UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

APLICACIÓN DE CALIBRACIÓN Y MONITOREO DE SIMULACIÓN GEOMÉTRICA DEL PROCESO DE EXTRACCIÓN MINERO A CIELO ABIERTO.

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

PABLO FRANCISCO SANCHEZ REBOLLEDO

PROFESORA GUÍA:
NANCY HITSCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
MAURICIO PALMA LIZANA

SANTIAGO DE CHILE
2024

Resumen

En la minería a cielo abierto es de gran importancia tener un entendimiento del estado actual de la superficie terrestre mientras se extrae el material. Es parte de la labor de varios equipos analizar la operación minera y planificar anual, mensual, semanal y diariamente el material extraído.

TIMining es una empresa que trabaja haciendo herramientas y software con tecnología a la vanguardia capaz de mejorar el entendimiento para el personal encargado de la planificación segura y óptima de distintos tipos de material. El software más reciente que ha desarrollado es *TIMA*, un gemelo digital capaz de reportar lo que ocurre en faena a tiempo real.

Durante la instalación de dicho software, se utiliza una API llamada *MineTiler* que hace una simulación de la realidad más actualizada de lo que actualmente es posible. Esta API hace uso de los datos de la operación minera para tener un fiel modelo de este. Sin embargo, al momento de su instalación, se necesita calibrar la herramienta para que reporte correctamente el estado actual de la topografía.

En el presente tema de memoria se aborda el desafío mencionado desarrollando una aplicación que permite responder a dos problemáticas actuales en el proceso de análisis y calibración de la visualización en *TIMA*. Primero se automatizan los procesos manuales necesarios para simular la ejecución de la API. Y segundo, una vez automatizado el proceso, se desarrolla una visualización comparativa de los sectores específicos de interés para el usuario y cómo varía el error con respecto a lo esperado utilizando distintas configuraciones de *MineTiler*.

Al concluir la investigación, se tiene una clara línea de trabajo para utilizar las visualizaciones de indicadores de error que se buscaban para entender lo que se ve en *TIMA*. Además de cómo se pueden mejorar las visualizaciones geométricas de la topografía para el estudio de lo que se ve realmente en el gemelo digital dado los puntos de referencia que el usuario ingresa diariamente.

Le dedico este trabajo a mi padre por mis manos,
a mi madre por mi mente
y a mi perrita Venus por mi corazón.

Tabla de Contenido

Índice de Ilustraciones

Capítulo 1

Introducción

En la minería a cielo abierto la obtención y análisis de la superficie es crucial para la planificación de extracción del material. Para este fin, se han implementado varias estrategias para obtener la realidad actualizada de la mina. Como por ejemplo medidas de altura por medio de drones o generación de una triangulación que representa la superficie actualizada a través de fotografías tomadas desde distintos puntos en faena.

La planificación de extracción se representa en distintos formatos como una geometría 3D. Existe un formato en el cual se muestra la superficie completa de la mina quitando las áreas que deben ser extraídas y también aquel donde se generan sólidos 3D que representan el volumen a extraer. Se genera esta información para planificar la extracción del año, del mes, de la semana y en algunos casos, del día.

Otra fuente de datos de cliente viene del control de flota, que genera información geoespacial de las máquinas en faena, como lo son las palas, las excavadoras y los camiones.

Dentro de esta industria, *TIMining* es una empresa de tecnología e innovación en minería que desarrolla software de simulación y análisis de datos para ayudar a la toma de decisiones en la operación minera a cielo abierto. Uno de los productos que provee es un gemelo digital llamado *TIMining Aware* o *TIMA* que busca entregar entendimiento a tiempo real de lo que está ocurriendo en faena [25].

Uno de los problemas que enfrenta TIMA es la simulación geométrica de la realidad actualizada más frecuentemente de lo que el usuario es capaz de obtener la topografía real. El gemelo digital simula cada 15 minutos la visualización de los avances en los distintos sectores de extracción de material. Esto se logra en conjunto con información geoespacial de una API de datos GPS llamada *Load-Haul Server (LHS)* de *TIMining*, encargada de disponibilizar y estandarizar los datos de control de flota en faena. Dicha frecuencia de simulación es considerada tiempo real en el actual estado del arte en minería [2]. Tomando en cuenta también la planificación de extracción del usuario, se calculan las diferencias en términos de volúmenes y se compara con la planificación [5]. Es tema de discusión cómo medir, validar y evaluar la realidad topográfica que debería estar mostrando TIMA. No existen indicadores para garantizar que la visualización del software sea la esperada para entregarle valor al usuario.

La estrategia utilizada para representar esta realidad es el objeto de estudio principal de *MineTiler*. Una API capaz de recibir archivos topográficos de la mina y, junto con información geoespacial, que puede ser imprecisa o tener ruido, genera una nueva simulación de la topografía. *TIMA* funciona haciendo llamados de manera periódica a la API para cargar la topografía generada y desplegarla en el software. Existe una serie de parámetros para ajustar los datos GPS que finalmente considera el algoritmo. Por ejemplo, uno de los parámetros es un percentil que se le aplica a la altitud de los datos GPS de las palas. Además, es posible aplicar filtros a los datos mediante las consultas a LHS, como por ejemplo, filtro por rapidez o por estado actual de las palas.

El desafío actual de *MineTiler* es la calibración y depuración de artefactos o deformaciones en la topografía simulada que no corresponden a una buena representación de la realidad. El algoritmo principal de la API hace una corrección de la topografía utilizando datos sujetos a ruido, desplazamiento y otros errores en la información debido a imprecisiones en las herramientas de medida en faena.

Para ejecutar este algoritmo, debido al uso de la topografía real del cliente y la información de *LHS*, se requiere configurar *MineTiler* y cargar datos del usuario. Una vez hecho esto, el hacer ejecuciones, deja un registro histórico de los sectores que han cambiado, por lo tanto, para no afectar las pruebas futuras por pruebas ya hechas, se deben limpiar estos registros. Los procesos descritos son ejecutados manualmente y en la práctica se han cometido errores donde el no eliminar estos registros afecta los resultados de las consultas, quedando inválidas como pruebas para la calibración.

Como principal motivación de la aplicación propuesta, se pretende agilizar el proceso de calibración automatizando pasos manuales necesarios para generar una geometría actualizada. Así, se facilita la iteración, evaluación y análisis de los resultados. Uno de los planes a futuro del software es automatizar este proceso lo más posible.

1.1. Objetivos

1.1.1. Objetivo General

El objetivo es agilizar el proceso de análisis y calibración de los algoritmos responsables de la simulación geométrica de la faena.

1.1.2. Objetivos Específicos

1. Identificar las necesidades del equipo a cargo de la instalación de MineTiler al momento de depurar, probar y comparar los resultados de la API.
2. Automatizar el proceso de preparación de la API para generar una simulación de la realidad.
3. Generar datos sintéticos topográficos y un mock de *LHS* que responda a las consultas de datos geoespaciales.

4. Comparar geometrías y utilizar indicadores que faciliten el análisis de las topografías resultantes en comparación con la realidad.

5. Generar una vista comparativa de los distintos resultados de las automatizaciones tomando en cuenta los indicadores definidos en el ítem anterior.

1.2. Descripción general de la solución

La solución propuesta provee métodos para hacer uso de *MineTiler* de forma local con la lógica para repetir las consultas dependiendo tanto de los parámetros de configuración como de los valores que escoge el usuario. Luego, recopilando las topografías resultantes, se comparan contra una topografía esperada por cada configuración. Además, se compara el caso de una consulta cada 15 minutos tomando como base la topografía generada más recientemente.

Para esto, se desarrollaron dos *scripts* en conjunto. Uno de estos es **minetiler_automation.py** que automatiza el proceso manual de configurar MineTiler y luego hacerle consultas. Y el otro es **compare_dtms.py**, que recopila las topografías resultantes para compararlas contra una topografía esperada ingresada por el usuario. Esta comparación consiste en el cálculo del error *RMSE* y *MAPE* y la generación de mallas geométricas para identificar las diferencias de manera visual.

El resultado final son dos tipos de visualizaciones, las triangulaciones coloreadas y los gráficos de línea de métricas de error, para evaluar topografía predicha por *MineTiler* para distintos valores de cada parámetro de configuración.

1.3. Estructura de la memoria

La estructura de la memoria por capítulo es la siguiente:

Capítulo 2: Antecedentes Se explican una serie de conceptos base incluyendo *MineTiler* y los formatos geométricos que se utilizan. Además, se hace un análisis del estado del arte en los temas relevantes en este documento.

Capítulo 3: Problema Describe la situación actual al momento de calibrar el algoritmo de fresado, destacando la relevancia del proceso, las limitaciones actuales y los beneficios de mejorarlo.

Capítulo 4: Solución Se describe el desarrollo realizado en 3 partes. Primero, la automatización de los pasos manuales del proceso de calibración. Segundo, que consideraciones se tuvieron al momento de generar los datos sintéticos. Y finalmente, el procesamiento de las respuestas hechas por el automatizado para ser comparadas contra un caso esperado y ser visualizadas.

Capítulo 5: Resultados Se definen los datos GPS, una serie de parámetros de configuración y sus rangos para hacer ejecuciones del automatizado. Luego se analizan las distintas herramientas de visualización disponibles en el *script* de comparación.

Capítulo 6: Conclusión Finalmente, se concluye el trabajo realizado volviendo al problema original y evaluando cuánto aporta la solución al proceso de calibración del algoritmo de fresado, sus deficiencias y cómo podría evolucionar no tanto solo esta solución, pero el problema en general.

Capítulo 2

Antecedentes

En el presente capítulo se introducen principios estadísticos, conceptos geométricos de interés y una descripción general de dos componentes de *TIMA* relevantes para el trabajo realizado. Primero se describen las estrategias estadísticas frecuentemente utilizadas para medir el desempeño de un modelo predictivo. Luego se ilustran los detalles técnicos de los formatos con los que *MineTiler* API trabaja la superficie de la mina.

Además, se presenta la investigación acerca del actual estado del arte en minería, incluyendo su evolución a través de los distintos hitos tecnológicos a lo largo su desarrollo. Se describe como dichos hitos han mostrado resultados prometedores en la industria.

Se concluye mencionando las herramientas utilizadas al comparar los formatos *DTM*. En concreto, cómo se ocupan los principios estadísticos mencionados en conjunto con estrategias visuales en la discretización del espacio por grilla.

2.1. Conceptos Base

2.1.1. Métricas de error para un modelo predictor

Root Mean Squared Error o *RMSE* es la raíz cuadrada de la media de la diferencia entre los valores predichos y los valores actuales al cuadrado. Una de las propiedades más importantes de esta métrica es la facilidad de su interpretación. La escala de *RMSE* es la misma que los datos observados, en nuestro caso, metros. Una segunda propiedad de interés es que debido al cuadrado del error, este indicador tiende a exponer grandes errores en las diferencias. Esta cualidad es de gran importancia cuando un error de gran magnitud en la predicción puede deformar la topografía simulada, generando desconfianza en el cliente que ya no puede depender de la vista principal en 3D de *TIMA*.

MAPE es el equivalente en porcentajes absolutos de *RMSE*, sin embargo, diverge para valores Z reales cercanos a 0 ya que se divide el valor Z esperado. Debido a que el algoritmo de fresado y *TIMA* están bien definidos para valores $Z = 0$ se necesita una alternativa para

este dominio. Una de estas alternativas es *sMAPE* o *Symmetrical Mean Absolute Percentage Error*. Este garantiza un límite superior al error resultante de 100 % o 200 % dependiendo de la implementación. Sin embargo, *sMAPE* tampoco está definido para cuando ambos Z_i y $\hat{Z}_i$ son 0. Por lo tanto, se utiliza *MASE* o *Mean Absolute Scaled Error* diseñado especialmente para ser usado cuando el dominio admite valores 0.

Cabe destacar que estos casos borde son poco frecuentes en la práctica aunque no imposibles. La indefinición de *MAPE* como *sMAPE*, dependerá estrictamente de alguna faena de cliente que trabaje con topografías con valores de altura cercanos al 0. A pesar de ello, podrían generar errores en la ejecución de la aplicación con una sola ocurrencia de estos casos.

MASE comparte los mismos beneficios de *RMSE* pero entrega un distinto entendimiento al ser independiente de la escala de los datos a medir, siendo un mejor indicador entre faenas y/o clientes distintos.

MSE o *Mean Scaled Error* es una métrica más simple y es el promedio de la diferencia entre lo real y lo simulado al cuadrado. La cualidad de interés en este indicador es que castiga aún más errores de mayor magnitud, dando gran opción de ser una alerta automatizada y rápida si es que el valor se dispara.

Casos como este han ocurrido cuando los datos de las palas, por alguna razón, cambian repentinamente y empiezan a fallar alterando profundamente la vista de *TIMA*. La escala es al cuadrado, por lo tanto los números no son tan simples de entender como *RMSE* o *MASE*.

MAE o *Mean Absolute Error* es una métrica más simple y es la diferencia absoluta promedio entre los valores esperados y los predichos.

Finalmente está *R2* o *R squared* y es un coeficiente que describe cual es la proporción de la variación proveniente del modelo predictivo. En el caso de este trabajo, es la proporción de la variación de valores Z explicada por el cambio de parámetro dada una configuración.

A continuación se definen las distintas fórmulas de los 4 indicadores de error mencionados.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(Z_i - \hat{Z}_i)^2} \tag{2.1}$$

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(Z_i - \hat{Z}_i)^2 \tag{2.2}$$

$$MAE = \frac{1}{n}\Sigma_{i=1}^{n}|Z_i - \hat{Z}_i| \tag{2.3}$$

$$MASE = \frac{n-1}{n}\frac{\Sigma_{i=1}^{n}|Z_i - \hat{Z}_i|}{\Sigma_{i=2}^{n}|Z_i - Z_{i-1}|} \tag{2.4}$$

2.1.2. Terminología y conceptos mineros

Se describe terminología en la industria minera para distintos elementos y prácticas comunes entre faenas a cielo abierto que serán mencionadas en las demás secciones.

Rajo: Espacio abierto en la superficie terrestre de un yacimiento mineral en proceso de ser extraído. Varios elementos que lo componen se muestran en la figura 2.1a.

Faena: Se entenderá por faena las instalaciones cerca del rajo donde se llevan a cabo otras etapas del trabajo con el mineral. El gemelo digital *TIMA* está diseñado para replicar una faena.

Banco: Cortes escalonados del yacimiento a rajo abierto. Se considera un banco de altura de 15 metros para los ejemplos de este trabajo. La figura 2.1b muestra las partes de un banco.

Palas: Máquina excavadora especializada en extraer material y cargarlo a camiones de transporte. La pala alimenta a las bases de datos en faena con datos de su operación. Por ejemplo, posición de la pala, posición del balde, estado según el operador, entre otros.

Carguío: Proceso de extracción del material donde la pala excavadora carga los camiones de transporte con el mineral.

2.1.3. Topografías

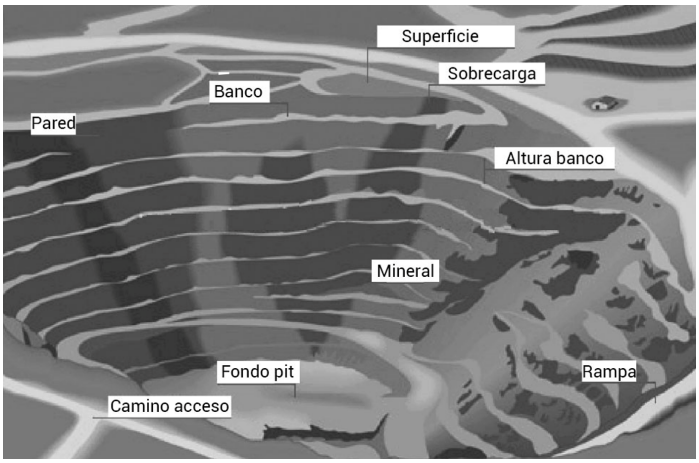
La topografía es un dato proveniente del usuario en faena que representa la superficie terrestre en una malla geométrica 3D. A medida que se va extrayendo material, se hace un muestreo de la superficie diariamente para tener su estado actualizado.

Este dato puede venir en distintos formatos, sin embargo, el más común utilizado por *MineTiler* son triangulaciones en formato OBJ. En estos archivos las mallas son representadas por triángulos, cuya definición son los 3 vértices que lo conforman, como se muestra en la figura 2.2. Por lo tanto, en el archivo también se define cada vértice y su posición en x, y, z.

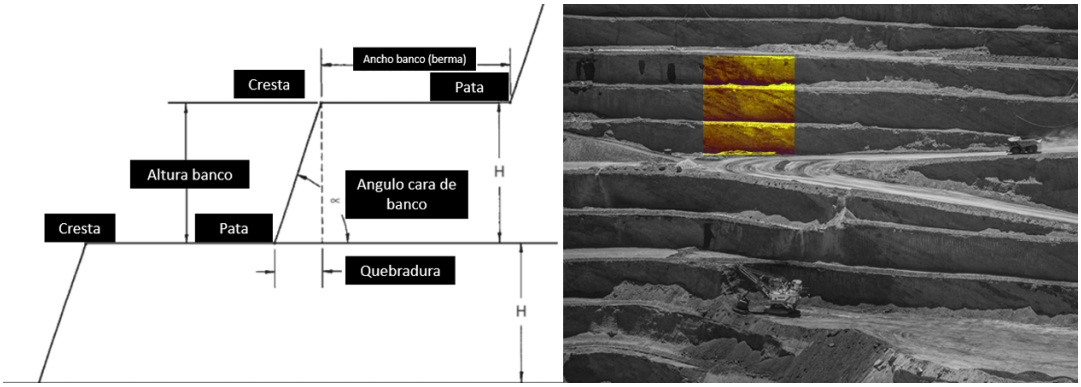
2.1.4. Baldosado

Al momento de cargar una topografía a *MineTiler*, pasa por un procesamiento llamado *baldosado*. En este, se subdivide la superficie terrestre en una grilla de **baldosas** o *tiles* de tamaño configurable que comienza en el origen del plano cartesiano. A cada *tile* le corresponde un archivo *Digital Terrain Model* o *DTM*.

