 = b * b * b * b * b;
  b6 = b * b * b * b * b * b;
  m1 = 1.0 - e2 / 4.0 - 3.0*e4 / 64.0 - 5.0*e6 / 256.0;
  m2 = 3.0*e2 / 8.0 + 3.0*e4 / 32.0 + 45.0*e6 / 1024.0;
  m3 = 15.0*e4 / 256.0 + 45.0*e6 / 1024.0;
  m4 = 35.0*e6 / 3072.0;
  m = a * (m1*dphi - m2 * sin(2.0*dphi) + m3 * sin(4.0*dphi) - m4 * sin(6.0*dphi));
  *x = k0 * n * (b + (1.0 - t + c)*b3 / 6.0 + (5.0 - 18.0*t + t2 + 72.0*c - 58.0*epr2)*b5 / 120.0);
  *x = *x + x0;
```

```
    *y = k0 * (m + n * tanphi*(b2 / 2.0 + (5.0 - t + 9.0*c + 4.0*c*c)*b4 / 24.0 + (61.0 - 58.0*t + t2
          + 600.0*c - 330.0*epr2)*b6 / 720.0));
    // if (dphi < 0.0) *y = *y + y0;
    *y = *y + y0; //Luciano maio 2016
    if (phi > 0) {
      *y = *y - y0;
    }
}

static int locate_beam(line_t *line_p, int ping, int beam) {
  double dx, dy, trans_lon, trans_lat, azimuth, length;

  trans_lat = deg2rad(line_p->pings[ping].lat);
  trans_lon = deg2rad(line_p->pings[ping].lon);

  dx = line_p->pings[ping].sonar_beam[beam].x;
  dy = line_p->pings[ping].sonar_beam[beam].y;

  int badNumber = _isnan(dx) || _isnan(dy);

  // If transducer has a valid navigation, calculate beam position using Semme's software
  if (trans_lat != 0.0 && trans_lon != 0.0 && !badNumber) {
    length = sqrt(dx * dx + dy * dy);
    if (dy == 0) {
      azimuth = line_p->pings[ping].heading;
    }
    else {
      // positive dx implies negative angle, positive dy adds 90 degrees to the heading
      if (dy > 0.0)
        azimuth = (double)(line_p->pings[ping].heading + 90.0 - rad2deg(atan(dx / fabs(dy))));
      else
        azimuth = (double)(line_p->pings[ping].heading - 90.0 + rad2deg(atan(dx / fabs(dy))));
    }
    if (azimuth > 360.0) azimuth -= 360;
    azimuth = deg2rad(azimuth);

    if (length != 0.0) {
      ConstructArc(&trans_lat, &trans_lon, length, azimuth);
    }
    line_p->pings[ping].sonar_beam[beam].lat = rad2deg(trans_lat);
    line_p->pings[ping].sonar_beam[beam].lon = rad2deg(trans_lon);

    line_p->pings[ping].sonar_beam[beam].length = length;
    if (dy < 0.0) {
      line_p->pings[ping].sonar_beam[beam].length *= (-1);
    }
    line_p->pings[ping].sonar_beam[beam].azimuth = azimuth;
    return(TRUE);
  }
  else {
    //showErrorMessage("bad positioning"); Yuri: done by the calling routine
    line_p->pings[ping].sonar_beam[beam].lat = 0.0;
    line_p->pings[ping].sonar_beam[beam].lon = 0.0;
    line_p->pings[ping].sonar_beam[beam].length = 0.0;
    line_p->pings[ping].sonar_beam[beam].azimuth = line_p->pings[ping].heading;
  }
  return(FALSE);
}

static void remove_duplicates(line_t *line_p) {
  int removed_fix = 0, removed_pings = 0, temp_fix = 1, temp_ping = 1;
  nav_fix_t *new_nav, *temp_nav;
  sonar_ping_t *new_pings, *temp_pings;
```

```
  temp_nav = calloc(line_p->n_fix, sizeof(nav_fix_t));
  temp_pings = calloc(line_p->n_pings, sizeof(sonar_ping_t));

  temp_nav[0] = line_p->navigation[0];

  for (int fix = 1; fix < line_p->n_fix; fix++) {
    if (line_p->navigation[fix].epoch != line_p->navigation[fix - 1].epoch) {
      temp_nav[temp_fix] = line_p->navigation[fix];
      temp_fix++;
    }
    else {
      removed_fix++;
    }
  }

  if (removed_fix) {
    new_nav = calloc(line_p->n_fix - removed_fix, sizeof(nav_fix_t));
    for (int fix = 0; fix < line_p->n_fix - removed_fix; fix++) {
      new_nav[fix] = temp_nav[fix];
    }
    line_p->navigation = new_nav;
    line_p->n_fix -= removed_fix;
  }


  temp_pings[0] = line_p->pings[0];

  for (int ping = 1; ping < line_p->n_pings; ping++) {
    if (line_p->navigation[ping].epoch != line_p->navigation[ping - 1].epoch) {
      temp_nav[temp_ping] = line_p->navigation[ping];
      temp_ping++;
    }
    else {
      removed_pings++;
    }
  }

  if (removed_pings) {
    new_pings = calloc(line_p->n_pings - removed_pings, sizeof(sonar_ping_t));
    for (int ping = 0; ping < line_p->n_pings - removed_pings; ping++) {
      new_pings[ping] = temp_pings[ping];
    }
    line_p->pings = new_pings;
    line_p->n_pings -= removed_pings;
  }
}

static void find_central_meridian(survey_t *survey_p) {
  double min_lon, max_lon, mid_lon;

  min_lon = survey_p->line_data[0].navigation[0].lon;
  max_lon = survey_p->line_data[0].navigation[0].lon;

  for (int line = 0; line < survey_p->line_list.n_lines; line++) {
    for (int fix = 0; fix < survey_p->line_data[line].n_fix; fix++) {
      if (min_lon > survey_p->line_data[line].navigation[fix].lon) {
        min_lon = survey_p->line_data[line].navigation[fix].lon;
      }
      if (max_lon < survey_p->line_data[line].navigation[fix].lon) {
        max_lon = survey_p->line_data[line].navigation[fix].lon;
      }
    }
  }
```

```
  mid_lon = (max_lon + min_lon) / 2.0;

  if (mid_lon > 0) {
    survey_p->utm_central_meridian = (int)(mid_lon / 6) * 6 + 3;
  }
  else {
    survey_p->utm_central_meridian = (int)(mid_lon / 6) * 6 - 3;
  }
  survey_p->utm_zone = ((survey_p->utm_central_meridian + 177) / 6) + 1;
  if (survey_p->line_data[0].navigation[0].lat < 0) { //Hemisfério sul
      survey_p->utm_zone *= -1
  }
}

void interpol(survey_t *survey_p) {
  double *tn, *lat, *lon; //Buffers de navegação
  double *tp, *i_lat, *i_lon; //Buffers de backscatter e interpolação
  int error;

  find_central_meridian(survey_p); //Encontra o meridiano central para conversão de WGS84 para
      UTM

  for (int line = 0; line < survey_p->line_list.n_lines; line++) {
    line_t *line_p = &survey_p->line_data[line];

    qsort(line_p->navigation, line_p->n_fix, sizeof(nav_fix_t), comp_epoch_nav);
    qsort(line_p->pings, line_p->n_pings, sizeof(sonar_ping_t), comp_epoch_ping);

    printf("%d\n", line);
    remove_duplicates(line_p);

    tn = calloc(line_p->n_fix, sizeof(double)); //Tempo na navegação
    lat = calloc(line_p->n_fix, sizeof(double)); //Latitude
    lon = calloc(line_p->n_fix, sizeof(double)); //Longitude

    for (int fix = 0; fix < line_p->n_fix; fix++) {
      tn[fix] = line_p->navigation[fix].epoch;
      lat[fix] = line_p->navigation[fix].lat;
      lon[fix] = line_p->navigation[fix].lon;
      geo2utm(lat[fix], lon[fix], survey_p->utm_central_meridian, &line_p->navigation[fix].
          utmx, &line_p->navigation[fix].utmy);
    }

    tp = calloc(line_p->n_pings, sizeof(double)); //Tempo nos pings
    i_lat = calloc(line_p->n_pings, sizeof(double)); //Latitude interpolada
    i_lon = calloc(line_p->n_pings, sizeof(double)); //Longitude interpolada

    for (int ping = 0; ping < line_p->n_pings; ping++) {
      tp[ping] = line_p->pings[ping].epoch;
    }

    error = linear(tn, lat, line_p->n_fix, tp, i_lat, line_p->n_pings);

    if (error) {
      printf("Erro %d na interpolacao de latitude!\n", error);
    }

    error = linear(tn, lon, line_p->n_fix, tp, i_lon, line_p->n_pings);

    if (error) {
      printf("Erro %d na interpolacao de longitude!\n", error);
    }

    for (int ping = 0; ping < line_p->n_pings; ping++) {
```

```
        line_p->pings[ping].lat = i_lat[ping]; //Latitude do ping
        line_p->pings[ping].lon = i_lon[ping]; //Longitude do ping
        geo2utm(i_lat[ping], i_lon[ping], survey_p->utm_central_meridian, &line_p->pings[ping].
            utmx, &line_p->pings[ping].utmy);
          for (int beam = 0; beam < line_p->pings[ping].n_beams; beam++) {
            locate_beam(line_p, ping, beam);
            geo2utm(line_p->pings[ping].sonar_beam[beam].lat, line_p->pings[ping].sonar_beam[
                beam].lon, survey_p->utm_central_meridian,
              &line_p->pings[ping].sonar_beam[beam].utmx, &line_p->pings[ping].sonar_beam[beam
                  ].utmy);
          }
      }

      free(tn);
      free(lat);
      free(lon);
      free(tp);
      free(i_lat);
      free(i_lon);
    }
}

double read_time(uint32_t tempo, uint32_t data) {
  uint32_t dia, mes, ano, jday, jday_31121969 = 2440587;
  ano = data / 10000;
  mes = (data % 10000) / 100;
  dia = (data % 100);
  ano += 8000;
  if (mes<3) {
    ano--;
    mes += 12;
  }
  jday = (ano * 365) + (ano / 4) - (ano / 100) + (ano / 400) - 1200820 + (mes * 153 + 3) / 5 - 92
      + dia - 1;
  jday -= jday_31121969; //Dia juliano desde 1970
  return (jday * 24.0 * 60.0 * 60.0 + tempo / 1000.0);
}
```

Listagem B.7 – *grid.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>

#include "structs.h"

#define NSTATS 5 //Número de valores mínimos para se realizar o binning

static int comp_double(const void * a, const void * b)
{
  if (*(double*)a > *(double*)b) return 1;
  else if (*(double*)a < *(double*)b) return -1;
  else return 0;
}

static void mean(double *arr, int size, double *mean) {
  *mean = 0.0;
  for (int i = 0; i < size; i++) {
    *mean += (arr[i] / (double)size);
  }
}
```

```
static void quartile(double *arr, int size, double *median, double *quatile_1, double *quatile_3) {
  qsort(arr, size, sizeof(double), comp_double);
  if (size % 2) {
    *median = arr[size / 2];
  }
  else {
    *median = (arr[size / 2] + arr[(size / 2) - 1]) / 2.0;
  }
  if (((size + 1) / 2) % 2) {
    *quatile_1 = arr[size / 4];
    *quatile_3 = arr[3 * size / 4];
  }
  else {
    *quatile_1 = (arr[size / 4] + arr[(size / 4) + 1]) / 2.0;
    *quatile_3 = (arr[3 * size / 4] + arr[(3 * size / 4) - 1]) / 2.0;
  }

}

static void find_borders(survey_t *survey_p) {
  double min_x, max_x, min_y, max_y;
  double mid_along = 0;

  double dx, dy, dist_along;

  min_x = survey_p->line_data[0].pings[0].sonar_beam[0].utmx;
  max_x = survey_p->line_data[0].pings[0].sonar_beam[0].utmx;
  min_y = survey_p->line_data[0].pings[0].sonar_beam[0].utmy;
  max_y = survey_p->line_data[0].pings[0].sonar_beam[0].utmy;

  for (int line = 0; line < survey_p->line_list.n_lines; line++) {
    line_t *line_p = &survey_p->line_data[line];
    for (int ping = 0; ping < line_p->n_pings; ping++) {
      for (int beam = 0; beam < line_p->pings[ping].n_beams; beam++) {
        if (min_x > line_p->pings[ping].sonar_beam[beam].utmx) {
          min_x = line_p->pings[ping].sonar_beam[beam].utmx;
        }
        if (max_x < line_p->pings[ping].sonar_beam[beam].utmx) {
          max_x = line_p->pings[ping].sonar_beam[beam].utmx;
        }
        if (min_y > line_p->pings[ping].sonar_beam[beam].utmy) {
          min_y = line_p->pings[ping].sonar_beam[beam].utmy;
        }
        if (max_y < line_p->pings[ping].sonar_beam[beam].utmy) {
          max_y = line_p->pings[ping].sonar_beam[beam].utmy;
        }
      }
      if (ping) {
        dx = line_p->pings[ping].utmx - line_p->pings[ping - 1].utmx;
        dy = line_p->pings[ping].utmy - line_p->pings[ping - 1].utmy;
        dist_along = sqrt(dx * dx + dy * dy);
        mid_along += dist_along / (line_p->n_pings * survey_p->line_list.n_lines);
      }
    }
  }

  survey_p->grid.min_x = min_x;
  survey_p->grid.max_x = max_x;
  survey_p->grid.min_y = min_y;
  survey_p->grid.max_y = max_y;

  survey_p->grid.mid_along = mid_along;
```

```
}

static void alloc_grid(grid_t *grid_p) {
  //grid.cell[x][y]
  double fac_x = 5.0, fac_y = 5.0; //Fatores multiplicativos
  grid_p->size_x = fac_x*grid_p->mid_along;
  grid_p->size_y = fac_y*grid_p->mid_along;
  grid_p->dx = (grid_p->max_x - grid_p->min_x);
  grid_p->dy = (grid_p->max_y - grid_p->min_y);
  grid_p->n_cols = (int)(grid_p->dx / grid_p->size_x);
  grid_p->n_lins = (int)(grid_p->dy / grid_p->size_y);
  grid_p->size_x = grid_p->dx / (double)grid_p->n_cols; //Ajustar tamanho
  grid_p->size_y = grid_p->dy / (double)grid_p->n_lins; //Ajustar tamanho
  grid_p->cell = calloc(grid_p->n_cols, sizeof(cell_t *));
  if (grid_p->cell == NULL) {
    printf("grid.cell sem memoria alocada!\n");
  }

  for (int col = 0; col < grid_p->n_cols; col++) {
    grid_p->cell[col] = calloc(grid_p->n_lins, sizeof(cell_t));
    if (grid_p->cell[col] == NULL) {
      printf("grid.cell.[%d] sem memoria alocada!\n", col);
    }
    for (int lin = 0; lin < grid_p->n_lins; lin++) {
      grid_p->cell[col][lin].cell_size = 0;
    }
  }
}

static void fill_grid(survey_t *survey_p) {
  grid_t *grid_p = &survey_p->grid;
  for (int line = 0; line < survey_p->line_list.n_lines; line++) {
    line_t *line_p = &survey_p->line_data[line];
    for (int ping = 0; ping < line_p->n_pings; ping++) {
      for (int beam = 0; beam < line_p->pings[ping].n_beams; beam++) {
        double dist_x = line_p->pings[ping].sonar_beam[beam].utmx - grid_p->min_x;
        double dist_y = line_p->pings[ping].sonar_beam[beam].utmy - grid_p->min_y;
        int col = (int)((dist_x / grid_p->dx)*grid_p->n_cols);
        int lin = (int)((dist_y / grid_p->dy)*grid_p->n_lins);

        if (col >= grid_p->n_cols) {
          col = grid_p->n_cols - 1;
        }

        if (lin >= grid_p->n_lins) {
          lin = grid_p->n_lins - 1;
        }

        if (grid_p->cell[col][lin].cell_size == 0) { //Alocação inicial
          grid_p->cell[col][lin].cell_buf = 10;
          grid_p->cell[col][lin].z = calloc(grid_p->cell[col][lin].cell_buf, sizeof(double));
          if (grid_p->cell[col][lin].z == NULL) {
            printf("grid.cell[%d][%d].z sem memoria alocada!\n", col, lin);
          }
          grid_p->cell[col][lin].bs = calloc(grid_p->cell[col][lin].cell_buf, sizeof(double));
          if (grid_p->cell[col][lin].bs == NULL) {
            printf("grid.cell[%d][%d].bs sem memoria alocada!\n", col, lin);
          }
          grid_p->cell[col][lin].angle = calloc(grid_p->cell[col][lin].cell_buf, sizeof(double));
          if (grid_p->cell[col][lin].angle == NULL) {
            printf("grid.cell[%d][%d].angle sem memoria alocada!\n", col, lin);
          }
        }
```

```
        if (grid_p->cell[col][lin].cell_size >= grid_p->cell[col][lin].cell_buf) {
          double *z, *bs, *angle;
          grid_p->cell[col][lin].cell_buf += 10;
          z = (double *)realloc(grid_p->cell[col][lin].z, grid_p->cell[col][lin].cell_buf*(sizeof(
              double)));
          if (grid_p->cell[col][lin].z == NULL) {
            printf("temp.cell[%d][%d].z sem memoria alocada!\n", col, lin);
          }
          bs = (double *)realloc(grid_p->cell[col][lin].bs, grid_p->cell[col][lin].cell_buf*(sizeof(
              double)));
          if (grid_p->cell[col][lin].bs == NULL) {
            printf("temp.cell[%d][%d].bs sem memoria alocada!\n", col, lin);
          }
          angle = (double *)realloc(grid_p->cell[col][lin].angle, grid_p->cell[col][lin].cell_buf*(
              sizeof(double)));
          if (angle == NULL) {
            printf("temp.cell[%d][%d].angle sem memoria alocada!\n", col, lin);
          }
          grid_p->cell[col][lin].z = z;
          grid_p->cell[col][lin].bs = bs;
          grid_p->cell[col][lin].angle = angle;
        }

        int index = grid_p->cell[col][lin].cell_size;
        grid_p->cell[col][lin].z[index] = line_p->pings[ping].sonar_beam[beam].z;
        grid_p->cell[col][lin].bs[index] = line_p->pings[ping].sonar_beam[beam].corr_bs;
        grid_p->cell[col][lin].angle[index] = line_p->pings[ping].sonar_beam[beam].angle;

        grid_p->cell[col][lin].cell_size++;
      }
    }
  }
}


static void stat_grid(grid_t *grid_p){
  for (int col = 0; col < grid_p->n_cols; col++) {
    for (int lin = 0; lin < grid_p->n_lins; lin++) {
      if (grid_p->cell[col][lin].cell_size > NSTATS) {
        quartile(grid_p->cell[col][lin].z, grid_p->cell[col][lin].cell_size, &grid_p->cell[col][lin].
            z_stat_cell.median,
          &grid_p->cell[col][lin].z_stat_cell.quartile_1, &grid_p->cell[col][lin].z_stat_cell.
              quartile_3);
        quartile(grid_p->cell[col][lin].bs, grid_p->cell[col][lin].cell_size, &grid_p->cell[col][lin
            ].bs_stat_cell.median,
          &grid_p->cell[col][lin].bs_stat_cell.quartile_1, &grid_p->cell[col][lin].bs_stat_cell.
              quartile_3);
        quartile(grid_p->cell[col][lin].angle, grid_p->cell[col][lin].cell_size, &grid_p->cell[col][
            lin].angle_stat_cell.median,
          &grid_p->cell[col][lin].angle_stat_cell.quartile_1, &grid_p->cell[col][lin].
              angle_stat_cell.quartile_3);

        grid_p->cell[col][lin].z_stat_cell.iqr = grid_p->cell[col][lin].z_stat_cell.quartile_3 -
            grid_p->cell[col][lin].z_stat_cell.quartile_1;
        grid_p->cell[col][lin].bs_stat_cell.iqr = grid_p->cell[col][lin].bs_stat_cell.quartile_3 -
            grid_p->cell[col][lin].bs_stat_cell.quartile_1;
        grid_p->cell[col][lin].angle_stat_cell.iqr = grid_p->cell[col][lin].angle_stat_cell.
            quartile_3 - grid_p->cell[col][lin].angle_stat_cell.quartile_1;
      }
      else { //Caso não haja valores suficientes para a realização do processamento estatístico
          atribuir "valor inválido"
```

```
        grid_p->cell[col][lin].z_stat_cell.median = INV_BIN;
        grid_p->cell[col][lin].z_stat_cell.quartile_1 = INV_BIN;
        grid_p->cell[col][lin].z_stat_cell.quartile_3 = INV_BIN;
        grid_p->cell[col][lin].z_stat_cell.iqr = INV_BIN;
        grid_p->cell[col][lin].bs_stat_cell.median = INV_BIN;
        grid_p->cell[col][lin].bs_stat_cell.quartile_1 = INV_BIN;
        grid_p->cell[col][lin].bs_stat_cell.quartile_3 = INV_BIN;
        grid_p->cell[col][lin].bs_stat_cell.iqr = INV_BIN;
        grid_p->cell[col][lin].angle_stat_cell.median = INV_BIN;
        grid_p->cell[col][lin].angle_stat_cell.quartile_1 = INV_BIN;
        grid_p->cell[col][lin].angle_stat_cell.quartile_3 = INV_BIN;
        grid_p->cell[col][lin].angle_stat_cell.iqr = INV_BIN;
      }

    }
  }
}

void create_grid(survey_t *survey_p) {
  find_borders(survey_p);
  alloc_grid(&survey_p->grid);
   fill_grid (survey_p);
  stat_grid(&survey_p->grid);
}
```