Un *DTM* discretiza el espacio 3D guardando en una grilla de celdas la elevación de una posición X e Y. Este formato requiere ciertas constantes como cabezales para su funcionamiento como se ve en la figura 2.4a y son parte de la configuración de *MineTiler*. En la siguiente lista se definen las propiedades del archivo que se comparten para todas las baldosas:



(a) Elementos de un rajo a cielo abierto.



(b) Elementos de un banco.

Figura 2.1: Distintos elementos de un rajo a cielo abierto y sus bancos. Fuente: Nube Minera, `https://nubeminera.cl/curso/elementos-rajo/`.

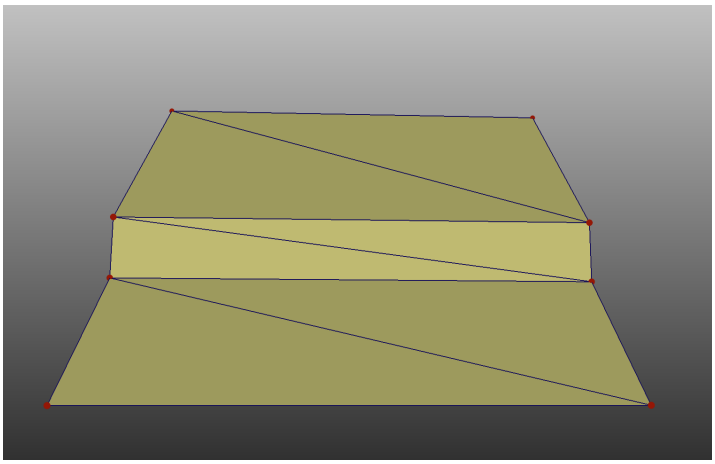


Figura 2.2: Triangulación en OBJ mostrando sus triangulos y vértices.

1. Número de filas *(NROWS)*. ej: 1024
2. Número de columnas *(NCOLS)*. ej: 1024
3. Valor **X** de esquina inferior izquierda *(XLLCORNER)*. ej: 2048
4. Valor **Y** de esquina inferior izquierda *(YLLCORNER)*. ej: 2048
5. Tamaño de celda en metros *(CELLSIZE)*. ej: 0.5
6. Valor nulo *(NODATA_VALUE)*. por defecto: -999999

El largo y ancho de las baldosas está definido por el parámetro configurable *tileSize* y es el mismo que el número de filas y columnas del *DTM* que las componen como se muestra en la figura 2.3.

Al momento de cargar una topografía a *MineTiler*, usualmente una triangulación en formato *OBJ*, se ignoran las baldosas que no interaccionan con la triangulación cargada. como se muestra en la figura 2.4b.

La mayoría de los procesamientos que hace *MineTiler* se hacen utilizando *DTMs*. Por lo tanto, para evaluar el algoritmo de fresado, los métodos para comparar los resultados, también usan *DTMs* y sus operadores.

2.1.5. MineTiler API

La arquitectura de *MineTiler* se muestra en la figura 2.5. El servidor disponibiliza los endpoints de la API encolando trabajos en una base de datos. Actualmente se utiliza una de estas colas para procesar cargas de datos del usuario y otra para procesar los distintos cálculos

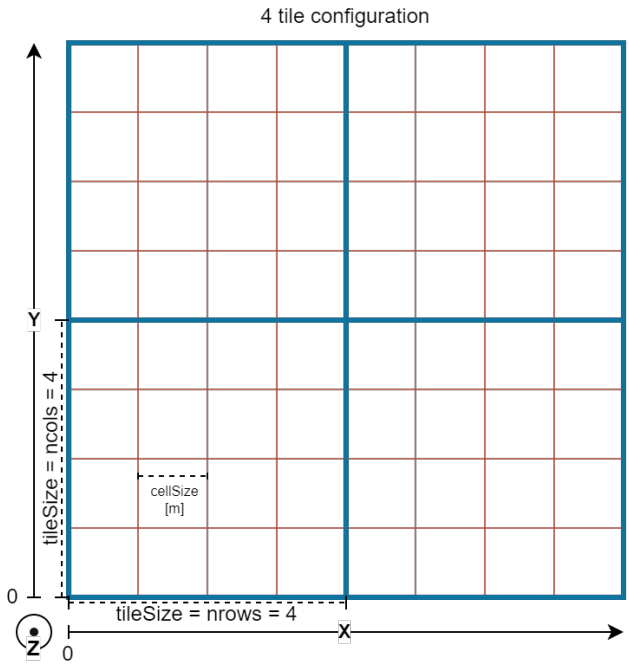
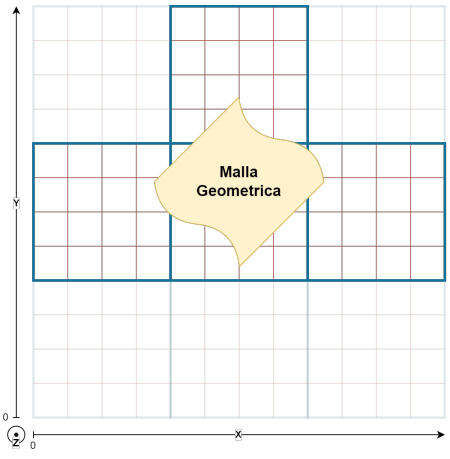


Figura 2.3: Figura mostrando la relación entre las baldosas, su tamaño, los *DTM* que la definen y el tamaño de celda.

```
NCOLS 10
NROWS 10
XLLCORNER 0
YLLCORNER 0
CELLSIZE 1
NODATA_VALUE -999999
10 10 10 10 10 10 10 10 10 10
10 10 10 15 15 15 15 10 10 10
10 10 15 20 20 20 15 15 10 10
10 10 10 15 20 20 20 15 10 10
10 10 10 10 15 20 20 15 10 10
10 10 10 10 10 15 15 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
```

(a) Cabezales y primera filas de un *DTM*.



(b) Representación gráfica de los *tiles* de una malla geométrica.

Figura 2.4: Formatos de archivos topográficos intermedios.

necesarios para la simulación de la faena. Los datos de usuario que pueden ser cargados son ortofotos de la mina, fotografías reales, planificación en formato de mallas, sólidos o polígonos y modelos de bloque.

El módulo definido como *Worker*, es una serie de *pipelines* que crea el servidor. Estos *pipelines*, si no son de carga de archivos, descargan archivos de entrada, consultan los datos GPS desde *LHS* y luego corren ejecutables que finalmente hacen los cálculos, procesamiento y simulación de la topografía. Una vez terminado, se cargan a las bases de datos y se publican los resultados.

Recientemente se desarrolló una opción de ejecución del algoritmo que genera la topografía de manera local, llamado fresado, separada de la API de *MineTiler* y sus bases de datos. Esto responde a la necesidad de facilitar el proceso de calibración cuando se requiere hacer comparaciones con los mismos datos de entrada. Estos datos involucran topografías de inicio y información geoespacial de máquinas en faena proveniente de otro endpoint que provee parte de la infraestructura de *TIMA*. Tal como la API de *MineTiler*, el algoritmo local está desarrollado en GoLang y hace uso de calculadores de procesamiento en Qt y C++.

La salida por parte de las consultas son rutas a objetos en *MinIO*, un sistema de *object storage* en formato *DTM (Digital Terrain Model)* [26] binarios divididos en baldosas de la topografía completa, o en archivos de diferencias de altura en formato PNG en escala de grises. Por parte de la ejecución local de los algoritmos se obtienen datos GPS en formato JSON o una conexión a una API y la salida son mallas geométricas en formato OBJ.

2.1.6. Load-Haul Server (LHS)

LHS es otra API de *TIMining* que se conecta directamente con el control de flota en faena. Este sistema extrae y traduce información de los sistemas de control de máquinas en el rajo de la mina. Dichas máquinas incluyen camiones, palas, perforadoras y excavadoras. La información que se le pide a esta herramienta en específico son las palas registradas y luego los datos GPS de esas palas en una ventana de tiempo configurable.

El control y manejo de *LHS* es específico de un equipo distinto al de MineTiler en TIMining y requiere de más esfuerzo configurar un ambiente real de LHS con datos GPS sintéticos. Por lo tanto, se desarrolló en *Python* un script que responde a las mismas rutas que consulta MineTiler pero con datos fijos utilizando *FastAPI*. Es con estos datos GPS que se genera un fresado en la topografía.

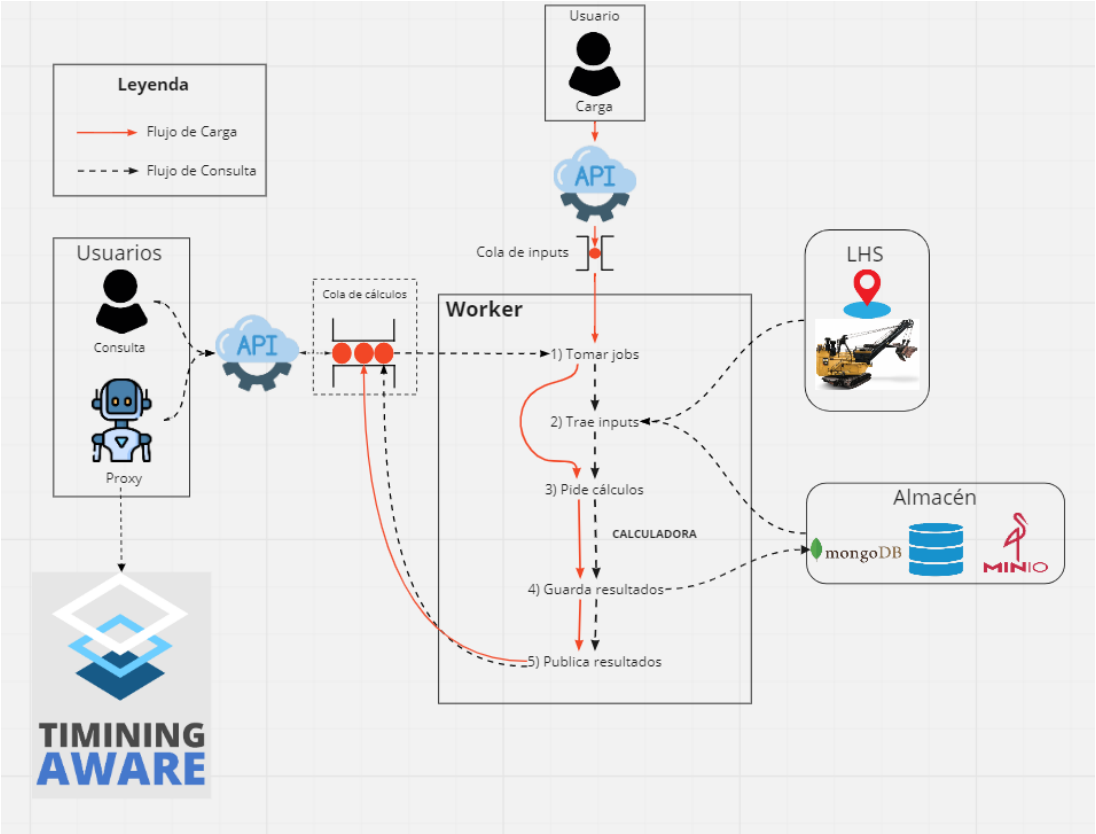


Figura 2.5: Arquitectura de *MineTiler*.

2.2. Estado del Arte

2.2.1. Minería 4.0

Históricamente, tanto en la minería como en otras industrias, por ejemplo, manufactura, automotriz o salud, se busca utilizar las tecnologías a la vanguardia para mejorar la cadena de valor. En la actualidad distintos sectores industriales están en un proceso de transición de la industria 3.0 a la industria 4.0 utilizando tecnologías como la inteligencia artificial, el procesamiento masivo de datos, drones, etcétera [19].

Como breve reseña, la evolución de la industria es reconocida de la siguiente manera. La industria 1.0 surge del comienzo de la maquinización durante el uso de motores de vapor. El siguiente paso tecnológico importante fue la incorporación de la electricidad a los procesos industriales y de manufactura, dando paso a la industria 2.0. Luego, durante las primeras décadas de la computación, se comienza la automatización mediante la robótica, dando paso a la industria 3.0, aquella vigente y desafiada actualmente por la industria 4.0. Específicamente involucra la aplicación de *Industrial Internet of Things (IIoT)*, Inteligencia artificial, *Big Data*, Gemelos Digitales, entre otros [27, 11].

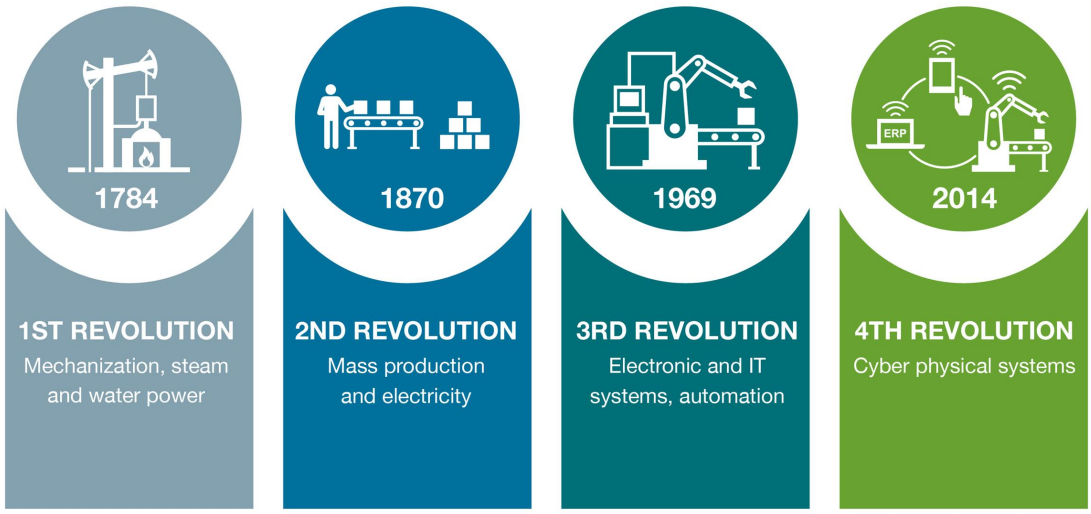


Figura 2.6: Evolución de la Industria 1.0 a la Industria 4.0. Fuente: Bismarck State College, `https://bismarckstate.edu/polytechnic/What-is-a-Polytechnic/Industry-40/`.

La industria de los recursos naturales ha estado presente en cada una de estas revoluciones industriales, y se ha mantenido a la par innovando para competir en sostenibilidad, seguridad y productividad. En Chile y Peru, como grandes exportadores de cobre, se pone especial

atención en impulsar la innovación en la industria para aumentar la productividad al tener acceso a un gran porcentaje de los depósitos disponibles en el mundo [13].

Específicamente, el proceso de extracción minero se ha vuelto más complejo a lo largo de los años debido a exigencias de seguridad, escasez de recursos naturales de fácil acceso y material con bajo grado de mineral [7]. Por lo tanto, para mantenerse al día con el estado del arte en la industria, el sector minero ha buscado activamente el uso que puedan tener estos avances tecnológicos en su cadena de valor.

Cabe destacar que es tema de investigación los factores que frenan la incorporación de nuevas tecnologías en el sector minero. Uno de estos es la preocupación del personal de la operación minera por el peligro que pueden correr sus lugares de trabajo debido a la innovación industrial en curso [3, 8].

2.2.2. Gemelos Digitales

Un gemelo digital es una representación virtual generada tomando información extraída de elementos físicos reales que componen la cadena productiva, generando una réplica virtual de los procesos industriales. Esta tecnología nace gracias a, y en conjunto con los avances en capacidad de procesamiento de datos, la mejora de los sensores y computación en la nube. Utilizando los distintos datos provenientes de los elementos físicos, se crea un modelo virtual como una réplica de estos elementos en conjunto [10].

Esta herramienta alimentada con la suficiente información, abre varias oportunidades. Por un lado puede ser utilizada en tiempo presente en la toma de decisiones de operaciones físicas complejas. También, con una representación virtual se tiene registro del historial de actividades en tiempo pasado y se puede recrear el proceso productivo en una ventana temporal específica. Y finalmente, se puede hacer un prototipado de la cadena de valor a tiempo futuro, haciendo modificaciones al modelo y viendo su efecto en el resto de los elementos de la realidad [23].

Debido al amplio uso de sensores desde la industria 3.0, la mayoría de los sectores industriales poseen gran cantidad de datos a lo largo de su proceso productivo. Estos datos varían dependiendo del sector, por ejemplo, el área de salud tiene un gran historial digitalizado de los pacientes [1], y las empresas de manufactura tienen múltiples sensores en cada una de las máquinas que generan sus productos finales [14].