Listagem B.8 – *byteswap.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "structs.h"

uint16_t byteswap2u(uint16_t num) {
  return ((num >> 8) | (num << 8));
}

int16_t byteswap2(int16_t num) {
  return ((num >> 8) | (num << 8));
}

uint32_t byteswap4u(uint32_t num) {
  return ((num >> 24) & 0xff) | ((num << 8) & 0xff0000) | ((num >> 8) & 0xff00) | ((num << 24)
      & 0xff000000);
}

int32_t byteswap4(int32_t num) {
  return ((num >> 24) & 0xff) | ((num << 8) & 0xff0000) | ((num >> 8) & 0xff00) | ((num << 24)
      & 0xff000000);
}

float byteswapf(float num) {
  float value;
  char *floatToConvert = (char*)&num;
  char *returnFloat = (char*)&value;

  for (int i = 0; i < sizeof(float); i++) {
    returnFloat[i] = floatToConvert[sizeof(float) - 1 - i];
  }

  return value;
}
```

```
double byteswapd(double num) {
  double value;
  char *doubleToConvert = (char*)&num;
  char *returnDouble = (char*)&value;

  for (int i = 0; i < sizeof(double); i++) {
    returnDouble[i] = doubleToConvert[sizeof(double) - 1 - i];
  }

  return value;
}

void byteswapvalidation(line_t *line_p) {
  header_t header;
  fread(&header.n_bytes, sizeof(uint32_t), 1, line_p->line_in);
  fread(&header.start, sizeof(uint8_t), 1, line_p->line_in);
  fread(&header.datagram, sizeof(uint8_t), 1, line_p->line_in);
  fread(&header.model, sizeof(uint16_t), 1, line_p->line_in);

  uint16_t modelo = header.model;

  if (modelo == EM1002 || modelo == EM120 || modelo == EM300 || modelo == EM710 || modelo
      == EM2000 || modelo == EM3000 || modelo == EM3002 ||
    modelo == EM302 || modelo == EM122 || modelo == EM121A || modelo == ME70BO || modelo
        == EM2040 || modelo == EM2040C) {
    line_p->byteswap = 0; //Modelo válido, sem necessidade de realizar byteswap
    line_p->model = modelo;
  }
  else {
    line_p->byteswap = 1; //Modelo inválido, necessária realização de byteswap
    modelo = byteswap2u(modelo);
    if (modelo == EM1002 || modelo == EM120 || modelo == EM300 || modelo == EM710 ||
        modelo == EM2000 || modelo == EM3000 || modelo == EM3002 ||
      modelo == EM302 || modelo == EM122 || modelo == EM121A || modelo == ME70BO ||
          modelo == EM2040 || modelo == EM2040C) {
      //Realização de byteswap correta
      line_p->model = modelo;
    }
    else {
      printf("Modelo invalido\nErro na realizacao de byteswap\n");
      line_p->model = 0;
    }
  }

  rewind(line_p->line_in);
}
```

Listagem B.9 – *angle.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>

#include "structs.h"

double deg2rad(double degrees) {
  return (degrees * PI / 180);
}

double rad2deg(double radians) {
```

```
  return (radians * 180 / PI);
}
```

APÊNDICE C – *Script em Python*

Listagem C.1 – *plugin_SIMRAD_QGIS_3_2.py*

```
# Plugin UnB TCC
'''
Desenvolvimento de metodologias de quantização espacial para o processamento de
sinais de retroespalhamento acústico e batimetria adquiridos por sonares multifeixe
'''

# QGIS 3.2
from qgis.PyQt.QtCore import QCoreApplication, QSettings, QVariant
from qgis.core import QgsField, QgsFeature, QgsFeatureSink, QgsFeatureRequest, QgsProcessing
from qgis.core import QgsProcessingAlgorithm, QgsProcessingParameterFeatureSource
from qgis.core import QgsProcessingParameterFeatureSink
from qgis.PyQt.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QFileDialog, QLineEdit,
    QLabel
from qgis.PyQt.QtWidgets import QPlainTextEdit, QCheckBox
from qgis.utils import *
from qgis.core import *
from qgis.gui import *
from osgeo import gdal
from qgis.PyQt.QtGui import QColor, QDoubleValidator

# Python libraries
import time
import numpy as np
import os
import struct
from scipy import interpolate
from scipy import ndimage
import math


def get_project_CRS():
    return iface.mapCanvas().mapSettings().destinationCrs().authid()

class Buttons(QWidget):
    def processar_simrad(self):
        self.update_textbox('Processamento iniciado')

        bad_val = -30000
        solar_vector = np.asarray([1, 1, 1])

        #Validações
        interp = self.itplcheck.isChecked()

        pixel_size_bm = self.receive_input(self.field1.text(), 'Tamanho de pixel inválido para BM!'
            )
        pixel_size_bs = self.receive_input(self.field2.text(), 'Tamanho de pixel inválido para BS!')

        if not pixel_size_bm or not pixel_size_bs:
            return

        self.update_textbox('Tamanhos de pixel: BM={:.2f}, BS={:.2f}'.format(pixel_size_bm,
            pixel_size_bs), False)

        if interp:
            self.update_textbox('Será realizada interpolação na área dilatada.', False)
```

```
fs = QFileDialog.getOpenFileNames(QFileDialog(), self.btn_names[0], self.get_cur_path(),
    self.data_types[0])[0]

if not fs:
    self.update_textbox('Nenhum arquivo lido!')
    return

position = []
depth = []
layer_names = ['Retroespalhamento', 'Batimetria', 'Relevo de batimetria']
layer_ranges = []
bm_layers = []
bm_ranges = []
rel_bm_layers = []

# Grades de batimetria
self.update_textbox('Lendo arquivos')
for f in fs:
    folder_name, file_name = os.path.split(f)
    self.set_cur_path(folder_name)
    self.update_textbox('Lendo arquivo ' + file_name)
    bm_layers.append('bm_' + file_name)
    rel_bm_layers.append('rel_bm_' + file_name)
    pos_dgs, depth_dgs, model = read_pos_dep_datagrams(f)
    position.append(pos_dgs)
    depth.append(depth_dgs)

if model == 850:
    self.update_textbox('Modelo: ME70BO', False)
elif model == 2045:
    self.update_textbox('Modelo: EM2040C', False)
else:
    self.update_textbox('Modelo: EM{}'.format(model), False)

position, depth, epsg_code = interpolate_data(position, depth)
self.update_textbox(epsg_code, False)

self.update_textbox('Criando grades de batimetria')
grid, info = create_grid(depth, pixel_size_bm, bad_val)
if interp:
    grid['bm'] = interpolate_grid(grid['bm'], pixel_size_bm, bad_val)
for i, layer in enumerate(bm_layers):
    grid_bm = create_bm_grid(depth[i], info, pixel_size_bm, bad_val)
    min_val = np.amin(grid_bm[grid_bm>bad_val])
    max_val = np.amax(grid_bm)
    bm_ranges.append((min_val, max_val))
    if interp:
        grid_bm = interpolate_grid(grid_bm, pixel_size_bm, bad_val)
    save_np_array(grid_bm.astype(np.float32), '{}/{}.tif'.format(folder_name, layer))
    grid_bm_sol = dot_solar_vector(grid_bm, solar_vector, bad_val)
    grid_bm_rel = generate_rgb(normalize_grid(grid_bm, bad_val),
                               normalize_grid(grid_bm_sol, bad_val), bad_val)
    grid_bm_rel = (grid_bm_rel*255).astype(np.uint8)
    save_np_array_rgb(grid_bm_rel, '{}/{}.tif'.format(folder_name, rel_bm_layers[i]))


self.update_textbox('Criando grades de retroespalhamento')
grid['sid'] = []

# Grades de retroespalhamento
for i, f in enumerate(fs):
    folder_name, file_name = os.path.split(f)
    sid_dgs = read_seabed_datagrams(f)
```

```
        sid_dgs = sort_sid(sid_dgs)
        grid_sid = create_sid_grids(sid_dgs, depth[i], info, pixel_size_bs, bad_val)
        grid['sid'].append(grid_sid)

    grid['sid'] = create_sid_grid(grid['sid'], bad_val)
    if interp:
        grid['sid'] = interpolate_grid(grid['sid'], pixel_size_bs, bad_val)


    # Ajuste de imagem
    self.update_textbox('Criando grade de batimetria com relevo')
    grid['bm_solar'] = dot_solar_vector(grid['bm'], solar_vector, bad_val)
    grid['bm_RGB'] = generate_rgb(normalize_grid(grid['bm'].copy(), bad_val),
                                  normalize_grid(grid['bm_solar'].copy(), bad_val), bad_val)

    layer_names.extend(bm_layers)
    layer_names.extend(rel_bm_layers)

    self.update_textbox('Ajustando as imagens')
    wtfall_img = grid['sid'].astype(np.float32)
    btm_img = grid['bm'].astype(np.float32)
    btm_img_rgb = (grid['bm_RGB']*255).astype(np.uint8)


    self.update_textbox('Salvando as imagens')
    save_np_array(wtfall_img, '{}/{}.tif'.format(folder_name, layer_names[0]))
    save_np_array(btm_img, '{}/{}.tif'.format(folder_name, layer_names[1]))
    save_np_array_rgb(btm_img_rgb, '{}/{}.tif'.format(folder_name, layer_names[2]))


    # Carregar no QGIS
    remove_layers(layer_names)

    self.update_textbox('Georreferenciando')
    gcpList = [get_convex_control_points(depth, wtfall_img, bad_val, info), #
        retroespalhamento
        get_convex_control_points(depth, btm_img, bad_val, info)] # batimetria

    self.update_textbox('Salvando imagens georreferenciadas')
    for layer in layer_names:
        img_tiff = '{}/{}.tif'.format(folder_name, layer)
        geo_tiff = '{}/{}_geo.tif'.format(folder_name, layer)
        if layer == layer_names[0]:
            translate_warp(img_tiff, gcpList[0], epsg_code, geo_tiff)
        else:
            translate_warp(img_tiff, gcpList[1], epsg_code, geo_tiff)
        # os.remove(img_tiff)

    self.update_textbox('Carregando imagens georreferenciadas')
    min_val_bs = np.amin(wtfall_img[wtfall_img>bad_val])
    max_val_bs = np.amax(wtfall_img)
    layer_ranges.append((min_val_bs, max_val_bs))
    mean_bs = np.mean(wtfall_img[wtfall_img>bad_val])
    std_bs = np.std(wtfall_img[wtfall_img>bad_val])
    min_val_bm = np.amin(btm_img[btm_img>bad_val])
    max_val_bm = np.amax(btm_img)
    layer_ranges.append((min_val_bm, max_val_bm))
    layer_ranges.append((min_val_bm, max_val_bm))
    mean_bm = np.mean(btm_img[btm_img>bad_val])
    std_bm = np.std(btm_img[btm_img>bad_val])
    layer_ranges.extend(bm_ranges)

    self.update_textbox('Estatísticas das grades totais:',False)
    self.update_textbox('  Valor min BS: {:.2f}; Valor max BS: {:.2f}'.format(min_val_bs,
```

```
                        max_val_bs),False)
        self.update_textbox('  Valor médio BS: {:.2f}; Desvio padrão BS {:.2f}'.format(mean_bs,
                        std_bs),False)
        self.update_textbox('  Valor min BM: {:.2f}; Valor max BM: {:.2f}'.format(min_val_bm,
                        max_val_bm),False)
        self.update_textbox('  Valor médio BM: {:.2f}; Desvio padrão BM {:.2f}'.format(mean_bm,
                        std_bm),False)
        print(layer_ranges)
        for i, layer in enumerate(layer_names):
            self.update_textbox('Carregando ' + layer)
            if layer == layer_names[0]:
                load_image_QGIS('{}/{}_geo.tif'.format(folder_name, layer), layer,
                layer_ranges[i])
            elif layer == layer_names[1] or layer[0:2] == 'bm':
                load_image_QGIS('{}/{}_geo.tif'.format(folder_name, layer), layer,
                layer_ranges[i], False)
            else:
                load_multiband_image_QGIS('{}/{}_geo.tif'.format(folder_name, layer), layer)

        self.update_textbox('Processar .all: completo')


    def update_textbox(self, cur_text, status=True):
        if status:
            #QGIS 3.2
            iface.messageBar().pushInfo('Status', cur_text)

            cur_time = time.strftime('[%H:%M:%S] ')
            self.log.insertPlainText(cur_time + cur_text + '\n')
        else:
            self.log.insertPlainText('    ' + cur_text + '\n')

    def get_cur_path(self):
        cur_settings = QSettings()
        if cur_settings.value('simrad_plugin/curpath')==None:
            return ''
        else:
            return cur_settings.value('simrad_plugin/curpath')

    def set_cur_path(self, cur_path):
        cur_settings = QSettings()
        cur_settings.setValue('simrad_plugin/curpath', cur_path)