En concreto, dentro del sector minero, toda la línea productiva tiene elementos físicos que juegan un importante rol en su productividad. Estas son, palas, camiones, cintas transportadoras, máquinas chancadoras y más, cada una con un registro de datos específicos a su actividad. Una de las líneas de innovación más evidentes para el usuario minero es la oportunidad que entrega tener la operación minera en un modelo virtual conectado directamente con la información de la realidad como así lo ofrecen los gemelos digitales [7, 18].

No obstante, uno de los desafíos más importantes en esta área, es la obtención de la geometría 3D del espacio en faena y cómo alimentar esta información al modelo virtual. Las topografías de la operación minera, sea subterránea o a cielo abierto, tienden a ser de gran

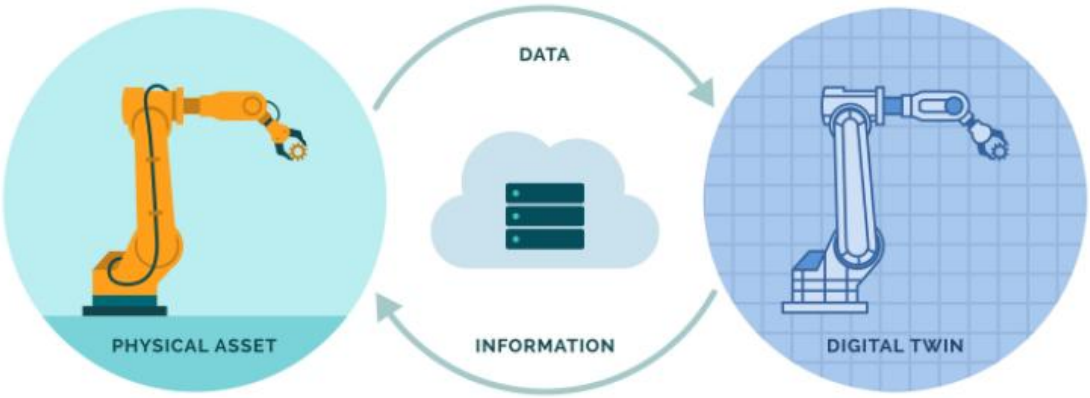


Figura 2.7: Principio fundamental de los gemelos digitales. Fuente: DataCenterKnowledge, `https://www.datacenterknowledge.com/security/how-secure-digital-twin-technology-your-data-center`.

tamaño y cambian cada hora mientras el material es extraído, mientras que la frecuencia de obtención de topografía es usualmente un día [4].

TIMA, como gemelo digital de la mina, resuelve este desafío incorporándose en el proceso productivo que involucra obtener la triangulación de la superficie y en conjunto con la demás información disponible, hace una simulación de la topografía. De esta manera, se acerca a otorgarle entendimiento de la actividad en faena al usuario minero con una frecuencia mayor a la que son capaces tanto con el muestreo láser, como con la fotometría posible gracias a los drones.

Es importante hacer hincapié en que *TIMA* aún depende de la obtención diaria de la topografía. Ya que es una simulación, a medida que pasa el tiempo, la diferencia entre la realidad y la simulación se hace evidente no tan solo en los cálculos volumétricos, si no también en el aspecto visual en la plataforma. Mediante la actualización de la topografía real, se corrige este error y se continúa con la simulación.

Es uno de los objetivos de *TIMA* acercarse cada vez más a la fuente real de los datos en su formato más crudo, mientras que actualmente trabaja principalmente con triangulaciones en formato .obj. Con este acercamiento, se podría alimentar los datos provenientes de los drones directamente al gemelo digital, enriqueciendo el ecosistema y su confiabilidad.

2.2.3. Drones

Uno de los temas más populares tanto en la industria como en la vida cotidiana durante la última década, ha sido la aparición de drones. Estos no tan solo han tenido un impacto como avance tecnológico sino que se les ha dado aplicación en áreas de la ciencia, la industria y la cultura.

El dron o también conocidos en la literatura como *Unmanned Aerial Vehicle (UAV)* es un artefacto volador controlado de manera remota o inteligente que tiene gran control, precisión y estabilidad en su vuelo. Específicamente, esta herramienta permite la rápida obtención de datos incorporando tecnologías como sensores de última generación, la nube y inteligencia artificial, especializando su vuelo al muestreo que esté realizando [22].

La flexible naturaleza del dron lo hace adaptable a distintos sensores diseñados para cualquier propósito. Una de las funciones relevantes usando sensores es el muestreo de la superficie terrestre. Debido a su precisión, capturan los datos necesarios para hacer una representación de la superficie a una triangulación o mapa de alturas como *Digital Elevation Model* [12].

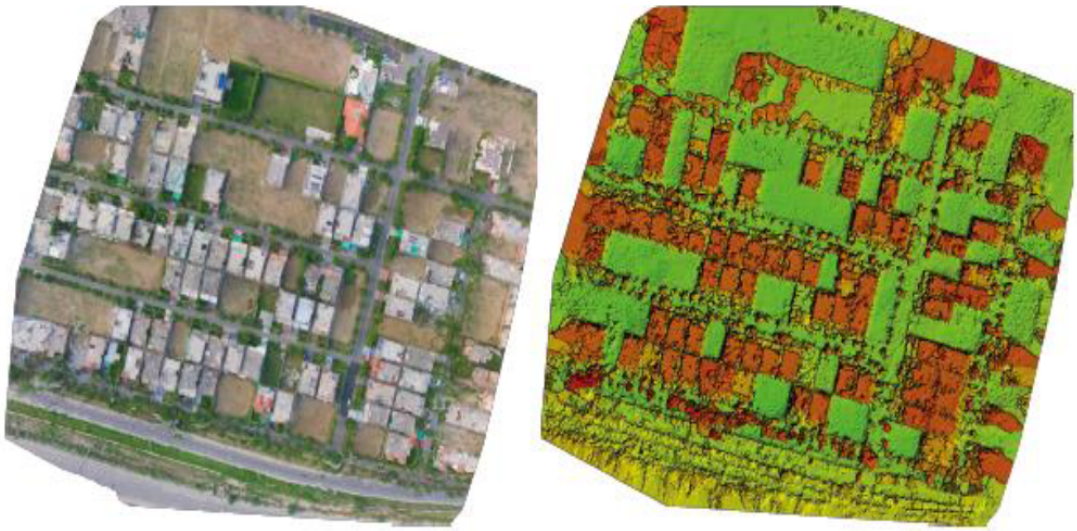


Figura 2.8: Ortofoto en mosaico y *Digital Surface Model* de fotometría resultante de muestreo utilizando *UAVs*, de Generation of Digital Surface Model (DSM) USING UAV/ QUADCOPTER [9].

Además, los drones han sido una de las primeras incorporaciones de la industria 4.0 a la minería. Metodologías como escaneo láser pueden hacer un muestreo a diario como es necesario en la industria, son de bajo costo y es una estrategia confiable [24]. Sin embargo, el dron da la posibilidad de conectar esta información a un ecosistema como el descrito en la sección de gemelos digitales.

Actualmente, hay abundante literatura sobre las aplicaciones y el uso de los drones en la industria, especialmente en el sector minero. El muestreo de la superficie terrestre en faena ayuda a la toma de decisiones diaria y el cumplimiento de la planificación de extracción, ambos claves en la cadena de valor de esta industria [12, 22, 4].

2.2.4. Comparación de *Digital Elevation Models*

Finalmente, alineada con la plataforma a desarrollar en este tema de memoria, se describen algunas alternativas en la literatura de cómo hacer un diagnóstico y comparación de la superficies terrestres generada por el gemelo digital. Esto orientado a demostrar de mejor manera, tanto cualitativamente como cuantitativamente, cuan fidedigno es el resultado de la simulación realizada por el gemelo digital.

Para el caso particular de *Digital Elevation Models*, la literatura describe, considerando la investigación del uso de *DTMs*, que la comparativa de la información topográfica depende de los requisitos del usuario [17, 21]. Sin embargo, se pueden utilizar métricas ya conocidas para detectar el error, como lo son MASE o RMSE.

RMSE o *Root mean squared error*, es una métrica utilizada para medir diferencias (error en nuestro caso) entre valores predichos por un modelo y los valores reales observados calculando su desviación estandar. Un valor 0 significa que las muestras se ajustan perfectamente al valor real observado.

MASE o *Mean absolute scaled error* es también una métrica para el mismo caso que *RMSE*, sin embargo, esta entrega una proporción indicando la precisión del modelo contra lo real. Lo útil de *MASE* es que es más intuitivo al identificar la escala de el error al ser un porcentaje.

En nuestro contexto, se comparan los valores de altura de los *DTM* predicho por el algoritmo de fresado comparado con la topografía original cargada por el usuario. Ambos son métricas de precisión de predicción que ayudarían a entender mejor cuánto se ajusta a la realidad la simulación de la topografía en *TIMA*

Existen aplicaciones especializadas en proveer varias comparaciones y métricas estándar, para el formato *DTM* que incluyen métricas estadísticas como las mencionadas [6].

Capítulo 3

Problema

A continuación se describe el problema que motiva este trabajo. Por una parte se explica la relevancia de una correcta simulación de la topografía para el cliente de *TIMA*, la situación actual de la calibración de *MineTiler* para su funcionamiento óptimo, los costos en los que se incurre y las herramientas que se utilizan al calibrar. También se incluyen las acciones a tomar para mejorar el proceso determinadas por el equipo encargado.

Finalmente, se describe el algoritmo de fresado de *MineTiler*, principal componente de estudio de esta memoria. Las estructuras de datos y estrategias de cálculo que se utilizan son tecnologías privadas. Sin embargo, se explicará la idea principal del algoritmo, los efectos de las configuraciones relevantes y a qué se debe su alta sensibilidad a la calidad de los datos.

3.1. Situación Actual

Hay características de la topografía que son de especial importancia para el usuario. Por ejemplo, las superficies planas por donde se mueve la pala y los frentes de extracción en los que trabaja. Ambos son sensibles a datos GPS erróneos o con exceso de ruido y una mala simulación de la superficie afecta el valor que le otorga la visualización principal en *TIMA*.

Durante la instalación de *TIMA* a un cliente, se hace un análisis de los datos del usuario y se calibran parámetros de configuración y filtros en *MineTiler* para tener la aproximación más cercana a la realidad. Durante este proceso, se debe iniciar la API, introducir suficientes entradas de usuario para generar una topografía y luego consultarla. La mayoría de las iteraciones en el proceso de calibración durante la instalación a un cliente han sido para analizar y asegurar la calidad de estas características de la geometría.

Repetir este proceso manualmente cada vez que se necesite procesar otra topografía con distintos parámetros de configuración para *MineTiler* en la práctica ha sido costoso. Esto incluye repetir las cargas de geometrías, la obtención y filtrado de datos GPS y finalmente la generación de la simulación. Hecho esto se utiliza otro proceso para generar una geometría en un formato más simple de comparar y visualizar.

Para el análisis de la realidad simulada se utilizan tanto visualizadores de mallas geométricas como software propio de la empresa para calcular las diferencias entre las topografías. Con este método de comparación se decide qué parámetros de configuración serían óptimos.

Existe un visor especializado para recibir respuestas de *MineTiler*, sin embargo, se utiliza como una herramienta de desarrollo. Su uso no es en directa comunicación con la API y requiere del ingreso manual de una respuesta. Este solo puede visualizar una malla geométrica a la vez.

La principal limitante de esta metodología es el costo del proceso manual de generar varias simulaciones utilizando *MineTiler* con distintas configuraciones. Además, la comparación y visualización de las mallas geométricas requiere pasos manuales adicionales.

Se realizó una retrospectiva del último proceso de calibración de *MineTiler* para el caso de un cliente específico y se decidieron tomar las siguientes acciones:

1. Construir el pipeline de los algoritmos que tienen incidencia en los datos.
2. Identificar un responsable de fidelidad del modelo 3D.
3. Iterar con el equipo de calibración información comparativa dentro de la visualización de relevancia para el proceso.
4. Buscar la forma de mostrar avances y resultados al cliente.

Por lo tanto, surge la necesidad de crear visualizaciones que faciliten ver los efectos del algoritmo de fresado en la superficie. Más aún, se requiere automatizar los pasos manuales para eliminar los procesos laboriosos.

3.2. Algoritmo de Fresado

El algoritmo de fresado es el módulo de *MineTiler* encargado de modificar la topografía en base a datos geoespaciales de palas durante el trabajo de extracción de material. La estrategia principal es tomar la superficie de la mina ingresada por el usuario y modificarla, haciendo una simulación del estado actual de la superficie. Dicha modificación se hace considerando que, si se encuentra un dato de pala por debajo de la superficie, quiere decir que el material que lo rodea ya ha sido extraído y la elevación real en la posición de la pala es menor en comparación con aquella en la triangulación original.

Existen constantes en la operación de extracción realizado por la pala. Por ejemplo, el radio dentro del que extraen material tiende a mantenerse por faena o incluso estar determinado por las especificaciones técnicas de la pala. Con esta información en mente se puede asumir que no solo la celda en donde está la pala tiene una elevación menor, sino un grupo de celdas dentro de un radio a su alrededor.

Ya que la elevación queda determinada por los datos de palas, es común que los datos geoespaciales tengan imprecisiones en su valor Z. Esto provoca estimaciones erróneas con

respecto a la altura que debería tener la superficie. Para corregir dichos errores se utilizan estrategias de filtraje y clustering para determinar que datos deberían afectar a que celda del *DTM* y la elevación real más probable dados dichos datos.

A continuación se describen los parámetros de configuración que se analizan en esta memoria. Estos son preprocesamientos a los datos de las palas y están en el orden en el que se aplican al iterar individualmente en las celdas del *DTM*.

Radio de extracción *(outerRadius)*: Al iterar en las filas y columnas de un *DTM*, se consideran solo los datos geoespaciales dentro de un radio centrado en la celda actual. Este dato comúnmente corresponde a la especificación técnica de la pala o experiencia minera de cuál es el radio en el que se trabaja extrayendo material.

Tolerancia en altura *(heightThreshold)*: Cualquier dato encontrado dentro del radio de la celda es descartado si su valor Z es mayor a la tolerancia en altura.

Mínimo de puntos en celda *(minPointsInCell)*: Una vez filtrados los datos de pala fuera del radio y por encima de la tolerancia de altura, el procesamiento de la celda continua si y sólo si la cantidad de datos a considerar es mayor al mínimo de puntos en celda. En el caso contrario, la celda no cambia su valor y se continúa con la siguiente celda del *DTM*.

Desplazamiento de extracción en altura *(extractionOffset)*: Existen casos particulares en donde las palas no extraen en la base del banco, pero por encima de este como se muestra en la figura 3.2. Para este caso, se configura un valor negativo para desplazamiento de extracción en altura que desplaza el valor Z de los datos de pala para compensar por la altura que tienen las palas por encima del banco. Este desplazamiento se configura por pala y aplica a todos los datos GPS en esta etapa del procesamiento.

Percentil *(percentile)*: El percentil P de una lista ordenada de forma ascendente de tamaño N es el menor valor de la lista tal que no más del porcentaje P de los valores sean estrictamente menores. Por ejemplo, el percentil 0 es el valor mínimo, 100 es el valor máximo, 50 es la mediana de la lista. Para una lista de 11 valores, cada valor representa el percentil en múltiplos de 10 como se ve en la tabla derecha de la figura 3.1. Existen varios métodos de cálculo si el percentil P cae entre dos valores de la lista, en el algoritmo de fresado se considera el proporcional lineal. Por ejemplo, si en la figura 3.1 se busca el percentil 35 de los datos, esto cae entre los valores 4 y 5, por lo tanto el resultado es 4,5. Para el algoritmo de fresado, el cálculo del percentil es el último procesamiento que determina el valor Z final que se le asigna a la celda y se repite a lo largo de la grilla.