    def receive_input(self, inp_val, error_msg):
        try:
            flt_val = float(inp_val)
        except:
            self.update_textbox(error_msg)
            return False
        if flt_val < 1 or flt_val > 100:
            self.update_textbox(error_msg)
            return False
        return flt_val

    def __init__(self, parent=None):
        self.btn_names = [
            'Processar arquivos .all']
        self.data_types = [
            'Arquivos SIMRAD (*.all)']
        self.btn_funcs = [
            self.processar_simrad]

        self.epsg_full = get_project_CRS()
        QWidget.__init__(self, parent)
```

```
self.layout = QVBoxLayout()
self.rb = []

self.upper_layout1 = QHBoxLayout()
self.fieldLabel1 = QLabel()
self.fieldLabel1.setText('Tamanho de pixel BM: 1-100 (metros)')
self.fieldLabel1.setMinimumWidth(180)
self.upper_layout1.addWidget(self.fieldLabel1)
self.field1 = QLineEdit()
self.field1.setObjectName('Tamanho de pixel BM: 1-100 (metros)')
self.field1.setText('25.0')
self.field1.setValidator(QDoubleValidator(1.0,100.0,2))
self.field1.setMaxLength(8)
self.field1.setMaximumWidth(60)
self.upper_layout1.addWidget(self.field1)
self.layout.addLayout(self.upper_layout1,1)

self.upper_layout2 = QHBoxLayout()
self.fieldLabel2 = QLabel()
self.fieldLabel2.setText('Tamanho de pixel BS: 1-100 (metros)')
self.fieldLabel2.setMinimumWidth(180)
self.upper_layout2.addWidget(self.fieldLabel2)
self.field2 = QLineEdit()
self.field2.setObjectName('Tamanho de pixel BS: 1-100 (metros)')
self.field2.setText('10.0')
self.field2.setValidator(QDoubleValidator(1.0,100.0,2))
self.field2.setMaxLength(8)
self.field2.setMaximumWidth(60)
self.upper_layout2.addWidget(self.field2)
self.layout.addLayout(self.upper_layout2,1)

self.itplcheck = QCheckBox('Aplicar dilatação e interpolação')
self.itplcheck.setChecked(True)
self.itplcheck.setToolTip('Aplica dilatação e interpola na área dilatada. ' +
    'É uma tentativa de se preencher "buracos".')
self.layout.addWidget(self.itplcheck)

for b in range(len(self.btn_names)):
    self.rb.append(QPushButton(self.btn_names[b]))
    self.rb[b].clicked.connect(self.btn_funcs[b])
    self.layout.addWidget(self.rb[b])

self.logLabel = QLabel()
self.logLabel.setText('Log:')
self.layout.addWidget(self.logLabel)

self.log = QPlainTextEdit()
self.log.setReadOnly(True)
self.log.setMaximumBlockCount(100)
self.layout.addWidget(self.log)

self.bottomlayout = QHBoxLayout()
self.clearbt = QPushButton('Limpar log')
self.clearbt.clicked.connect(self.log.clear)
self.bottomlayout.addWidget(self.clearbt)
self.exitbt = QPushButton('Sair')
self.exitbt.clicked.connect(self.close)
self.bottomlayout.addWidget(self.exitbt)
self.layout.addLayout(self.bottomlayout)

self.setWindowTitle('Plugin SIMRAD')
self.setLayout(self.layout)
```

```
##############################################
# Leitura dos datagramas
##############################################
def read_pos_dep_datagrams(file_name):
    file_size = os.path.getsize(file_name)
    pos_dgs = []
    depth_dgs = []
    model_numbers = [1002, 120, 300, 710, 2000, 3000,
      3020, 302, 122, 121, 850, 2040, 2045]
    endian = '>' # '>' = big-endian, '<' = little-endian no módulo struct
    with open(file_name, 'rb') as f:
        model = struct.unpack('>IBBH',f.read(8))[3]
    if model not in model_numbers:
        with open(file_name, 'rb') as f:
            model = struct.unpack('<IBBH',f.read(8))[3]
        if model in model_numbers:
            endian = '<'
        else:
            raise ValueError('Erro de modelo para o arquivo {}'.format(
                os.path.split(file_name)[1]))

    with open(file_name, 'rb') as f:
        while f.tell()<file_size:
            cur_dg_len = struct.unpack(endian + 'I',f.read(4))[0]
            cur_dg_id = struct.unpack(endian + 'cc',f.read(2))[1]
            cur_dg = f.read(cur_dg_len-2)
            if checksum_error(cur_dg, cur_dg_id, endian):
                raise ValueError('Erro de checksum para o datagrama {}'.format(cur_dg_id))
            if(cur_dg_id==b'P'):
                pos_dgs.append(read_position_datagrams(cur_dg, endian))
            elif(cur_dg_id==b'D'):
                depth_dgs.append(read_depth_datagrams(cur_dg, endian))
            elif(cur_dg_id==b'X'):
                depth_dgs.append(read_xyz_datagrams(cur_dg, endian))
    return pos_dgs, depth_dgs, model

def read_seabed_datagrams(file_name):
    file_size = os.path.getsize(file_name)
    sid_dgs = []
    model_numbers = [1002, 120, 300, 710, 2000, 3000,
      3020, 302, 122, 121, 850, 2040, 2045]
    endian = '>' # '>' = big-endian, '<' = little-endian no módulo struct
    with open(file_name, 'rb') as f:
        model = struct.unpack('>IBBH',f.read(8))[3]
    if model not in model_numbers:
        with open(file_name, 'rb') as f:
            model = struct.unpack('<IBBH',f.read(8))[3]
        if model in model_numbers:
            endian = '<'
        else:
            raise ValueError('Erro de modelo para o arquivo {}'.format(
                os.path.split(file_name)[1]))

    with open(file_name, 'rb') as f:
        while f.tell()<file_size:
            cur_dg_len = struct.unpack(endian + 'I',f.read(4))[0]
            cur_dg_id = struct.unpack(endian + 'cc',f.read(2))[1]
            cur_dg = f.read(cur_dg_len-2)
            if checksum_error(cur_dg, cur_dg_id, endian):
                raise ValueError('Erro de checksum para o datagrama {}'.format(cur_dg_id))
            elif(cur_dg_id==b'S'):
                sid_dgs.append(read_sid_datagrams(cur_dg, endian))
```

```
            elif (cur_dg_id==b'Y'):
                sid_dgs.append(read_sid89_datagrams(cur_dg, endian))
    return sid_dgs

def checksum_error(cur_bytes, cur_dg_id, endian):
    dg_checksum = struct.unpack(endian + 'H', cur_bytes[-2:])[0]
    cur_bytes = cur_bytes[:-3]
    cur_bytes_sum = sum(struct.unpack(endian + 'B'*len(cur_bytes), cur_bytes))
    cur_bytes_sum += struct.unpack(endian + 'B',cur_dg_id)[0]
    return (cur_bytes_sum & 65535) != dg_checksum

def read_initial_datagram_bytes(cur_bytes, field_names, field_types, endian):
    field_types = endian + field_types
    cur_index = struct.calcsize(field_types)
    field_values = struct.unpack(field_types, cur_bytes[:cur_index])
    cur_dg = dict(zip(field_names, field_values))
    return cur_dg, cur_index

def read_cycled_datagram_bytes(cur_bytes, cycle_len, field_names, field_types, endian):
    field_types = endian + field_types*cycle_len
    cur_index = struct.calcsize(field_types)
    field_values = struct.unpack(field_types, cur_bytes[:cur_index])
    cur_dg = {}
    N = len(field_names)
    for k in range(N):
        cur_dg[field_names[k]] = field_values[k::N]
    return cur_dg, cur_index

def read_datagram_array(cur_bytes, array_len, field_name, field_type, endian):
    field_type = endian + field_type*array_len
    cur_index = struct.calcsize(field_type)
    cur_dg = {}
    cur_dg[field_name] = struct.unpack(field_type, cur_bytes[:cur_index])
    return cur_dg, cur_index

def read_position_datagrams(cur_bytes, endian):
    field_names = ('em_model', 'date', 'time', 'pos_counter', 'system_sn',
        'latitude', 'longitude', 'measure_fix_qual_cm', 'speed_cm_s', 'course_dot01',
        'heading_dot01', 'pos_system_descriptor', 'pos_in_dg_N')
    field_types = 'HIIHHiiHHHHBB'
    cur_dg = read_initial_datagram_bytes(cur_bytes,
        field_names, field_types, endian)[0]
    cur_dg['latitude'] = float(cur_dg['latitude'])/20000000
    cur_dg['longitude'] = float(cur_dg['longitude'])/10000000
    return cur_dg

def read_depth_datagrams(cur_bytes, endian):
    field_names = ('em_model', 'date', 'time', 'ping', 'system_sn', 'heading_dot01',
        'sound_speed_dm_s', 'depth_re_water_level', 'N_max', 'N', 'z_res', 'x_y_res', 'smp_freq')
    field_types = 'HIIHHHHHBBBBH'
    cur_dg, cur_index = read_initial_datagram_bytes(cur_bytes,
        field_names, field_types, endian)
    field_names = ('z', 'y', 'x',
        'beam_dep_angle_dot01', 'beam_azi_angle_dot01',
        'range_1way', 'quality_factor',
        'length_det_win',
        'reflectivity_dot5dB', 'beam_number')
    field_types = 'hhhhHHBBbB'
    cur_extra_dg, cur_index = read_cycled_datagram_bytes(cur_bytes[cur_index:], cur_dg['N'],
        field_names, field_types, endian)
    cur_dg.update(cur_extra_dg)
    cur_dg['z'] = [float(z)*float(cur_dg['z_res'])*0.01 for z in cur_dg['z']]
    cur_dg['y'] = [float(y)*float(cur_dg['x_y_res'])*0.01 for y in cur_dg['y']]
```

```
    cur_dg['x'] = [float(x)*float(cur_dg['x_y_res'])*0.01 for x in cur_dg['x']]
    cur_dg[' reflectivity '] = [float(bs)*0.5 for bs in cur_dg['reflectivity_dot5dB']]
    cur_dg['sound_speed'] = float(cur_dg['sound_speed_dm_s'])*0.1