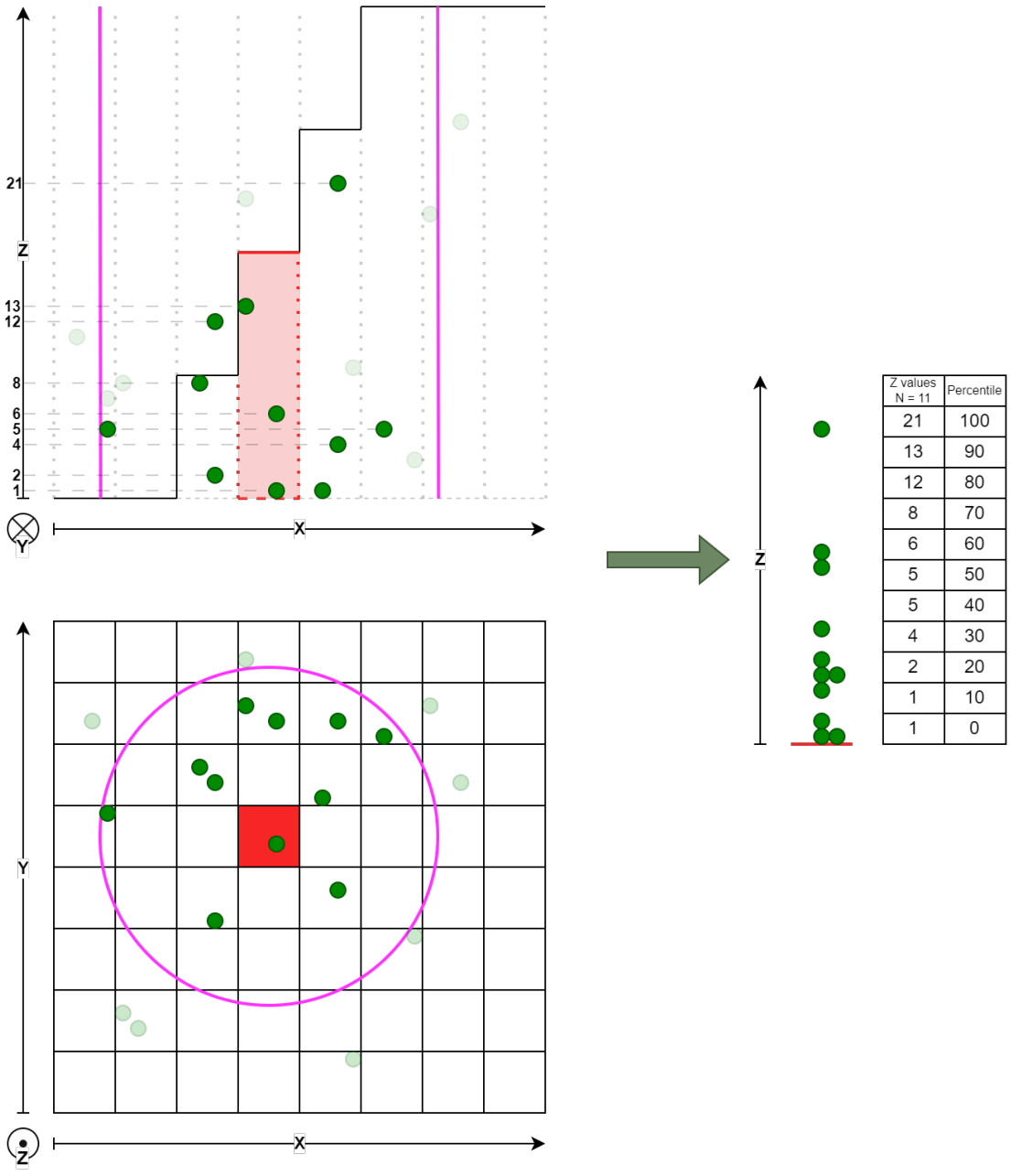


Figura 3.1: Ilustración del procesamiento de una celda individual en rojo del *DTM*. En la figura izquierda inferior está la grilla del *DTM* mirada desde arriba y en la figura izquierda superior la vista lateral. En púrpura está el radio de extracción, en verde datos GPS y en verde transparente los datos filtrados por el radio. En la figura izquierda superior están los valores en altura de los datos GPS y de las celdas del *DTM* en línea continua. En la derecha están los datos GPS finales y sus percentiles que determinan la altura final de la celda.

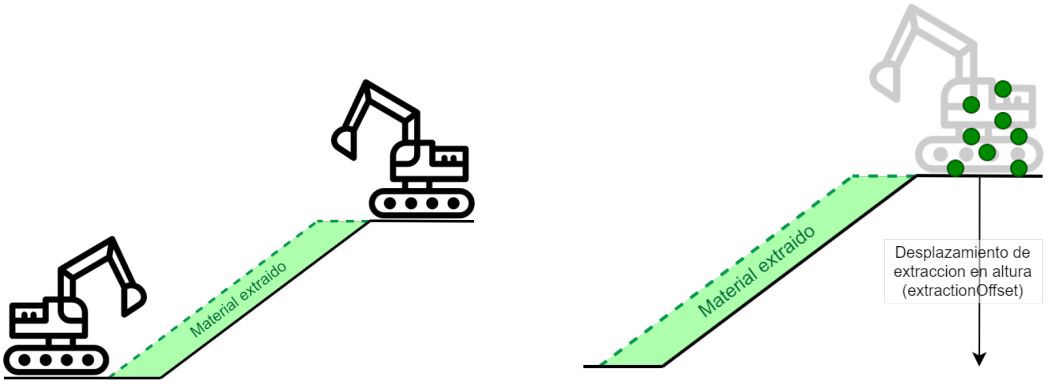


Figura 3.2: Ilustración de palas extrayendo desde la base del banco versus por encima del banco. El área verde es el material que se extrae y las palas están en la posición desde donde realizan la extracción. En la figura izquierda los círculos verdes son datos *GPS* de la pala y cómo se desplazan debido a la configuración *extractionOffset*.

Capítulo 4

Solución

En este capítulo se describen a detalle los aspectos técnicos de la solución. Primero se mostrarán las características de los datos de prueba a utilizar y cómo estos son generados. Luego, se explica la ejecución de *MineTiler* API y sus requisitos describiendo la ejecución de las consultas necesarias.

Luego, se ilustra el cálculo de los indicadores de error comparando los resultados generados por *MineTiler* mediante visualizaciones de datos en gráficos de línea.

Finalmente se describe la propuesta para comparar la simulación con lo esperado y poder con esta solución mejor entender el desempeño del algoritmo de fresado para distintas configuraciones y datos de entrada con distinta calidad.

4.1. Generación de datos sintéticos

Los datos geométricos y geoespaciales de palas utilizados por *TIMining* para el funcionamiento de *TIMA* están sujetos a contratos de privacidad. Por lo tanto, para evaluar el uso y la efectividad de la solución desarrollada, se utilizan datos sintéticos diseñados con características observadas en la práctica. Como por ejemplo, altura de bancos, radios de extracción esperados, ángulo de reposo de material y en magnitudes similares.

4.1.1. Datos geoespaciales de palas

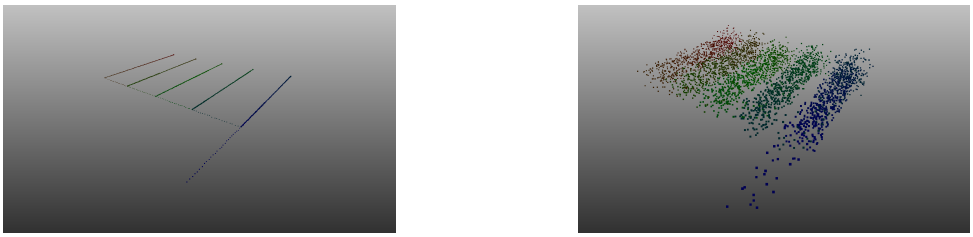
Los datos geoespaciales de palas que utiliza el algoritmo de fresado para simular la topografía se obtienen de la API de *LHS* en una ruta configurable. Por lo tanto y debido al uso de datos sintéticos, se ha desarrollado una herramienta generadora de datos de palas que simulan sus movimientos al momento de extraer material.

Al momento de configurar la aplicación, es posible elegir el servidor de *LHS* a utilizar. De

esta manera se vincula *MineTiler* a esta herramienta que responde a las consultas realizadas por la API con los datos generados. Dichos datos consisten de posición descrita en x, y, z, rapidez y fecha. Se utilizará una frecuencia de un dato GPS cada 15 segundos como lo es en palas con antenas de alta precisión.

Esta herramienta consiste de un solo archivo **gpsgenerator.py** y provee una serie de métodos simulando el movimiento de un objeto dada una nueva posición y su velocidad, recibiendo como parámetros una fecha inicial y frecuencia de datos. Estos son creados en el mismo formato que utiliza *LHS* para responder a la consulta que hace *MineTiler*, es decir, una respuesta JSON con una lista de pares vinculando el identificador de una pala a la lista de sus datos. También, para su visualización, se exportan los datos como una nube de vértices en un archivo OBJ coloreados en base a su orden cronológico.

Es posible agregar grados de aleatoriedad a los datos para mejor representar las imprecisiones en las lecturas de posición reales. Como una de las configuraciones del generador existe una tolerancia que determina la magnitud de esta funcionalidad. Se utilizaran 3 grupos de datos, datos ideales, datos de mala calidad y un punto medio representando un caso más cercano a la realidad.



(a) Datos de pala ideales.

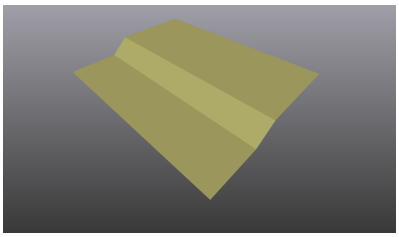
(b) Datos de pala imprecisos.

Figura 4.1: Datos sintéticos de palas.

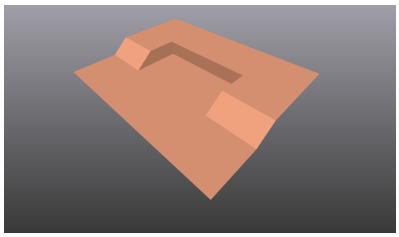
4.1.2. Datos topográficos

Para ser utilizado en conjunto con esta herramienta, se crearon mallas geométricas en formato OBJ que cumplen con las características de un banco en faena en proceso de extracción. Con una dimensión de 150 por 200 metros, altura de banco de 15 metros y ángulo de reposo de material de 35°.

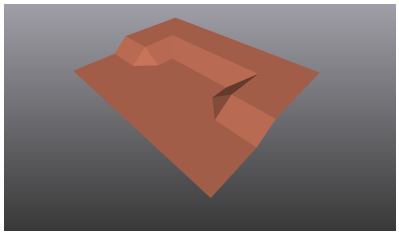
Además de la topografía inicial, se definieron dos topografías esperadas posterior al proceso de extracción, es decir, el resultado final del trabajo realizado por las palas en la realidad en un periodo de 16 a 24 horas. La primera, es una mejor representación de una topografía de planificación minera, donde es común planificar removiendo una sección del banco y dejando los frentes de extracción en 90°. Y la segunda es una topografía que tiene una pendiente correspondiente al ángulo de reposo del material en el sector extraído.



(a) Topografía sintética inicial.



(b) Topografía sintética final.



(c) Topografía sintética final con ángulo de reposo de material de 35°.

Figura 4.2: Topografías sintéticas en formato OBJ.

4.2. Automatización de MineTiler API.

El primer trabajo realizado fue automatizar las consultas a la API para generar una simulación de la realidad. El desarrollo de esta tarea se hizo en un *script* y dado una serie de parámetros de configuración, genera las simulaciones necesarias para evaluar el desempeño del algoritmo. Estas simulaciones se comparan con la realidad de la operación minera en un paso posterior.

Mas específicamente, un *script* en *Python* automatiza los llamados a la API de *MineTiler* cambiando las configuraciones del algoritmo de fresado que el usuario desea probar. Dichas configuraciones determinan la simulación de la superficie real y son los distintos resultados de este algoritmo el principal objeto de estudio de este documento.

Para que el algoritmo de fresado pueda ser ejecutado se deben realizar una serie de pasos y configuraciones previas. A continuación se detallarán los pasos previos y el ambiente que se debe montar para el correcto funcionamiento de la API *MineTiler*, y luego, el algoritmo de fresado. Finalmente, se describe el *script* desarrollado.

4.2.1. Conexión a MongoDB

Primero, se debe configurar la base de datos documental *MongoDB*. Dentro de una *database* de esta herramienta se definen colecciones que agrupan documentos en la base de datos. Para las configuraciones de *MineTiler* se utilizan las colecciones *'configuration'* y *'shovels'*.

Dentro de la colección *'configuration'* se requiere definir el tamaño de celda como *cellSize* y la cantidad de celdas por baldosa *tileSize*, datos para el acceso a *MinIO*, ruta a la API de *LHS*, parámetros de configuración para el algoritmo de fresado y otras configuraciones que no son de relevancia para esta investigación. Como por ejemplo, configuraciones de *logs*, de conciliación o de modelo de bloques.

Por otro lado, en la colección *'shovels'* se define cada pala que procesa el algoritmo Esto incluye el identificador de la pala, si está activa o no y parámetros como máximo de altura de extracción, radio menor, radio mayor y desplazamiento de extracción. Máximo de altura de extracción se utiliza para filtrar extracciones mayores a esta cifra, por ejemplo, si una celda fuera a cambiar 50 metros y el parámetro tiene un valor de 40 metros, la celda no cambia en el *DTM* resultante.

4.2.2. Ejecución de MineTiler

La instalación de *MineTiler* consiste en extraer el paquete en una carpeta para ser ejecutado. Dentro de la carpeta *'geometry-configuration'* se define *'documentStore.yml'*, que define la conexión al servidor de *MongoDB* para leer las demás configuraciones. La API requiere más definiciones para ser ejecutada que no son relevantes para el control del algoritmo de fresado. Por ejemplo, tamaños de ortofoto o propiedades del modelo de bloques u otros datos con los que trabaja la API.

Una vez ejecutado *MineTiler*, el primer paso es cargar una topografía real. Consultando el endpoint de carga de topografía, se toma el OBJ de entrada, se baldosa, se guardan los datos de la consulta junto a la ubicación del archivo resultante en *MongoDB* y en *MinIO* se almacenan los archivos binarios *DTM* y los archivos de alturas en escala de grises. Dichos archivos se utilizan para hacer cálculos volumétricos de la topografía y para visualizar la superficie en el gemelo digital respectivamente. Los archivos binarios que generen las consultas automatizadas son las que serán comparadas en la sección 4.3.

La consulta de carga de topografía es un *POST* que contiene el archivo como *multipart/-formdata* desde un form. En la figura 4.3 se muestra la función que hace la carga de archivo dentro del *script*.

Ya que MineTiler funciona con colas de trabajo, hacer dicha consulta solo nos responde que la tarea ha sido creada y nos da su *jobId*. Utilizando este identificador se hace una consulta distinta para saber si la tarea fue creada, está actualmente siendo procesada en la cola de trabajo, o ya ha finalizado de procesar. Solo en caso de haber terminado, entrega los resultados de la consulta. En la figura 4.4 se puede ver la parte del *script* que automatiza el proceso de esperar que esta tarea finalice para retornar la respuesta.

```
## http://{SERVER_HOST}:{SERVER_PORT}/jobs/newtopography/{DATE}
def newTopography(filename, date):
    url = minetiler_url + "/newtopography" + "/" + date
    files = {'file1': open(filename,'rb')}

    print("Requesting newtopography: ", url)
    return requests.post(url, files=files, headers=headers).json()
```

Figura 4.3: Función de carga de topografía de MineTiler.

4.2.3. Ejecución de algoritmo de fresado

Una vez configurado *LHS* y la topografía inicial está cargada se deben definir en *MongoDB* los parámetros de *'diggerConfiguration'* dentro de la colección *'configuration'*.

1. Tolerancia de altura *(heightThreshold)*: Define la profundidad máxima que puede tener un fresado. Esto es, para un valor de 20 metros, por celda, será ignorada aquella información que reduzca el valor de la celda más de 20 metros.
2. Mínimo de puntos en celda *(minPointsInCell)*: Define la cantidad mínima de datos GPS que debe tener una celda para que su altura sea modificada. Filtra datos de pala aislados.
3. Percentil *(percentile)*: Aplica un percentil a los datos GPS asociados a una celda, tal de considerar las alturas resultantes del procesamiento de datos GPS lo más cercanas al piso de la posición de la pala.

Para ejecutar fresado, se consulta el endpoint **freshtiledtopography** con una fecha específica. Existe también un endpoint **tiledtopography**, uno hace la consulta y sólo considera como base la topografía cargada mediante **new_topography**, el otro, toma como base la topografía calculada más recientemente, o con **new_topography** o **tiledtopography**. Esta última consulta será la utilizada para diagnosticar el uso de algoritmo de fresado ejecutándose cada 15 minutos.

Finalmente, se debe crear una nueva colección, está definirá los parámetros de configuración individuales para cada pala. Se debe crear en *MongoDB* la colección *'shovels'* y, para cada pala, definir una lista con las siguientes propiedades.

1. Identificador *(_id)*: Nombre de la pala, hace un vínculo con la consulta a *LHS* de *shovels/shovels*, asociando el nombre de la pala al id en la consulta *shovels/gpslogs*.
2. Desplazamiento de extracción en altura *(extractionOffset)*: Modifica los datos GPS de la pala en un valor z constante.
3. Estado *(status)*: Activa o desactiva la lectura de los datos GPS de la pala.

```python
# MineTiler API
## http://{SERVER_HOST}:{SERVER_PORT}/jobs/{job}
def statusFromRequest(request):
    # Get JobID from input POST request
    job_id = request['jobId']
    url = minetiler_url + "/status" + "/" + job_id

    print("Requesting status: ", url)
    response = requests.post(url, headers=headers).json()

    # Wait until task has finished and print response JSON
    while response['status'] != "Finished":
        if response['status'] == "Created":
            print("Task created, not processing: ", url)
        elif response['status'] == "Failed":
            return "Task failed."
        else:
            print("Processing, waiting for finished: ", url)

        time.sleep(5)
        response = requests.post(url, headers=headers).json()

    print("Finished request.", '\n')
    return response
```

Figura 4.4: Automatización de la espera del procesamiento en las colas de trabajo de MineTiler.

4. Radio interior *(innerRadius)*: Define el radio inferior centrado en una celda. Define el radio en el contacto entre el piso de la pala y la superficie de la topografía.

5. Radio exterior *(outerRadius)*: Define el radio superior centrado en una celda que define los datos geoespaciales de pala que se considerarán y también define el ángulo entre el piso de la superficie y el banco superior.

Debido a que *MineTiler* guarda los archivos en *MinIO*, no es necesario almacenarlos y se borran al terminar la ejecución de fresado. Sin embargo, existe una configuración que mantiene estos archivos para su depuración. Se hará uso de esta configuración y el hecho de que el *script* se ejecuta de manera local para obtener los archivos.

4.2.4. Script control_automation.py

Como se muestra en la figura 4.5, el script de automatización interactúa con MongoDB, con *MineTiler* para hacer consultas y recaudar archivos, y finalmente, con un ejecutable para convertir los archivos *DTM* binarios a formato *ASCII*.

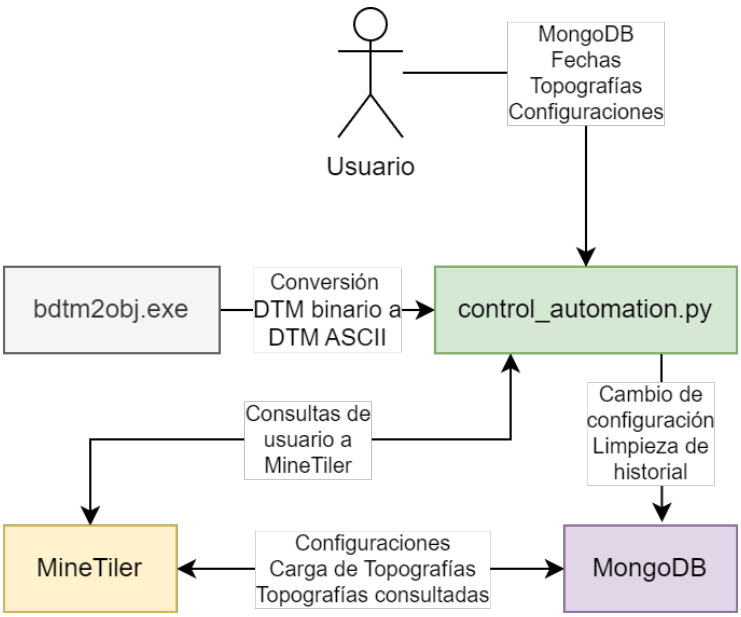


Figura 4.5: Arquitectura de la automatización de consultas a *MineTiler*.

Al inicio del *script* se dejaron las siguientes variables ambiente a utilizar para el resto de su ejecución:

- Servidor de *MineTiler*: Ruta y puerto de la API.
- *MongoDB*: URI para el acceso a MongoDB y base de datos a utilizar.
- Ruta de *'geometric-server'*: Directorio que almacena archivos temporales de *MineTiler*, configurado desde el *script* a no ser borrados.
- Directorio de salida: Donde se guardan los resultados del *script*.
- Fecha Inicial.
- Fecha Final.

Se desarrollaron en el *script* de automatización una serie de métodos que cambian valores en la base de datos de configuraciones en *MongoDB*, hacen llamados a *MineTiler* y transforman archivos resultantes en *DTM* binarios a *DTMs* en formato ASCII.

Los siguientes métodos son aquellos que cambian valores en la base de datos en MongoDB.

- changeVal: Dado un parámetro de configuración **key**, en el documento con identificador **config** en la colección *'configuration'* dentro de la base de datos de *MongoDB* y un nuevo valor **value**, se cambia dicho parámetro y se retorna el valor original.
- changeShovelVal: Dado un parámetro de configuración **key**, en el documento con identificador **shovel_id** en la colección *'shovels'* dentro de la base de datos de *MongoDB* y un nuevo valor **value**, se cambia dicho parámetro y se retorna el valor original.

- ClearCollections: Elimina las colecciones *'tiledTopographies'* y *'topographies'* que guardan el historial de las consultas de topografías hechas a *MineTiler*.

Los métodos que siguen, interactúan con *MineTiler* haciéndole consultas a su *API*.

- iRequestJob: Retorna el *URL* de una consulta cualquiera **request_job** con fecha **date** a *MineTiler*.
- newTopography: Dado un archivo topográfico en formato *OBJ* con nombre *filename* y una fecha **date**, hace una consulta *POST* con el archivo adjunto y retorna la respuesta de la *API*.
- tiledTopography: Dada una fecha **date** hace una consulta de un nuevo cálculo de topografía. Tiene un booleano opcional **fresh** que determina si considerara topografías intermedias calculadas por *MineTiler* o si solo considerará como base la ingresada por el usuario en la carga de topografía inicial.
- statusFromRequest: Dada la respuesta a una consulta **request**, obtiene el **jobId** de la consulta de respuesta y espera a que la consulta haya terminado de ser procesada.

A continuación, se describen los métodos que manejan los archivos *DTM* binarios locales de las consultas hechas a *MineTiler*.

- tunrFilesInFolderToDTM: Utiliza el ejecutable para convertir los archivos *DTM* binarios en una carpeta **bdtms** dentro de una carpeta **folder** a archivos *DTM* en formato *ASCII*.
- getBDTMFile: Dada una carpeta **folder**, y un nombre de archivo **filename** copia los archivos *DTM* binarios de la carpeta local de *MineTiler* correspondiente a una consulta en particular y a la guarda en una carpeta con la demás información de la ejecución del automatizado.

Los siguientes métodos utilizan lógicas distintas para cambiar los parámetros de configuración tomando en cuenta la colección en la que están y el formato. Un ejemplo es la configuración de suavizado de *DTM*, que consta de 4 parámetros pero que está guardado como JSON dentro de otra configuración en *MongoDB*. O también las configuraciones de palas que están en una colección distinta.

- ProcessConfigResult: Dado un parámetro de configuración **key**, en el documento con identificador **config** y valor **val**, se guarda el **DTM** binario resultante de una consulta **/freshtiledtopograhpy**.
- ProcessTemporalResult: Se hace una consulta a **tiledtopography** en pasos de 15 minutos desde la fecha inicial hasta llegar a la fecha final y se guardan los *DTMs* resultantes.
- ProcessShovelConfigResult: Dada una configuración **config** de pala con identificador **shovel_id** y valor **val**, se guarda el **DTM** binario resultante de una consulta **/freshtiledtopograhpy**.

- ProcessShovelRadiusResult: Dado un identificador de pala *shovel_id*, un radio exterior *outer* y un radio interior *inner*, se guarda el **DTM** binario resultante de una consulta **/freshtiledtopograhpy**.

Los métodos principales que ocupan lo que se ha descrito hasta ahora son los siguientes.

- generateControlDTMs: Hace la consulta de carga de topografía con la información que ingresó el usuario en las variables globales del script.
- generateConfigDTMs: Procesa las consultas cambiando las configuraciones en los rangos de valores que ingresa el usuario.
- generateTemporalDTMs: Procesa las consultas a lo largo del tiempo en el rango de fecha que ingresa el usuario.

El script primero crea una carpeta que identifica la ejecución del automatizado con fecha y hora. Luego, utiliza los métodos **generateControlDTMs**, **generateConfigDTMs** y **generateTemporalDTMs** para generar los archivos locales y un *JSON* de todas las respuestas a las consultas ingresadas por el usuario. Una vez terminado el procesamiento de las consultas, se limpia el historial de la base de datos con el método **ClearCollections**. Finalmente, se convierten todos los archivos binarios a ASCII y se guarda el *JSON* con todas las respuestas.

4.3. Comparación entre simulación y realidad

La comparación es el segundo paso de la aplicación y se hace recopilando las ejecuciones automatizadas y una topografía de control configurable. De este modo se puede evaluar el desempeño de la topografía generada por *MineTiler* para un rango de parámetros de configuración con respecto al dato de control.

Debido a la naturaleza de los archivos *DTM*, al compararlos, la única variable son los valores Z de ambos, ya que gracias a la estructura de *tiles*, las posiciones X e Y coinciden si ambos tienen el mismo tamaño de baldosa y de celda. Los cálculos y comparaciones se harán principalmente con respecto a esta variable. Es decir, el valor de cada celda del *DTM*.

Los archivos *DTM* que genera MineTiler están en formato binario, por ende, se utiliza un ejecutable de *TIMining* para pasarlos a formato ASCII, como en la figura 4.6. En este formato, se utiliza una estructura de datos de *DTM* en Python desarrollada por *TIMining* que hace uso de la librería *numpy* para hacer operaciones sobre la grilla de celdas.

Teniendo la lista de *DTMs*, se utilizan las siguientes funcionalidades para poder comparar ambas topografías. Primero, se generan gráficos de línea mostrando cómo varía el error para el rango de parámetros de configuración. Esto con el objetivo de encontrar tendencias y/o óptimos que minimizan el error contra la realidad basados en la topografía de control. Segundo, se generan mallas geométricas en formato *OBJ* visualizando las diferencias de cada

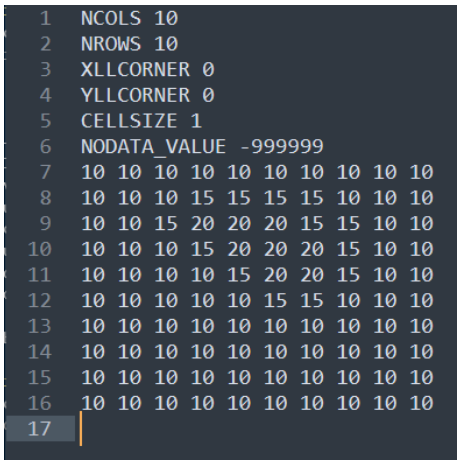


Figura 4.6: Formato *ASCII* de un *DTM*.

celda con mapas de calor en dos variables. La primera variable es qué valor parámetro de configuración cambió el valor de una celda. Y la segunda variable es la diferencia entre la topografía de control y una simulada.

4.3.1. Generación de gráficos

Para entender los resultados de las distintas métricas calculadas se desarrolló una serie visualizaciones de datos mediante un segundo *script* en *Python* llamado *compare_dtms.py*. En esta etapa se definen distintos parámetros ingresados por el usuario, como por ejemplo, la ubicación de los archivos creados por la automatización de *MineTiler* y las fechas. También, se define el *DTM* de control que representa el punto de referencia contra el cual se comparan las predicciones de la API.

Dadas estas definiciones, el usuario especifica, utilizando distintos métodos, los parámetros de configuración a evaluar, el archivo de control y finalmente el rango de valores para cada parámetro de la misma manera que se hace para el *script* de automatización. Los métodos están separados con el objetivo de ordenar los archivos con sus respectivos rangos de valores y configuraciones. Por ejemplo, se utiliza un método específico para la configuración de radio ya que este contiene dos rangos de valores, uno para el radio interior y exterior. También, se utiliza uno distinto para el caso temporal dado que utiliza un rango de valores en fechas dependientes de la cantidad de simulaciones realizadas a lo largo del periodo definido.

El usuario ingresa las siguientes variables globales.

- Carpeta de Datos: Resultados del script de automatización.
- Carpeta de Salida: Imágenes en formato *PNG* de gráficos de línea y mallas geométricas coloreadas en formato *OBJ*.

- Ruta de *'geometric-server'*: Directorio que almacena archivos temporales de *MineTiler*, configurado desde el *script* a no ser borrados.
- *DTM* de control: Archivo contra el cual comparar los *DTMs* en la carpeta de datos.
- Fecha Inicial.
- Fecha Final.

Los siguientes métodos trabajan con los archivos *DTM* de salida del script de automatización para generar los cálculos de las métricas de error.

- CalcMetrics: Dado un *DTM* de referencia y una lista de *DTMs*, se retornan en una lista tuplas que contienen los cálculos de métricas de error contra el archivo de referencia.
- GenData: Dado un *DTM* de referencia **dtm_control**, una configuración **config** y un rango de valores **range_l**, se leen los archivos nombrados con esas características en la carpeta de datos definida por el usuario y se llama a **CalcMetrics** con la lista de archivos.
- GenRadiusData: Dado un *DTM* de referencia **dtm_control**, un radio exterior **radius_outer** y un radio interior **radius_inner**, se leen los archivos nombrados con esas características en la carpeta de datos definida por el usuario y se llama a **CalcMetrics** con la lista de archivos.
- GenTemporalData: Dado un *DTM* de referencia **dtm_control**, se leen las topografías generadas en el periodo de tiempo ingresado por el usuario y se llama a **CalcMetrics** con la lista de archivos.

Sin embargo, la labor de estos métodos es la misma y consiste en leer los archivos en formato *DTM*, ordenarlos en una lista en base al parámetro y finalmente hacer el llamado para calcular las métricas de error. Como resultado, se retornan 5 listas, cada una representando las métricas de error calculadas entre el archivo de control y la lista de resultados entregados por *MineTiler* para cada parámetro. Por ejemplo, en un diccionario, con la llave *minPointsInCell* se guardan 5 listas de métricas de error, cada una conteniendo los resultados de cada parámetro de configuración. Es decir, un resultado de las 5 métricas para un valor 1 de mínimo de puntos en celda, otros 5 resultados para un valor 2 de mínimo de puntos en celda y así consecutivamente hasta completar el rango de valores y las configuraciones definidas por el usuario.

El cálculo de las métricas *RMSE*, *MASE*, *MSE*, *MAE* y *R2* se logra dentro de la estructura de datos de *DTM* en *Python* que ocupa *TIMining* que contiene un arreglo en 2 dimensiones con sus valores Z. Cabe destacar que primero se filtran los valores nulos del *DTM*, que en este caso en particular tiene un valor numérico de -999999, para no considerarlos en pasos como el cálculo del promedio de valores Z. Una vez filtrados los valores nulos, se aplican las fórmulas descritas en la sección 2.1.1.

En todos los gráficos que se mencionan en esta sección se utilizó en la variable horizontal los parámetros de configuración, en la variable vertical los resultados de los cálculos y finalmente en la variable color las distintas métricas de error. Cada generación de gráficos debe ser

especificada por el usuario haciendo coincidir la llave de la configuración como sale en la base de datos de *MineTiler* y también el rango de valores con el que se hizo la automatización.

Se desarrollaron 3 métodos generadores de visualizaciones de datos utilizando la librería *matplotlib*, uno para *RMSE* y *MASE*, otro para *MSE*, *MAE* y *R2* y una última para visualizar las métricas a lo largo del periodo temporal definido por el usuario.

La herramienta de visualización de datos generada por el equipo de calibración de *MineTiler* tiene una variedad de indicadores que les ayuda a identificar cuáles son los efectos y el óptimo para cada configuración. Durante el proceso de calibración que motivó esta investigación, se calcularon estadísticas básicas como promedios, desviaciones estándar y con un solo parámetro de configuración a la vez, requiriendo un alto costo en poder ordenar los datos y hacer las ejecuciones pertinentes, tomando días para poder identificar las fuentes de error de una simulación inadecuada. Por lo tanto, usando la ilustración de las tendencias del error en conjunto con el conocimiento minero del equipo, rápidamente se identifica cuál es el indicador que tiene más efectos en la topografía. Cabe destacar, que debido a las diferencias en la manera de operar, las bases de datos y las imprecisiones de los datos geoespaciales en cada faena, este es un proceso obligatorio en el cual lo más probable es que los efectos de las configuraciones sean distintos para cada cliente.

Uno de los casos de éxito de *TIMA* es un cliente que debido a su insatisfactoria administración de sus datos, notaba errores en lo que reportaba el gemelo digital. Debido a las conversaciones sobre estos resultados, la faena mejoró considerablemente su manejo y organización de los datos. Es posible con esta solución informar al cliente de los altos errores que causan los datos irregulares en el software, especialmente si se ocupan comparándolo con otros clientes en el cual la simulación aporta valor.

A continuación se describen cada una de las visualizaciones de datos de las métricas de error.

4.3.1.1. Visualización principal de *RMSE/MAPE*

Debido a la facilidad de interpretación de dos métricas de error en particular, *RMSE* y *MASE*, se realizó un gráfico de líneas exclusivamente para mostrar ambos resultados como se muestra en la figura 4.7. Debido a que son escalas distintas, se utilizó el eje izquierdo para mostrar la escala en metros de *RMSE* y el eje derecho para mostrar la escala en porcentaje de *MASE*. Se utilizo la variable de color para distinguir fácilmente las métricas de error explicitadas en la leyenda. Debido a que el sector de mayor interés es cuando las métricas están en su mínimo, para esta visualización la leyenda se encuentra en la parte superior y al centro.

La generación de esta visualización es definida por el usuario mediante el método *PlotMainConfig* y se le entrega como parámetro propiedades del eje horizontal. En concreto, la llave de configuración como sale en *MongoDB*, el nombre que le corresponde en el eje horizontal y el rango de valores en el mismo eje. Finalmente, se guarda la visualización con el prefijo *main* seguido de la llave de configuración en la carpeta especificada por el usuario en una imagen en formato *PNG*.