    return cur_dg

def read_xyz_datagrams(cur_bytes, endian):
    field_names = ('em_model', 'date', 'time', 'ping', 'system_sn', 'heading_dot01',
    'sound_speed_dm_s', 'depth_re_water_level', 'N', 'N_valid', 'smp_freq', 'info',
    'spare1', 'spare2', 'spare3')
    field_types = 'HIIHHHHfHHfB' + 3*'B'
    cur_dg, cur_index = read_initial_datagram_bytes(cur_bytes,
        field_names, field_types, endian)
    field_names = ('z', 'y', 'x', 'length_det_win', 'quality_factor', 'beam_angle_adj_dot01',
        'detect_info', 'real_time_info', 'reflectivity_dot1dB')
    field_types = 'fffHBbBbh'
    cur_extra_dg, cur_index = read_cycled_datagram_bytes(cur_bytes[cur_index:], cur_dg['N'],
        field_names, field_types, endian)
    cur_dg.update(cur_extra_dg)
    cur_dg[' reflectivity '] = [float(bs)*0.1 for bs in cur_dg['reflectivity_dot1dB']]
    cur_dg['sound_speed'] = float(cur_dg['sound_speed_dm_s'])*0.1
    return cur_dg

def read_sid_datagrams(cur_bytes, endian):
    field_names = ('em_model', 'date', 'time', 'ping', 'system_sn', 'mean_abs_coef_db_km',
        'pulse_length_us', 'range_norm_inc', 'start_range_tvg', 'stop_range_tvg',
        'bsn', 'bso', 'tx_beamwidth', 'tvg_law', 'N')
    field_types = 'HIIHHHHHHHHbbHBB'
    cur_dg, cur_index1 = read_initial_datagram_bytes(cur_bytes,
        field_names, field_types, endian)
    field_names = ('beam_idx', 'sort_dir', 'Ns', 'cnt_num')
    field_types = 'BbHH'
    cur_extra_dg, cur_index2 = read_cycled_datagram_bytes(cur_bytes[cur_index1:], cur_dg['N'],
        field_names, field_types, endian)
    cur_dg.update(cur_extra_dg)
    cur_extra_dg = read_datagram_array(cur_bytes[(cur_index1+cur_index2):],
        sum(cur_dg['Ns']), 'samples', 'b', endian)[0]
    cur_dg.update(cur_extra_dg)
    cur_dg['samples'] = [float(sample)*0.5 for sample in cur_dg['samples']]
    return cur_dg

def read_sid89_datagrams(cur_bytes, endian):
    field_names = ('em_model', 'date', 'time', 'ping', 'system_sn', 'smp_freq',
        'range_norm_inc', 'bsn', 'bso', 'tx_beamwidth', 'tvg_law', 'N')
    field_types = 'HIIHHfHhhHHH'
    cur_dg, cur_index1 = read_initial_datagram_bytes(cur_bytes,
        field_names, field_types, endian)
    field_names = ('sort_dir', 'detect_info', 'Ns', 'cnt_num')
    field_types = 'BbHH'
    cur_extra_dg, cur_index2 = read_cycled_datagram_bytes(cur_bytes[cur_index1:], cur_dg['N'],
        field_names, field_types, endian)
    cur_dg.update(cur_extra_dg)
    cur_extra_dg = read_datagram_array(cur_bytes[(cur_index1+cur_index2):],
        sum(cur_dg['Ns']), 'samples', 'h', endian)[0]
    cur_dg.update(cur_extra_dg)
    cur_dg['bsn'] = float(cur_dg['bsn'])*0.1
    cur_dg['bso'] = float(cur_dg['bso'])*0.1
    cur_dg['samples'] = [float(sample)*0.1 for sample in cur_dg['samples']]
    return cur_dg

##############################################
# Interpolação e tratamento dos dados
##############################################
def interpolate_data(position, depth):
```

```
    central_meridian, epsg_code = get_utm(position)

    for n in range(len(position)):
        pos_dgs = position[n]
        depth_dgs = depth[n]
        t_pos = get_datenum(pos_dgs)
        t_xyz = get_datenum(depth_dgs)
        for i, t in enumerate(t_pos):
            pos_dgs[i]['epoch'] = t
        for i, t in enumerate(t_xyz):
            depth_dgs[i]['epoch'] = t
        pos_dgs = organize_dtgs(pos_dgs)
        depth_dgs = organize_dtgs(depth_dgs)
        t_pos = [pos['epoch'] for pos in pos_dgs]
        t_xyz = [depth['epoch'] for depth in depth_dgs]

        latitude  = np.asarray([p['latitude'] for p in pos_dgs])
        longitude = np.asarray([p['longitude'] for p in pos_dgs])
        xs, ys = geoutm(latitude, longitude, central_meridian)

        fx = interpolate.interp1d(t_pos, xs, fill_value='extrapolate')
        fy = interpolate.interp1d(t_pos, ys, fill_value='extrapolate')
        xs_utm_xyz = fx(t_xyz)
        ys_utm_xyz = fy(t_xyz)

        for i in range(len(xs_utm_xyz)):
            depth_dgs[i]['utmx'] = xs_utm_xyz[i]
            depth_dgs[i]['utmy'] = ys_utm_xyz[i]

        depth_dgs = locate_beams(depth_dgs)

        position[n] = pos_dgs
        depth[n] = depth_dgs

    return position, depth, epsg_code

def get_datenum(dgs):
    cur_seconds = [jday(dg['date'], dg['time']) for dg in dgs]
    return cur_seconds

def jday(date, time):
    year = int(date/10000)
    month = int((date%10000)/100)
    day = int(date%100)
    year += 8000
    if month<3:
        year -= 1
        month += 12
    jday = int((year * 365) + (year / 4) - (year / 100) + (year / 400) - 1200820 +
        (month * 153 + 3) / 5 - 92 + day - 1)
    jday -= 2440587 # Julian day since 1970
    return float(jday) * 24.0 * 60.0 * 60.0 + time / 1000.0

def organize_dtgs(dtgs):
    dtgs.sort(key=lambda x: x['epoch'])
    i = 1
    while i < len(dtgs):
        if dtgs[i]['epoch'] == dtgs[i-1]['epoch']:
            del dtgs[i]
        i += 1
    return dtgs

def sort_sid(sid_dgs):
    t_sid = get_datenum(sid_dgs)
```

```
    for i, t in enumerate(t_sid):
        sid_dgs[i]['epoch'] = t
    sid_dgs = organize_dtgs(sid_dgs)

    return sid_dgs

def get_soundspeed(depth_dgs):
    sound_speed = 0
    for depth_dg in depth_dgs:
        sound_speed += depth_dg['sound_speed']
    sound_speed /= len(depth_dg)
    return sound_speed

def get_utm(position):
    latitude = []
    longitude = []
    for pos_dgs in position:
        lat = [pos['latitude'] for pos in pos_dgs]
        lon = [pos['longitude'] for pos in pos_dgs]
        latitude.extend(lat)
        longitude.extend(lon)
    min_lat = np.amin(latitude)
    max_lat = np.amax(latitude)
    mid_lat = max_lat + min_lat
    mid_lat = int(mid_lat/2)
    min_lon = np.amin(longitude)
    max_lon = np.amax(longitude)
    mid_lon = max_lon + min_lon
    mid_lon = int(mid_lon/2)

    utm_longs = np.asarray(range(-180,180))
    utm_zones = np.zeros(361)
    utm_zones[utm_longs<=0] = 30-np.floor(-utm_longs[utm_longs<=0]/6)
    utm_zones[utm_longs>0] = 31+np.floor(utm_longs[utm_longs>0]/6)
    utm_cnt_mer=np.floor(utm_longs/6)*6+3
    epsg_code = 'EPSG:32'
    if mid_lat<0:
        epsg_code += '7'
    else:
        epsg_code += '6'
    epsg_code = '{}{}'.format(epsg_code, utm_zones[utm_longs==np.round(mid_lon)][0])
    central_meridian = utm_cnt_mer[utm_longs==np.round(mid_lon)]

    return central_meridian, epsg_code

def geoutm(latitude, longitude, central_meridian):
    # WGS84
    WGS84_ECCEN = 0.0818191908417579
    WGS84_RADIUS = 6378137 # units are meters

    k0 = 0.9996        # central scale factor
    x0 = 500000.0      # x0 to be added in standard UTM
    y0 = 10000000.0    # y0 to be added in standard UTM for southern hemisphere

    e2 = WGS84_ECCEN**2
    a = WGS84_RADIUS

    e4 = e2**2
    e6 = e2**3
    m1 = 1.0 - e2 / 4.0 - 3.0*e4 / 64.0 - 5.0*e6 / 256.0
    m2 = 3.0*e2 / 8.0 + 3.0*e4 / 32.0 + 45.0*e6 / 1024.0
    m3 = 15.0*e4 / 256.0 + 45.0*e6 / 1024.0
    m4 = 35.0*e6 / 3072.0
    epr2 = e2 / (1.0 - e2)
```

```
    dphi = latitude*math.pi/180  # dphi = phi*math.pi/180
    dlam = longitude*math.pi/180 # dlam = lam*math.pi/180

    clam = central_meridian #-57
    dclam = clam*math.pi/180
    cosphi = np.asarray([math.cos(dp) for dp in dphi])
    sinphi = np.asarray([math.sin(dp) for dp in dphi])
    tanphi = np.asarray([math.tan(dp) for dp in dphi])
    n = np.asarray([a / math.sqrt(1.0 - e2 * (sp**2)) for sp in sinphi])
    t = tanphi**2
    t2 = t**2
    c = epr2 * (cosphi**2)
    b = (dlam - dclam) * cosphi # b is A in Snyder's formulas
    b2 = b*b
    b3 = b2*b
    b4 = b3*b
    b5 = b4*b
    b6 = b5*b
    dummy1 = np.asarray([math.sin(2.0*dp) for dp in dphi])
    dummy2 = np.asarray([math.sin(4.0*dp) for dp in dphi])
    dummy3 = np.asarray([math.sin(6.0*dp) for dp in dphi])
    m = a * (m1*dphi - m2*dummy1 + m3*dummy2 - m4*dummy3)
    xs = k0 * n * (b + (1.0 - t + c)*b3 / 6.0 + (5.0 - 18.0*t + t2 + 72.0*c - 58.0*epr2)*
        b5 / 120.0)
    xs = xs + x0
    ys = k0 * (m + n*tanphi*(b2 / 2.0 + (5.0 - t + 9.0*c + 4.0*(c**2))*b4 / 24.0 +
        (61.0 - 58.0*t + t2 + 600.0*c - 330.0*epr2)*b6 / 720.0))
    if max(latitude)<0.0: # (hemisphere ~= 0):
        ys = ys + y0
    return xs, ys

def locate_beams(depth_dgs):
    for n, depth_dg in enumerate(depth_dgs):
        utmx = depth_dg['utmx']
        utmy = depth_dg['utmy']
        x = np.asarray(depth_dg['x'])
        y = np.asarray(depth_dg['y'])
        z = np.asarray(depth_dg['z'])
        length = np.sqrt((x**2) + (y**2))
        azimuth = depth_dg['heading_dot01']*0.01*math.pi/180 * np.ones(x.shape)
        azimuth[y>0] += math.pi/2 - np.arctan(x[y>0]/np.abs(y[y>0]))