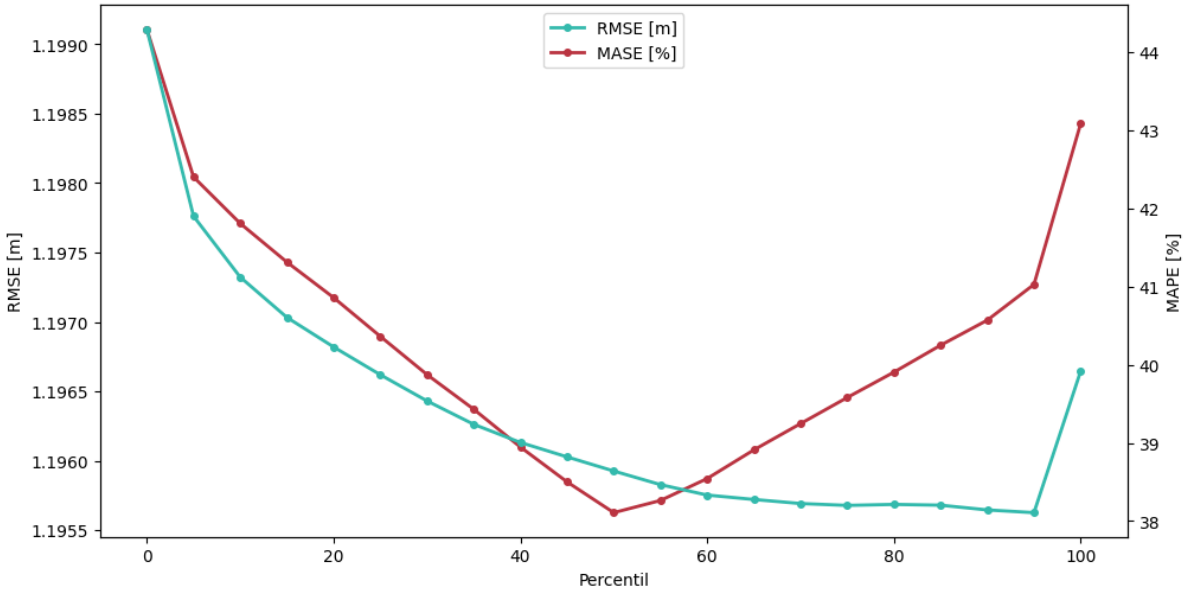


Figura 4.7: Ejemplo de visualización de *RMSE/MAPE* para la configuración de percentil entre los valores 0 y 100 en 20 pasos para datos ideales.

La visualización muestra, en cada punto de la línea, el resultado de la comparación entre la topografía de control y la simulación generada por *MineTiler* con un parámetro de configuración en particular. Esto tiene el objetivo de notar la tendencia del error utilizando distintas configuraciones. Debido a que la variable vertical es error, el punto de más interés en el gráfico son los valles que representan el menor error.

4.3.1.2. Visualización de *MSE*, *MAE* y *R2*

Debido a que *MSE*, *MAE* y *R2* tienen escalas distintas, se generó una visualización con 3 gráficos de línea ordenados verticalmente, compartiendo la variable horizontal como se ve en la figura 4.8. De este modo, cada gráfico de línea tiene su respectiva escala. De todas maneras se utiliza la variable color para distinguir las métricas con su respectiva leyenda en cada uno de los 3 gráficos. La leyenda es la misma que la visualización anteriormente descrita excepto por el gráfico de *R2* debido a que en este caso los valores máximos cercanos a 1 son de más interés.

Esta visualización se genera utilizando el método *PlotConfig* y funciona utilizando lógica similar al caso de la visualización de *RMSE/MASE*, recibiendo los mismos parámetros. Se guarda de la misma manera que la visualización anterior excepto por el prefijo.

Al igual que el gráfico de línea descrito anteriormente, cada punto representa la simulación de una topografía para un parámetro de configuración. El objetivo es ver la tendencia del

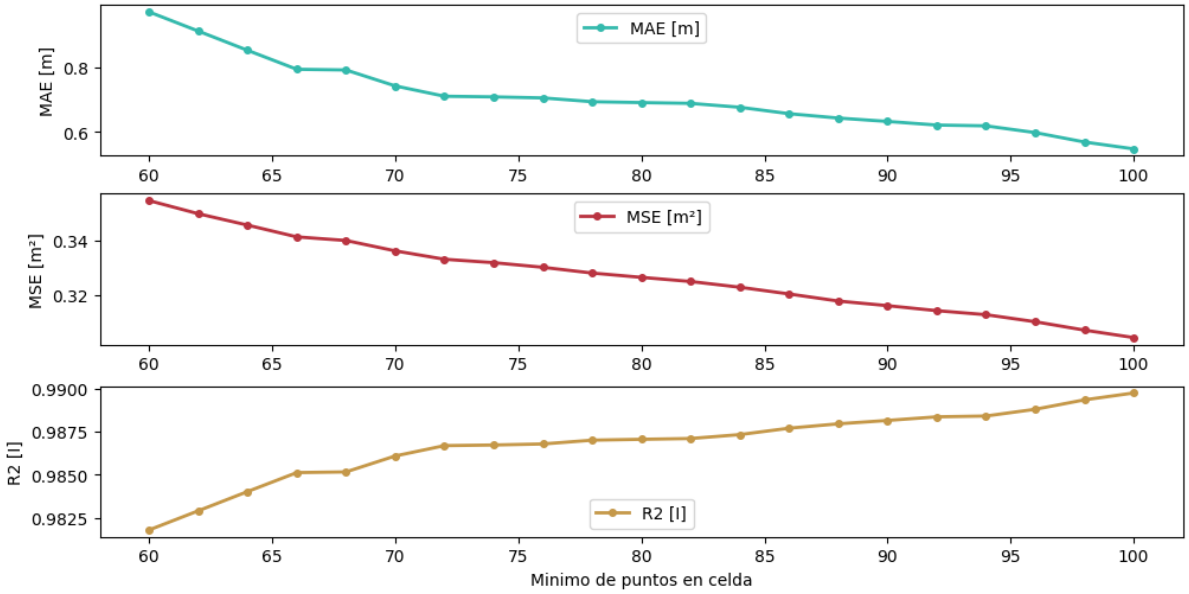


Figura 4.8: Ejemplo de visualización de *MAE/MSE/R2* para la configuración de mínimo de puntos en celda entre los valores 60 y 100 en 20 pasos para datos normales.

error mientras que cambia el valor del eje horizontal. En el caso de la métrica *R2*, lo relevante es cuando el valor tiende a 1 que representa una buena aproximación de los valores de altura en el *DTM*. Las unidades de error de *MAE* y *MSE* están en metros y metros cuadrados respectivamente, por lo tanto el menor valor representa el mínimo error en la topografía.

4.3.1.3. Visualización temporal de *RMSE/MAPE*

Esta visualización ayuda al diagnóstico del algoritmo de fresado en su caso de uso principal. Como es mencionado anteriormente en sección 3.2, el algoritmo utiliza, si es que existen, las simulaciones pasadas para generar una nueva topografía. Por lo tanto, dada una fecha inicial y final definida por el usuario, se recopilan los archivos *DTM* generados cada 15 minutos y se calculan las métricas de error. En particular esta visualización hace uso de los archivos generados por la automatización mediante el método *ProcessTemporalResult*.

En esta visualización, la variable horizontal es el periodo entre la fecha inicial y final, manteniendo la lógica y las mismas variables de las visualizaciones ya descritas. Al igual que las demás visualizaciones de métricas de error, se guardan en formato *PNG* con nombre *temporal*

El objetivo de esta visualización es identificar cuales son los horarios donde hay cambios en el error de la topografía y poder identificar los horarios de consultas que hayan generado error. La topografía contra la que se compara esta al final del periodo y se muestra como

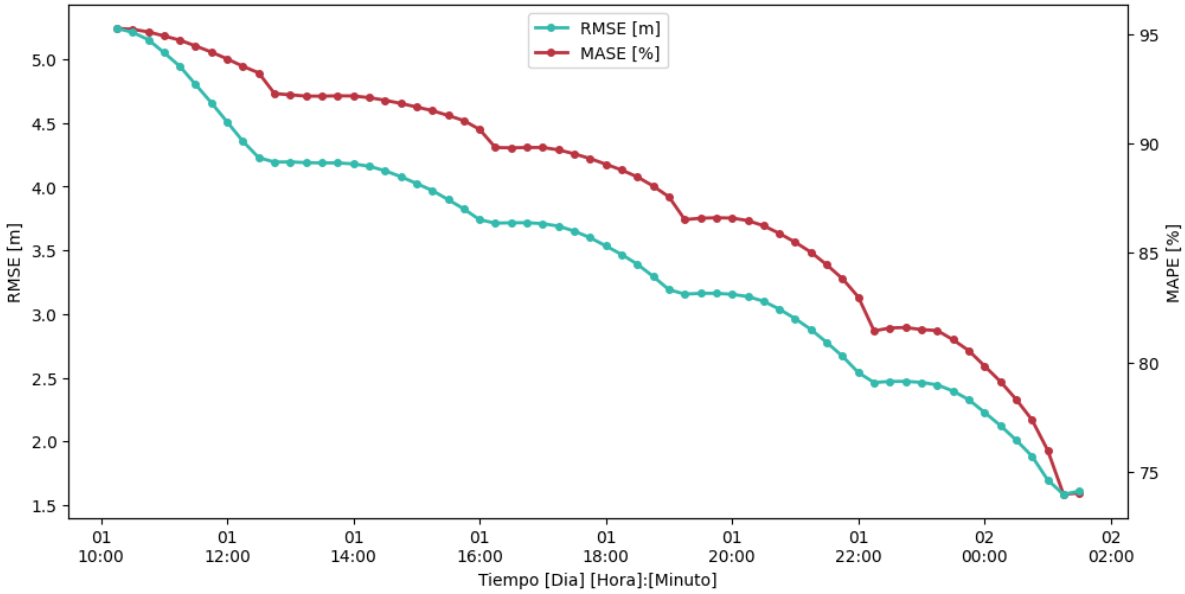


Figura 4.9: Ejemplo de visualización de *RMSE/MASE* para el caso de uso acumulativo del algoritmo de fresado cada 15 minutos para el periodo del 1 de Marzo del 2023 a las 10:00 hrs al 2 de Marzo del 2023 a las 02:00 hrs para datos ideales.

gradualmente tiende al mínimo error al aproximarse al horario del archivo de referencia. La información relevante está en cambios repentinos o pronunciados en el error.

4.3.2. Generación de mallas geométricas comparativas

Otra de las posibles salidas del *script* son dos variedades de visualización con mapas de calor en formato OBJ entre *DTMs*. Ambas harán uso de la facilidad de restar *DTMs* entre sí ya que solo se deben restar sus valores Z por celda y así tener un *DTM* de la diferencia entre ambos. Se colorea la malla geométrica por dos variables, la diferencia en valor Z de cada celda y el primer parámetro de configuración que cambió una celda como se muestra en la figura 4.10.

En la primera, se hace una máscara *DTM* en paralelo al de entrada y se itera en todos los *DTM* resultantes de una configuración específica. En esta iteración se hace un histórico de las celdas que han ido cambiando asignándole valores secuenciales de las ejecuciones a la celda.

Las mallas geométricas en formato *OBJ* se colorean asignando valores *RGBA* a cada vértice. Debido a que el *DTM* es una grilla, cada celda está conformada por dos triangulos y estos se colorean en base a los colores de sus vértices. Esta propiedad no es compatible con cualquier visualizador, en este caso se utiliza el software *MeshLab* que es común para la

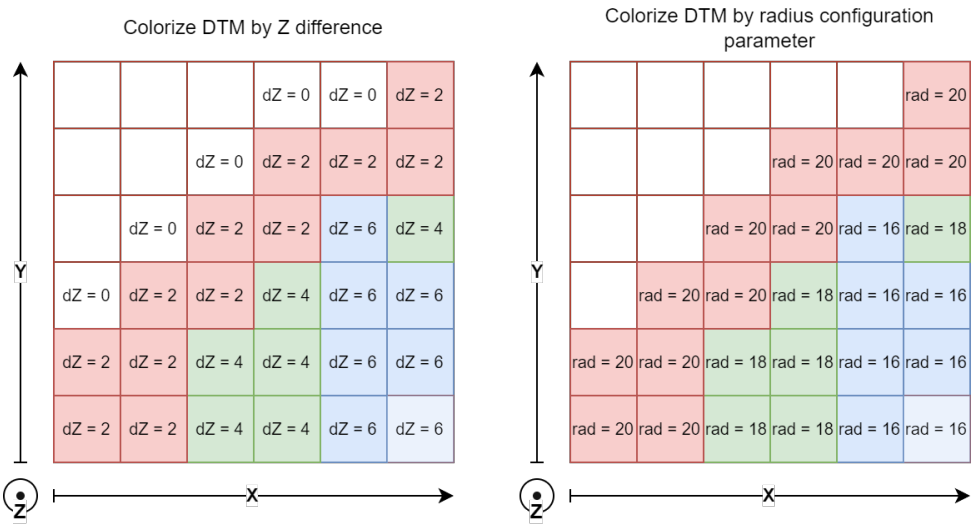


Figura 4.10: Ejemplo de coloreado de un *DTM*. En la derecha se hace un mapa de calor por la diferencia en altura de la celda con la de referencia. En la izquierda se colorea por el parámetro de configuración que cambió la celda por primera vez.

empresa y usada habitualmente para visualizar mallas geométricas en formato *OBJ*.

Se implementaron los siguientes métodos para la generación del archivo *OBJ* coloreado.

- ColorizeByParam: Genera una malla geométrica en formato *OBJ* coloreada por valor de parámetro de configuración.
- ColorizeByRadiusParam: Genera una malla geométrica en formato *OBJ* coloreada por valor de parámetro de configuración de pala.
- ColorizeByObjDiffs: Genera una malla geométrica en formato *OBJ* coloreada por diferencia en valor de altura.

4.3.3. Visualización geométrica con variable de color por parámetro de configuración

Como ejemplo, para una prueba de la configuración de radio de pala, se itera en valores de radio de 15 a 25 metros en 20 pasos. Entonces las celdas que fueron cambiadas teniendo un radio 15 estarán a un extremo del espectro del mapa y las celdas que fueron cambiadas teniendo un radio de 25, estarán en el otro extremo del espectro. Habrán 20 pasos entre ellos representando cada ejecución. De esta manera, se puede ver como el cambio en los valores de configuración influyen en las celdas del *DTM* por sobre la topografía final esperada o la simulada.

El objetivo de esta visualización es primero poder notar anomalías que haya causado un parámetro en particular y segundo ver cómo avanza el trabajo realizado por fresado a medida

que cambia la configuración.

En términos de desarrollo, se utiliza un método *FiveColorHeatMap* que convierte un valor entre 0 y 1 a valores *RGB*. Luego, se itera en los *DTMs* de cada parámetro y para cada celda que presente una diferencia con el archivo de referencia, se le asigna ese parámetro a la celda. El valor de la última ejecución toma prioridad ya que en la mayoría de los casos las diferencias se repiten dentro de las celdas. La malla geométrica base para el coloreado es con el último parámetro ejecutado.

4.3.4. Visualización geométrica con variable de color por diferencia en valor Z

La segunda visualización, es con la última ejecución de cada configuración con una variable de color distinta. En esta oportunidad, simplemente se toma la topografía y se colorea en base al *DTM* de diferencias con el último valor del parámetro ejecutado. Lo que se busca identificar con esta visualización, es la magnitud de diferencia en cada celda comparado con lo esperado sin considerar cada valor.

Nuevamente, se ocupa el mismo método para asignar valores entre 0 y 1 para convertirlo en un valor *RGBA* y se le asigna a la celda.

Capítulo 5

Resultados

En esta sección se hará un análisis de los resultados arrojados por la solución propuesta para cada uno de sus componentes exceptuando la creación de datos sintéticos ya que no es relevante en la investigación del desempeño del algoritmo de fresado.

Primero se estudia la ejecución de la aplicación. Esto incluye su salida en consola y tiempo de respuesta. Luego se discuten sus archivos de salida. Es decir, los gráficos de las métricas de error implementados y las visualizaciones geométricas de la topografía resultante.

Por un lado se estudia cuáles son las preguntas que puede contestar el equipo de calibración de *TIMA* utilizando dichas visualizaciones. Y además, se discute la solución en sí, identificando sus limitaciones y posibles mejoras.

Se utilizaron 3 conjuntos de datos de palas con aleatoriedades distintas pero movimientos idénticos. El primer grupo son datos geoespaciales ideales del movimiento de la pala como en la figura 4.1a. También serán usados los datos de la figura 4.1b. Y finalmente se creó un punto medio más parecido a palas con poca precisión.

Las siguientes son las configuraciones base de *MineTiler* para las ejecuciones realizadas:

- Tolerancia de altura *(heightThreshold)*: 70
- Mínimo de puntos en celda *(minPointsInCell)*: 40
- Percentil *(percentile)*: 6

Junto a estas configuraciones de pala:

- Identificador *(_id)*: 0
- Desplazamiento en altura de la pala *(extractionOffset)*: 0
- Estado *(status)*: Activo
- Radio interior *(innerRadius)*: 20

- Radio exterior *(outerRadius)*: 20

Los parámetros y rangos elegidos a cambiar fueron los siguientes:

- Mínimo de puntos en celda: De 60 a 100 en 20 pasos.
- Desplazamiento en altura de la pala: De 0 a 4 en 20 pasos.
- Percentil: De 0 a 100 en 20 pasos.
- Radio interior y exterior*(innerRadius)*: De 15 a 25 en 20 pasos.

Por último, la evaluación temporal del algoritmo de fresado cada 15 minutos se hará en una ventana de 16 horas, desde la fecha del 1 de enero del 2023 a las 10:00 hrs, hasta el 2 de enero hasta las 02:00 hrs para un total de 62 ejecuciones.

5.1. Ejecución de *scripts*

En esta sección se describe la ejecución de los distintos scripts en *Python* para generar la información necesaria, calcular las métricas de error y generar visualizaciones geométricas del algoritmo de fresado.

Lo más relevante son los resultados entregados por el script de automatización ya que contiene bastante información de progreso debido a su interacción con componentes como por ejemplo, *LHS*, *MinIO* y *MineTiler* en sí.

Uno de los procesos que toman más tiempo, tanto en esta solución, como en el uso normal de *MineTiler*, es la obtención de datos desde *LHS*. Debido a que los datos de palas se obtienen de manera local en las pruebas de este trabajo de memoria este tiempo es reducido considerablemente.

5.1.1. Automatización de *MineTiler*

Para la automatización de *MineTiler*, el progreso consiste en reportar cuando una consulta a la API está siendo procesada. También reporta las variables de *MongoDB* que fueron tanto cambiadas como restauradas a su valor original.

La automatización cumple con el objetivo necesario al dejar todos los *DTMs* resultantes en la carpeta configurada por el usuario, restaura los parámetros de configuración modificados en *MongoDB* y finalmente limpia los datos intermedios generados por estas consultas.

En la figura 5.1 se muestra la automatización de *MineTiler* cambiando los parámetros de configuración y esperando a que las consultas a la API respondan.

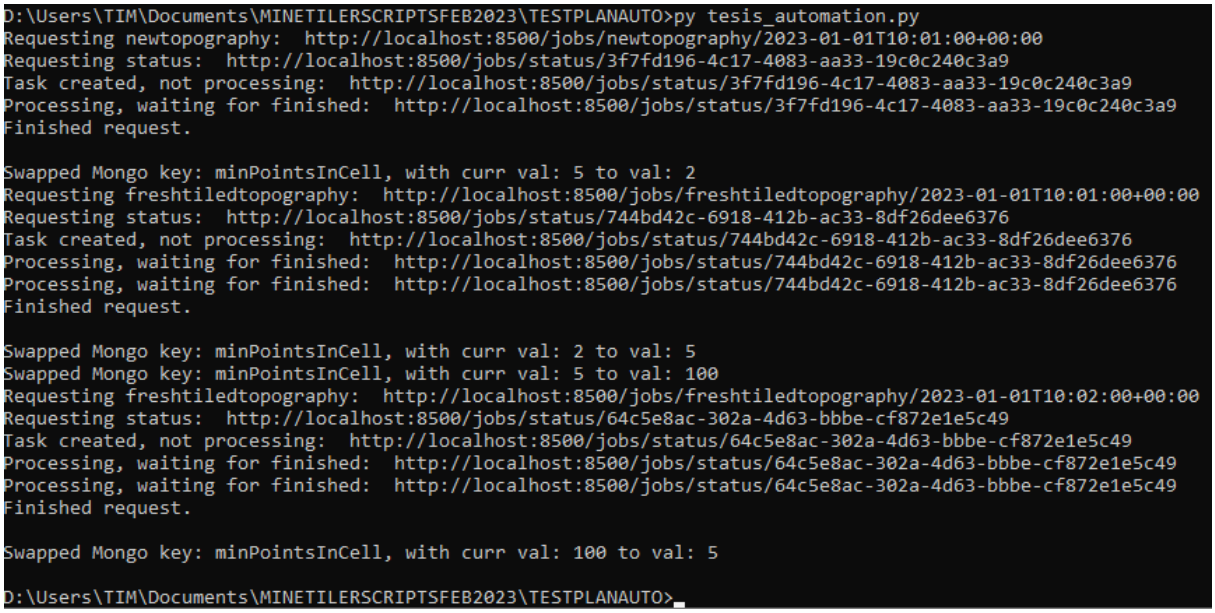


Figura 5.1: Ejecución de script de automatización.

5.1.2. Comparación de *DTMs*

Finalmente la ejecución de *compare_dtms.py* entrega principalmente la información de progreso del cálculo de métricas de error, que es la operación más costosa de este script. Aun así, sigue siendo una pequeña fracción del tiempo que toma *MineTiler* en responder a las consultas en el paso anterior, especialmente en un ambiente real con una API de *LHS*.

En la figura se muestra como cada vez que se empiezan a calcular métricas de error para alguna configuración, el *script* indica cuantos *DTMs* se están utilizando y cuánto demoró el cálculo para dichos archivos.

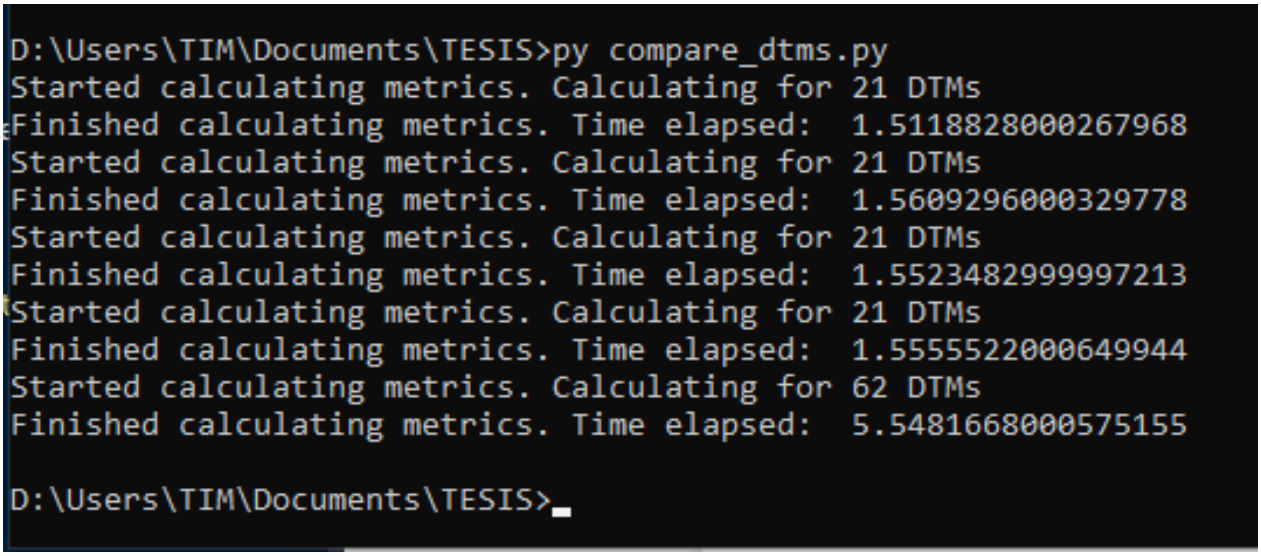


Figura 5.2: Ejecución de script de comparación de *DTMs*.

5.2. Gráficos

A continuación se analizan los resultados arrojados por las visualizaciones de datos en gráficos de línea de cada uno de los parámetros de configuración y su conjunto de datos.

5.2.1. Percentil

La configuración de percentil se utiliza para filtrar datos anómalos que tienen una elevación mayor a la de la superficie de la mina donde es más probable que esté la pala. Este cálculo se hace posterior al de mínimo de puntos en celda, el cual define cuales son los datos a considerar. Por defecto se utiliza un valor de 6.

Primero se considera la visualización utilizando datos ideales. En Chile la industria minera está a la vanguardia en términos de tecnología, por lo tanto es común ver palas con datos geoespaciales de alta precisión que, aunque con imprecisiones, suelen ser consistentes con la posición de la pala, especialmente en términos de altura.

Podemos notar en la figura 5.3 que para casos ideales que están perfectamente alineados con el banco, un valor de percentil de 50 es el que tiene el menor error en los resultados. Sin embargo, la mayor diferencia en *RMSE* es de tan solo aproximadamente 5 centímetros. Por parte de *MASE*, es de aproximadamente menos del 5 %. Dados estos valores, es improbable que el cambiar el percentil en casos ideales tenga un efecto considerable en la topografía que muestra *TIMA*, especialmente solo a la vista.

Siguiente, se consideran los datos más imprecisos en la figura 5.4. Para este caso, podemos notar no tan solo una tendencia más interesante, en donde el mayor error aún está en el menor percentil, prefiriendo considerar la mayor cantidad de datos utilizando un parámetro mayor a 60. También, el indicador *MASE* muestra una variación más considerable, reduciendo drásticamente de un 40 % de error a un 20 % solo pasando de un parámetro de percentil 80 a 100.

Finalmente, utilizando el conjunto de datos de mala calidad en la figura 5.5, el mínimo error por parte del indicador *MASE* no baja del 70 %, teniendo un claro mínimo en el valor de percentil 50.

Las visualizaciones secundarias de *MSE*, *MAE* y *R2* muestran tendencias similares para cada uno de los conjuntos de datos.

Las visualizaciones muestran que el valor sugerido del percentil para los datos utilizados tiende a ser mucho mayor que el por defecto actualmente utilizado por *MineTiler*. El percentil se usa principalmente para eliminar anomalías; al no tener datos muy elevados como se ha visto en la práctica, en casos de puntos aislados con alturas de más de 30 metros del banco, se prefiere considerar la mayor cantidad de datos posible.

Por lo tanto, con esta información se evidencia que si se tiene datos sin anomalías extremas, es mejor utilizar un valor lo más alto posible del percentil.

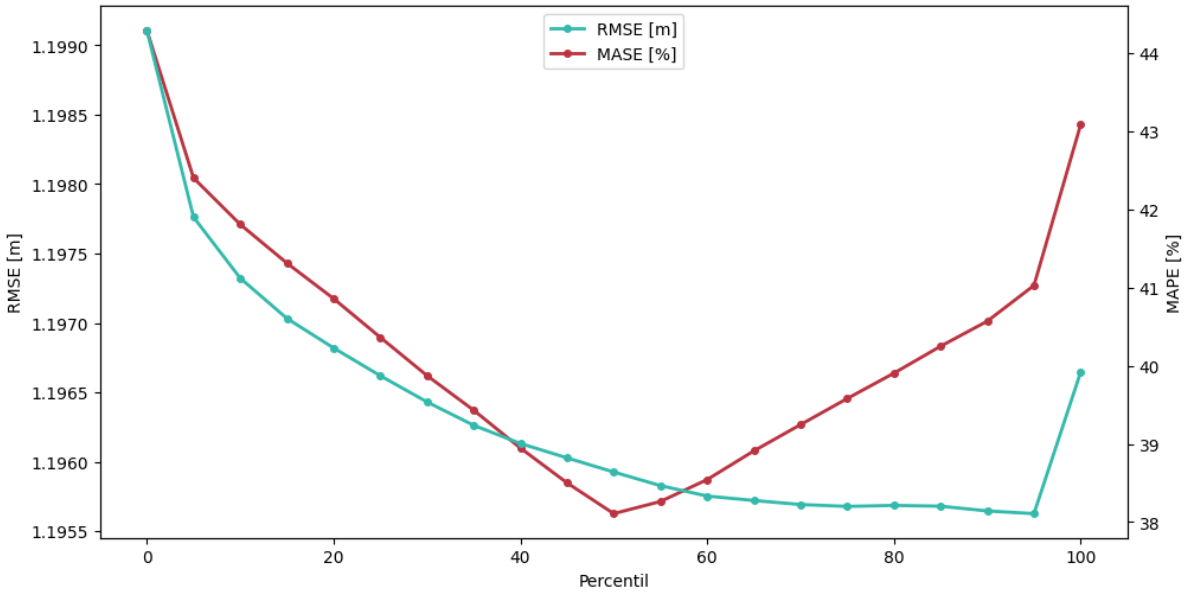


Figura 5.3: Ejemplo de visualización de *RMSE/MAPE* para la configuración de percentil entre los valores 0 y 100 en 20 pasos para datos ideales.

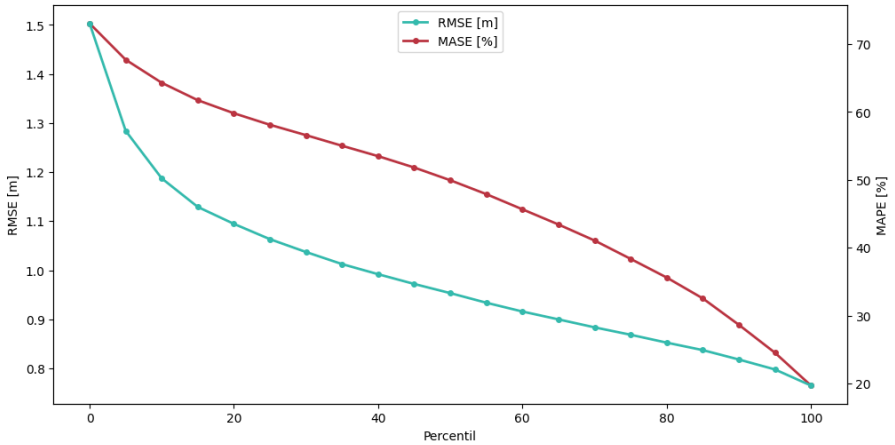


Figura 5.4: Ejemplo de visualización de *RMSE/MAPE* para la configuración de percentil entre los valores 0 y 100 en 20 pasos para datos imprecisos.

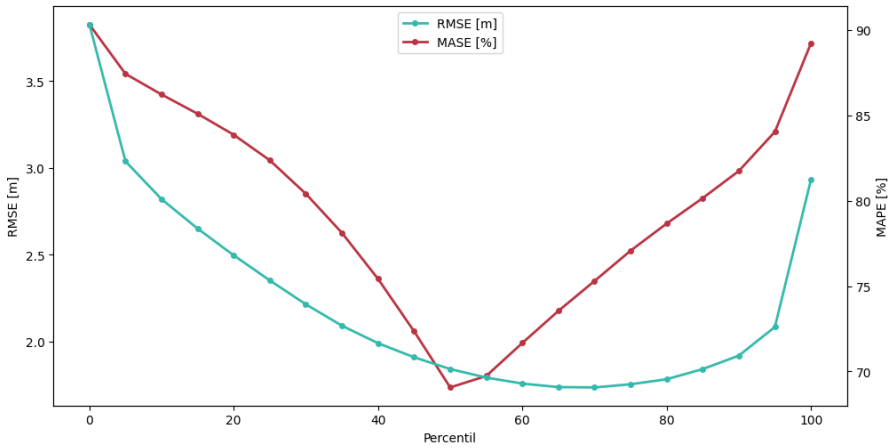


Figura 5.5: Ejemplo de visualización de *RMSE/MAPE* para la configuración de percentil entre los valores 0 y 100 en 20 pasos para datos de mala calidad.

5.2.2. Mínimo de puntos en celda

La configuración de mínimo de puntos en celda se utiliza considerando que las palas en faena tienden a quedarse detenidas mientras extraen material, dándole más prioridad de modificar la topografía a estas posiciones de la pala donde se concentra una cantidad de los datos *GPS*.

Entre los movimientos definidos en los datos de pala generados, se simularon detenciones de la pala en un mismo punto por un cierto periodo de tiempo con el *LHS* sintético usando para este trabajo.

Siguiendo el mismo orden de conjunto de datos, para los datos ideales en la figura 5.6, nuevamente hay poca magnitud de cambio en el error por parámetro. Sin embargo, la forma de la visualización es mucho más interesante, mostrando varios sectores en línea horizontal. Esto se interpreta como cambios en los parámetros que realmente no incorporaron más datos en la simulación de la topografía. De todas maneras, hay un claro mínimo de error entre los valores 70 y 75.

Para los demás conjuntos de datos de la figura 5.7 y 5.8 los resultados son similares, el error tiene la tendencia de disminuir mientras se aumenta el mínimo de puntos en celda, prefiriendo el mayor valor antes de que sea tan alto que no considere datos para hacer el fresado. Una de las posibles acciones en estos casos es aumentar el rango para el cual se hacen los cálculos de los indicadores tal de determinar cuál es el punto de inflexión que comienza a incrementar el error. De todos modos, el efecto en magnitud que tiene cambiar este parámetro al menos en el rango indicado es bajo, variando solo un 3 en el caso de datos de mala calidad, un valor mucho menor al esperado al cambiar dicha configuración.

En conclusión, para el mínimo de puntos en celda, se determina que la configuración tiene

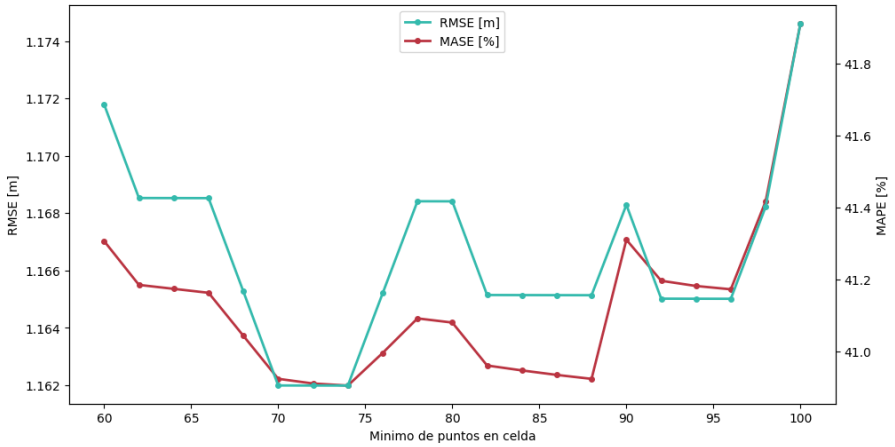


Figura 5.6: Ejemplo de visualización de *RMSE/MAPE* para la configuración de mínimo de puntos en celda entre los valores 60 y 100 en 20 pasos para datos ideales.