        azimuth[y<0] += -math.pi/2 + np.arctan(x[y<0]/np.abs(y[y<0]))

        dy = length*np.cos(azimuth)
        dx = length*np.sin(azimuth)
        utmxs = utmx + dx
        utmys = utmy + dy

        full_length = np.sqrt((length**2) + (z**2))
        depression = np.arctan2(full_length, z)

        depth_dg['utmxs'] = utmxs.tolist()
        depth_dg['utmys'] = utmys.tolist()
        depth_dg['length'] = length.tolist()
        depth_dg['azimuth'] = azimuth.tolist()
        depth_dg['full_length'] = full_length.tolist()
        depth_dg['depression'] = depression.tolist()

        depth_dgs[n] = depth_dg

    return depth_dgs

def find_borders(depth):
    utmxs = []
```

```
    utmys = []

    for depth_dgs in depth:
        utmx = [[min(depth_dg['utmxs']), max(depth_dg['utmxs'])] for depth_dg in depth_dgs]
        utmy = [[min(depth_dg['utmys']), max(depth_dg['utmys'])] for depth_dg in depth_dgs]
        utmxs.extend(utmx)
        utmys.extend(utmy)

    min_x = np.amin(utmxs)
    max_x = np.amax(utmxs)
    dx = max_x - min_x
    min_y = np.amin(utmys)
    max_y = np.amax(utmys)
    dy = max_y - min_y

    info = {'min_x':min_x, 'max_x':max_x, 'dx':dx,
            'min_y':min_y, 'max_y':max_y, 'dy':dy}

    return info

##############################################
# Criação das grades
##############################################
def create_grid(depth, pixel_size, no_val):
    info = find_borders(depth)

    # Largura e altura, com uma borda de 10 pixels para possivel interpolação
    W = int(info['dx']/pixel_size) + 20
    H = int(info['dy']/pixel_size) + 20

    grid_bm = [[[] for y in range(W)] for x in range(H)] # grid[y][x]
    grid_bs = [[[] for y in range(W)] for x in range(H)] # grid[y][x]

    for depth_dgs in depth:
        for depth_dg in depth_dgs:
            dist_x = np.subtract(depth_dg['utmxs'], info['min_x'])
            dist_y = np.subtract(depth_dg['utmys'], info['min_y'])

            cols = np.asarray((dist_x/info['dx'])*(W-20), dtype=np.uint32)
            lins = np.asarray((dist_y/info['dy'])*(H-20), dtype=np.uint32)
            cols += 10
            lins += 10
            cols[cols>(W-10)] = W - 10
            lins[lins>(H-10)] = H - 10

            for i, (col, lin) in enumerate(zip(cols, lins)):
                grid_bm[lin][col].append(depth_dg['z'][i])
                grid_bs[lin][col].append(depth_dg['reflectivity'][i])

    for lin in range(H):
        for col in range(W):
            if len(grid_bm[lin][col]) >= 3:
                grid_bm[lin][col] = np.median(grid_bm[lin][col])
            else:
                grid_bm[lin][col] = no_val
            if len(grid_bs[lin][col]) >= 3:
                grid_bs[lin][col] = np.median(grid_bs[lin][col])
            else:
                grid_bs[lin][col] = no_val

    grid = {'bs': np.asarray(grid_bs), 'bm': np.asarray(grid_bm)}
    info['W'], info['H'] = W, H

    return grid, info
```

```
def create_bm_grid(depth_dgs, info, pixel_size, no_val):
    # Largura e altura, com uma borda de 10 pixels para possivel interpolação
    W = int(info['dx']/pixel_size) + 20
    H = int(info['dy']/pixel_size) + 20

    grid = [[[] for y in range(W)] for x in range(H)] # grid[y][x]

    for depth_dg in depth_dgs:
        dist_x = np.subtract(depth_dg['utmxs'], info['min_x'])
        dist_y = np.subtract(depth_dg['utmys'], info['min_y'])

        cols = np.asarray((dist_x/info['dx'])*(W-20), dtype=np.uint32)
        lins = np.asarray((dist_y/info['dy'])*(H-20), dtype=np.uint32)
        cols += 10
        lins += 10
        cols[cols>(W-10)] = W - 10
        lins[lins>(H-10)] = H - 10

        for i, (col, lin) in enumerate(zip(cols, lins)):
            grid[lin][col].append(depth_dg['z'][i])

    for lin in range(H):
        for col in range(W):
            if len(grid[lin][col]) >= 3:
                grid[lin][col] = np.median(grid[lin][col])
            else:
                grid[lin][col] = no_val

    grid = np.asarray(grid)

    return grid

def create_sid_grids(sid_dgs, depth_dgs, info, pixel_size, no_val):
    # Largura e altura, com uma borda de 10 pixels para possivel interpolação
    W = int(info['dx']/pixel_size) + 20
    H = int(info['dy']/pixel_size) + 20

    grid = [[[] for y in range(W)] for x in range(H)] # grid[y][x]

    for sid_dg, depth_dg in zip(sid_dgs,depth_dgs):
        samples = np.asarray(sid_dg['samples'])
        Ns = np.asarray(sid_dg['Ns'])
        # Indices de onde começam as amostras (último indice éa quantidade total de amostras)
        i_smps = np.cumsum(Ns)
        i_smps = np.insert(i_smps,0,0)
        cnt_num = np.asarray(sid_dg['cnt_num']) # Número da amostra central
        cnt_num -= 1 # Indice da amostra central

        utmx = depth_dg['utmx']
        utmy = depth_dg['utmy']
        z = np.asarray(depth_dg['z'])
        azimuth = np.asarray(depth_dg['azimuth'])
        full_length = np.asarray(depth_dg['full_length'])

        smp_dist = depth_dg['sound_speed'] / depth_dg['smp_freq'] # Distância entre duas
            amostras

        for j in range(len(i_smps)-1):
            l, h = i_smps[j], i_smps[j+1]
            c = cnt_num[j]

            beam_smps = samples[l:h]
            # Distâncias em relação à amostra central
            smp_dists = np.asarray(list(range(-c,h-l-c))) * smp_dist
```

```
            if sid_dg['sort_dir'][j]<0: # Sort direction = -1
                smp_dists *= -1

            full_lengths = full_length[j] + smp_dists
            lengths = np.sqrt((full_lengths**2) - (z[j]**2))
            dy = lengths*np.cos(azimuth[j])
            dx = lengths*np.sin(azimuth[j])
            utmxsmp = utmx + dx
            utmysmp = utmy + dy

            dist_x = utmxsmp - info['min_x']
            dist_y = utmysmp - info['min_y']

            cols = np.asarray((dist_x/info['dx'])*(W-20), dtype=np.int32)
            lins = np.asarray((dist_y/info['dy'])*(H-20), dtype=np.int32)
            cols += 10
            lins += 10
            cols[cols<0] = 0
            lins[lins<0] = 0
            cols[cols>(W-1)] = W-1
            lins[lins>(H-1)] = H-1

            for k, (col, lin) in enumerate(zip(cols, lins)):
                grid[lin][col].append(beam_smps[k])

    for lin in range(H):
        for col in range(W):
            if len(grid[lin][col]) >= 3:
                grid[lin][col] = np.median(grid[lin][col])
            else:
                grid[lin][col] = no_val

    return grid

def create_sid_grid(grid, no_val):
    grid = np.asarray(grid)
    H, W = grid.shape[1:3]
    grid = np.transpose(grid, (1,2,0)).tolist()

    for lin in range(H):
        for col in range(W):
            grid[lin][col] = [val for val in grid[lin][col] if val!=no_val]
            if grid[lin][col]:
                grid[lin][col] = np.median(grid[lin][col])
            else:
                grid[lin][col] = no_val

    grid = np.asarray(grid)

    return grid

def dot_solar_vector(grid, solar_vector, bad_val):
    normal = np.ones(grid.shape)*bad_val

    for lin in range(1, grid.shape[0]-1):
        for col in range(1, grid.shape[1]-1):
            if grid[lin][col] != bad_val:
                a, b = grid[lin][col+1], grid[lin-1][col]
                c, d = grid[lin][col-1], grid[lin+1][col]
                p = np.asarray([col, lin, grid[lin][col]])
                pa = np.asarray([col+1, lin, a])
                pb = np.asarray([col, lin-1, b])
                pc = np.asarray([col-1, lin, c])
                pd = np.asarray([col, lin+1, d])
```

```
                if a != bad_val and b != bad_val:
                    cross_prod = np.cross(pa-p, pb-p)
                    normal[lin][col] = np.dot(cross_prod, solar_vector)
                elif b != bad_val and c != bad_val:
                    cross_prod = np.cross(pb-p, pc-p)
                    normal[lin][col] = np.dot(cross_prod, solar_vector)
                elif c != bad_val and d != bad_val:
                    cross_prod = np.cross(pc-p, pd-p)
                    normal[lin][col] = np.dot(cross_prod, solar_vector)
                elif d != bad_val and a != bad_val:
                    cross_prod = np.cross(pd-p, pa-p)
                    normal[lin][col] = np.dot(cross_prod, solar_vector)
            else:
                grid[lin][col] = bad_val

    return normal

def normalize_grid(grid, bad_val):
    min_val = grid[grid>bad_val].min()
    max_val = grid.max()

    grid[grid>bad_val] = (grid[grid>bad_val] - min_val)/(max_val - min_val)

    return grid

def generate_rgb(H, L, bad_val):
    import colorsys
    R = np.zeros(H.shape)
    G = np.zeros(H.shape)
    B = np.zeros(H.shape)

    for lin in range(H.shape[0]):
        for col in range(H.shape[1]):
            h = H[lin][col]
            l = L[lin][col]
            if h != bad_val and l != bad_val:
                r, g, b = colorsys.hls_to_rgb((2/3)*h, l, 0.8)
                R[lin][col] = r
                G[lin][col] = g
                B[lin][col] = b

    return np.asarray([R, G, B])

def interpolate_grid(grid, pixel_size, bad_val):
    s_size = int(100/pixel_size)
    if s_size < 2:
        s_size = 2
    elif s_size > 9:
        s_size = 9

    valid_pixels = grid != bad_val
    structure = np.ones((s_size,s_size)) == 1
    interp_pixels = ndimage.binary_closing(valid_pixels, structure=structure)

    points = []
    values = []
    interp_points = []

    for lin in range(grid.shape[0]):
        for col in range(grid.shape[1]):
            if grid[lin][col] != bad_val:
                points.append((lin, col))
                values.append(grid[lin][col])
            if interp_pixels[lin][col]:
```

```
                interp_points.append((lin, col))

    interp_values = interpolate.griddata(points, values, interp_points, method='cubic',
        fill_value=bad_val) # Interpola pontos

    interp_grid = np.ones(grid.shape)*bad_val
    for i, (lin, col) in enumerate(interp_points):
        interp_grid[lin][col] = interp_values[i]

    min_val = np.amin(grid[grid>bad_val])
    max_val = np.amax(grid)

    interp_grid[np.logical_and(interp_grid>bad_val, interp_grid<min_val)] = min_val
    interp_grid[interp_grid>max_val] = max_val

    return interp_grid

##################################################
# Arquivamento das imagens e carregamento no QGIS
##################################################
def get_convex_control_points(depth, img, bad_val, info):
    Ny = 40
    Nx = 40
    H = img.shape[0]
    W = img.shape[1]
    stepy = int(H/Ny) + 1
    stepx = int(W/Nx) + 1
    ys_img = list(range(0,H,stepy))
    ys_img[-1] = H-1
    real_xy = []
    img_xy = []
    for y in ys_img:
        good_vals = np.nonzero(img[y,:]!=bad_val)
        for x in good_vals[0][0:-1:stepx]:
            x_utm = (float(x-10) / float(W-20))*info['dx'] + info['min_x']
            y_utm = (float(y-10) / float(H-20))*info['dy'] + info['min_y']
            cur_real_xy = [x_utm, y_utm]
            cur_img_xy = [float(x),float(y)]
            real_xy.append(cur_real_xy)
            img_xy.append(cur_img_xy)
    gcpList = []
    for k in range(0,len(real_xy)):
        gcpList.append(gdal.GCP(real_xy[k][0], real_xy[k][1], 0, img_xy[k][0], img_xy[k][1]))
    return gcpList


def save_np_array(img_array, file_name):
    if img_array.dtype=='uint16':
        data_type = gdal.GDT_UInt16
    elif img_array.dtype=='int16':
        print('OK')
        data_type = gdal.GDT_Int16
    elif img_array.dtype=='float32':
        data_type = gdal.GDT_Float32
    else: # if img_array.dtype=='uint8':
        data_type = gdal.GDT_Byte

    # Create(file_name, width, height, num_bands, data_type)
    dst_ds = gdal.GetDriverByName('GTiff').Create(file_name,
        img_array.shape[1], img_array.shape[0], 1, data_type)
    dst_ds.GetRasterBand(1).WriteArray(img_array)

def save_np_array_rgb(img_array, file_name):
    if img_array.dtype=='uint16':
```

```
        data_type = gdal.GDT_UInt16
    elif img_array.dtype=='int16':
        print('OK')
        data_type = gdal.GDT_Int16
    elif img_array.dtype=='float32':
        data_type = gdal.GDT_Float32
    else: # if img_array.dtype=='uint8':
        data_type = gdal.GDT_Byte

    # Create(file_name, width, height, num_bands, data_type)
    dst_ds = gdal.GetDriverByName('GTiff').Create(file_name,
        img_array.shape[2], img_array.shape[1], 3, data_type)
    dst_ds.GetRasterBand(1).WriteArray(img_array[0])
    dst_ds.GetRasterBand(2).WriteArray(img_array[1])
    dst_ds.GetRasterBand(3).WriteArray(img_array[2])

def translate_warp(input_tif, input_points, epsg_code, output_tif):
    use_thinplate = True
    errorThreshold = 200

    in_tif = gdal.Open(input_tif)

    gtif = gdal.Translate('',
        in_tif,
        GCPs=input_points,
        outputSRS = epsg_code,
        outputType = in_tif.GetRasterBand(1).DataType,
        noData=None,
        format='MEM')
    if not output_tif:
        return gdal.Warp('',
            gtif,
            format='MEM',
            srcSRS=epsg_code,
            dstSRS=epsg_code,
            tps=use_thinplate,
            errorThreshold=errorThreshold)
    else:
        if os.path.isfile(output_tif):
            os.remove(output_tif)
        gdal.Warp(output_tif, gtif,
            format='GTiff',
            srcSRS=epsg_code,
            dstSRS=epsg_code,
            tps=use_thinplate,
            errorThreshold=errorThreshold)

def remove_layers(layer_names):
    layers = [QgsProject.instance().mapLayersByName(layer_name) for layer_name in layer_names]
    if layers:
        QgsProject.instance().removeMapLayers([layer.id() for layer_list in layers
            for layer in layer_list])

def load_image_QGIS(file_name, layer_name, layer_range, gray=True):
    rasterLyr = QgsRasterLayer(file_name, layer_name)
    if gray:
        myEnhancement = QgsContrastEnhancement(rasterLyr.renderer().dataType(
            rasterLyr.renderer().grayBand()))
        myEnhancement.setContrastEnhancementAlgorithm(
            QgsContrastEnhancement.StretchAndClipToMinimumMaximum, True)
        myEnhancement.setMinimumValue(float('{:.2f}'.format(layer_range[0])))
        myEnhancement.setMaximumValue(float('{:.2f}'.format(layer_range[1])))
        rasterLyr.renderer().setContrastEnhancement(
            myEnhancement)
```

```
    else:
        import colorsys
        n_colors = 10
        hs = np.linspace(0,2/3,n_colors)
        colors = ['#' + hex(int(r*255)).split('x')[-1] + hex(int(g*255)).split('x')[-1] +
                  hex(int(b*255)).split('x')[-1] for (r,g,b) in (colorsys.hls_to_rgb(h,0.5,0.8)
                  for h in hs)]
        # colors = ['#e61919', '#e6a219', '#a2e619', '#19e619', '#19e6a1', '#19a1e6', '#1919e6']
        values = np.linspace(layer_range[0], layer_range[1], len(colors))
        lst = [QgsColorRampShader.ColorRampItem(value, QColor(color), '{:.2f}'.format(value))
            for (value,color) in zip(values,colors)]

        myRasterShader = QgsRasterShader()
        myColorRamp = QgsColorRampShader()

        myColorRamp.setColorRampItemList(lst)
        myColorRamp.setColorRampType(QgsColorRampShader.Interpolated)
        myColorRamp.setClip(True)
        myRasterShader.setRasterShaderFunction(myColorRamp)

        myPseudoRenderer = QgsSingleBandPseudoColorRenderer(rasterLyr.dataProvider(),
            rasterLyr.type(), myRasterShader)
        myPseudoRenderer.setClassificationMin(layer_range[0])
        myPseudoRenderer.setClassificationMax(layer_range[1])
        rasterLyr.setRenderer(myPseudoRenderer)

    rasterLyr.isValid()
    QgsProject.instance().addMapLayers([rasterLyr])

def load_multiband_image_QGIS(file_name, layer_name):
    rasterLyr = QgsRasterLayer(file_name, layer_name)
    myEnhancement = QgsContrastEnhancement()
    myEnhancement.setContrastEnhancementAlgorithm(
        QgsContrastEnhancement.ClipToMinimumMaximum, True)
    myEnhancement.setMinimumValue(1)
    rasterLyr.renderer().setRedContrastEnhancement(myEnhancement)
    rasterLyr.renderer().setGreenContrastEnhancement(myEnhancement)
    rasterLyr.renderer().setBlueContrastEnhancement(myEnhancement)

    rasterLyr.isValid()
    QgsProject.instance().addMapLayers([rasterLyr])


# Definidas todas as classes e funções, vamos rodar o plugin
class ExAlgo(QgsProcessingAlgorithm):
    INPUT = 'INPUT'
    OUTPUT = 'OUTPUT'

    def __init__(self):
        super().__init__()

    def name(self):
        return 'plugin_simrad'

    def tr(self, text):
        return QCoreApplication.translate('plugin_simrad', text)

    def displayName(self):
        return self.tr('Plugin_SIMRAD')

    def group(self):
        return self.tr('Plugin_TCC')

    def groupId(self):
```

```
        return 'plugin_tcc'

    def shortHelpString(self):
        return self.tr('Plugin para quantização espacial de dados de batimetria e ' +
            'retroespalhamento de piso marinho. Desenvolvido pelos alunos Haroldo Júnio ' +
            'dos Santos e Bruno Mota de Souza com a orientação dos professores Luciano ' +
            'Emídio Neves da Fonseca e Diogo Caetano Garcia da FGA/UnB.')

    def helpUrl(self):
        return ''

    def createInstance(self):
        return type(self)()

    def initAlgorithm(self, config=None):
        self.input = 'INPUT'

    def processAlgorithm(self, parameters, context, feedback):
        buttons = Buttons()
        buttons.show()
        return {self.OUTPUT: 'ok'}
```