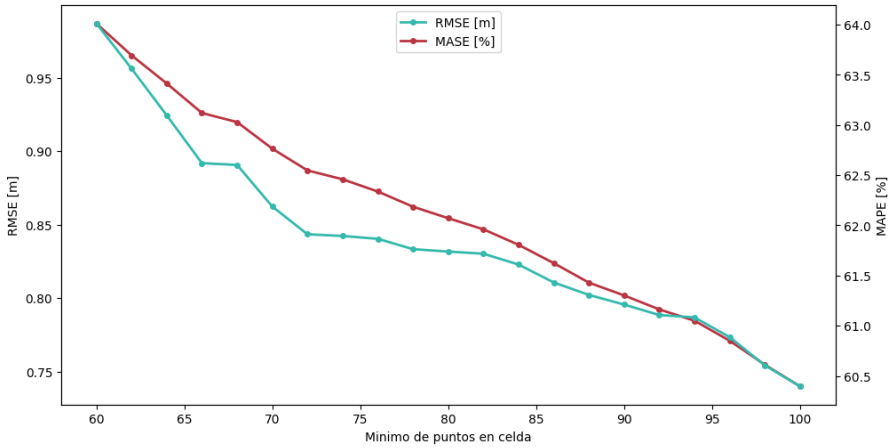


Figura 5.7: Ejemplo de visualización de *RMSE/MAPE* para la configuración de mínimo de puntos en celda entre los valores 60 y 100 en 20 pasos para datos imprecisos

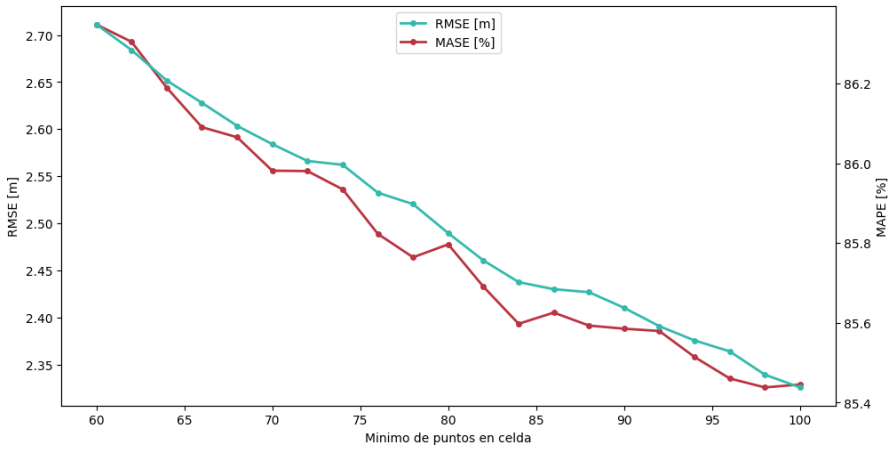


Figura 5.8: Ejemplo de visualización de *RMSE/MAPE* para la configuración de mínimo de puntos en celda entre los valores 60 y 100 en 20 pasos para datos de mala calidad.

poco efecto en el error del algoritmo de fresado. Esto se debe a que en las figuras 5.6, 5.7 y 5.8 el error porcentual en el eje izquierdo es de a lo más un 4 % para el caso de datos imprecisos o incluso menos de 1 % para datos ideales. Un resultado de este tipo le ahorra tiempo al equipo de calibración identificando que estos parámetros no son fuente de error para estos ejemplos.

5.2.3. Desplazamiento de extracción en altura

Esta configuración fue implementada para controlar manualmente la altura de la extracción realizada por el algoritmo de fresado para abordar el caso de un cliente en particular donde las palas no extraían material recorriendo la base del banco, si no que por el nivel superior de este extrayendo el material desde arriba.

Aun así, la configuración es capaz de ambos cubrir este nuevo caso de uso y también minimizar el error cuando las palas habituales tienen algún desplazamiento en su elevación. En las ocasiones que esta configuración ha sido usada, solo se configuró la altura del banco para compensar donde se extrae. Sin embargo, con esta visualización se comprueba cual es el valor óptimo para minimizar error en la topografía, especialmente si difiere de la altura del banco.

Dados los conjuntos de datos de prueba, se visualizó el error para un rango de parámetros de esta configuración e identificar si tienen algún valor óptimo para datos de pala de distinta calidad.

Para los datos ideales en la figura 5.9, dado a que los datos están bien ajustados al banco, el mejor valor es el mínimo y al incrementarlo sólo deforma la base de la topografía, incrementando el error de un 40 % a un 80 %.

En los casos de datos imprecisos y de mala calidad de las figuras 5.10 y 5.11 se presenta el caso contrario, donde la imprecisión de los datos genera un óptimo reducir la altura de la extracción del algoritmo de fresado. El cambio en porcentaje es considerable, de hecho más considerable que la configuración de percentil reduciendo el porcentaje de error un 40 %.

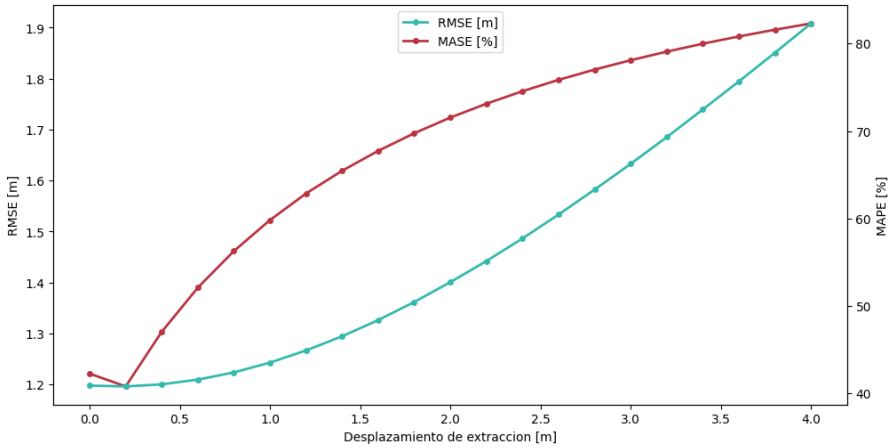


Figura 5.9: Ejemplo de visualización de *RMSE/MAPE* para la configuración de desplazamiento de extracción entre los valores 0 y 4 en 20 pasos para datos ideales.

El desplazamiento tiene la particularidad de cambiar toda la base de topografía, mientras que el cambiar el percentil tiene un efecto más dependiente de cada dato. Por lo tanto, es esencial trabajar con ambos para asegurar el menor error posible. Identificando primero el percentil adecuado dadas las anomalías en los datos y luego acomodar el desplazamiento de extracción para estar a nivel con el banco en la realidad.

5.2.4. Radio

El último resultado de las visualizaciones de indicadores de error es utilizando la configuración de radio. Las palas tienen en sus especificaciones técnicas el radio de alcance en los que operan y *MineTiler* lo ocupa por defecto configurándose por pala. Sin embargo, el gráfico tiene como objetivo corroborar que dicho valor sea el óptimo, y si no lo es, evidenciar que radio utilizar teniendo en cuenta los datos. Por lo tanto esta visualización busca minimizar el error que pueden causar las discrepancias entre el dato de la especificación y el trabajo que realiza el algoritmo de fresado.

Para el caso ideal en la figura 5.12 se ve que el óptimo radio para ambos indicadores está cercano a los 20 metros, evidenciado por un claro valle en la visualización. Debido a la naturaleza de la configuración de radio, el error varía drásticamente entre parámetros aunque en este caso eso se corrige utilizando un rango de parámetros más acotado.

Los tres gráficos de línea varían considerablemente en su forma, entregando distintos

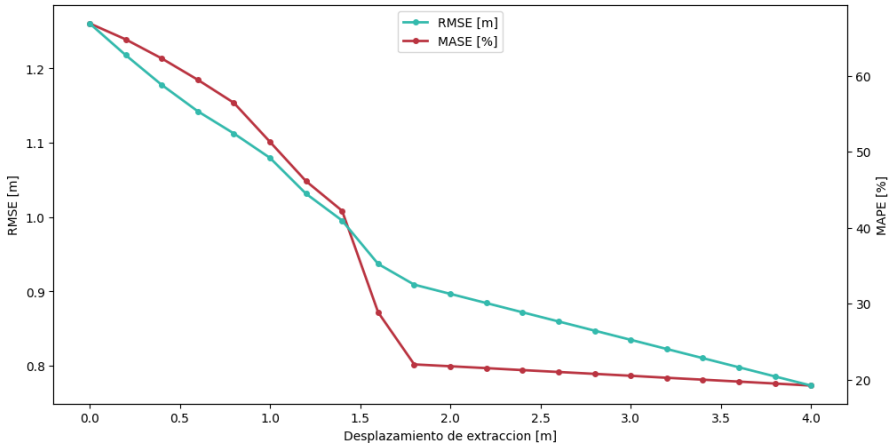


Figura 5.10: Ejemplo de visualización de *RMSE/MAPE* para la configuración de desplazamiento de extracción entre los valores 0 y 4 en 20 pasos para datos imprecisos

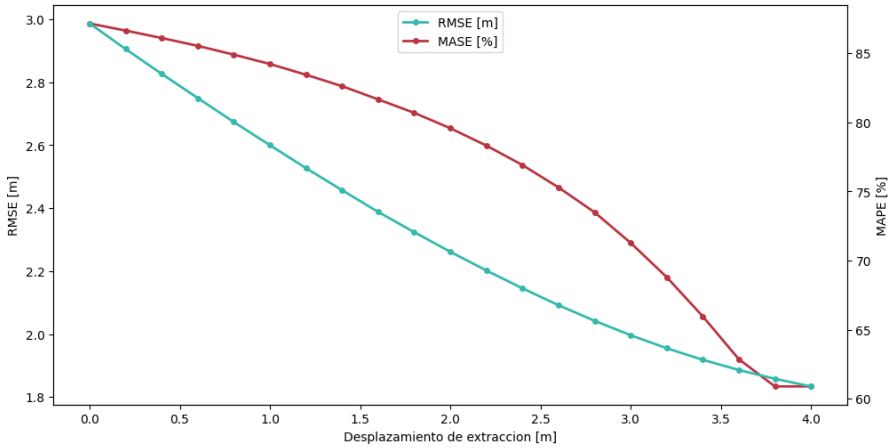


Figura 5.11: Ejemplo de visualización de *RMSE/MAPE* para la configuración de desplazamiento de extracción entre los valores 0 y 4 en 20 pasos para datos de mala calidad.

resultados óptimos, en el caso de la figura 5.13 el error comienza en 0 y varía lentamente hasta llegar al parámetro de 19 metros. Entonces, el error empieza a incrementarse.

La horizontalidad en el comienzo puede tener dos razones. Primero, puede deberse a que el rango elegido ofusca el error presente en valores menores a 19 metros. Segundo, puede deberse a que la imprecisión de los datos naturalmente formen un radio más cercano al óptimo y incrementarlo no tiene tanto efecto en el error si no ha pasado el óptimo.

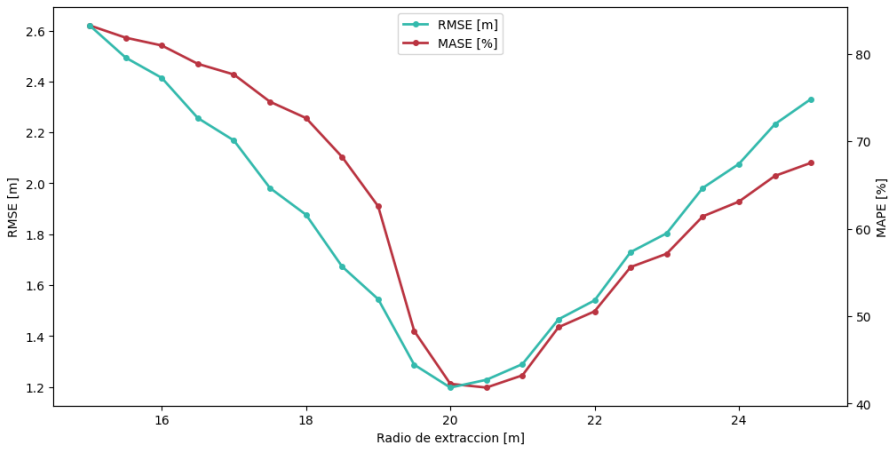


Figura 5.12: Ejemplo de visualización de *RMSE/MAPE* para la configuración de radio de extracción entre los valores 15 y 25 en 20 pasos para datos ideales.

De todos modos, existe una alta sensibilidad a la calidad de los datos por parte de esta configuración, evidenciando un claro óptimo, distinto para cada conjunto de datos, 20, 19 y 18 para datos ideales, impreciso y de mala calidad respectivamente, mientras que los datos están diseñados para ser óptimos en un radio de 20 metros. Esto contradice el hábito de especificar el radio solo mediante la especificación técnica de la pala para tener el mejor resultado desde *MineTiler* y prefiriendo ser definidos en base a la calidad de los datos.

5.2.5. Temporal

La última visualización corresponde a la variable horizontal temporal, mostrando cómo varía el error a través del periodo definido haciendo uso del cálculo acumulativo de *MineTiler*. Esta visualización aporta valor al ser parte del caso de uso principal de *MineTiler* para su despliegue en *TIMA*. Con dicha visualización se podría tener un constante seguimiento de cómo ha variado la extracción del material el día anterior o en algún periodo pasado. También podría ser utilizado para levantar alertas en caso de que se dispare el error en algún horario en particular, evidenciando el mal funcionamiento de alguno de los componentes de *TIMA* o de los datos del cliente. Como por ejemplo, perdida de conexión con la base de datos, mal ingreso de las topografías actualizadas del usuario o fallas en *LHS*, etc.

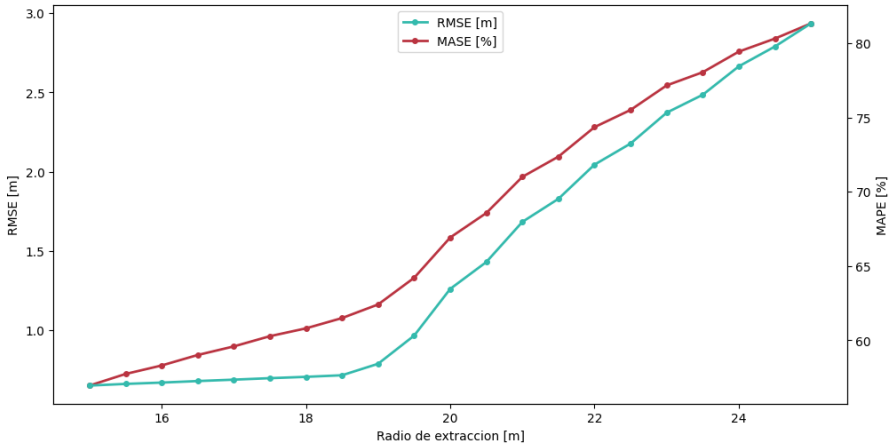


Figura 5.13: Ejemplo de visualización de *RMSE/MAPE* para la configuración de radio de extracción entre los valores 15 y 25 en 20 pasos para datos imprecisos

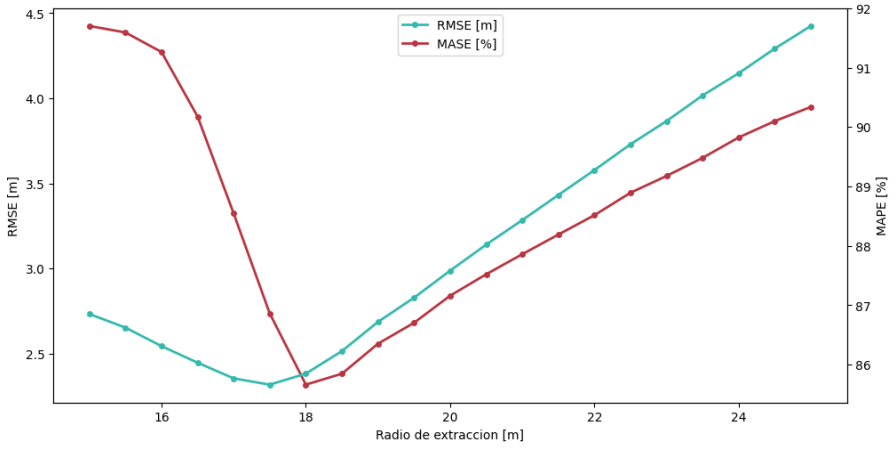


Figura 5.14: Ejemplo de visualización de *RMSE/MAPE* para la configuración de radio de extracción entre los valores 15 y 25 en 20 pasos para datos de mala calidad.

Se tiene mejor entendimiento de los horarios donde realmente la topografía está cambiando debido a los movimientos de las palas y los momentos en los que se detiene. Se mencionó anteriormente que las palas diseñadas para estas pruebas se mueven hacia el frente de extracción, se detienen, se vuelven a mover para extraer más material y se vuelven a detener. Por lo tanto, existe una discrepancia entre que la topografía se modifica cuando la pala recién se mueve hacia un frente de extracción, como son las pendientes en los gráficos de línea, mientras que en la realidad, la extracción de material se realiza al momento en el que la pala se detiene para cargar los camiones.

De todos modos, la visualización cumple con demostrar como, durante el periodo definido, el algoritmo de fresado se va aproximando a la nueva topografía real ingresada y cualquier anomalía cambiaría drásticamente los resultados.

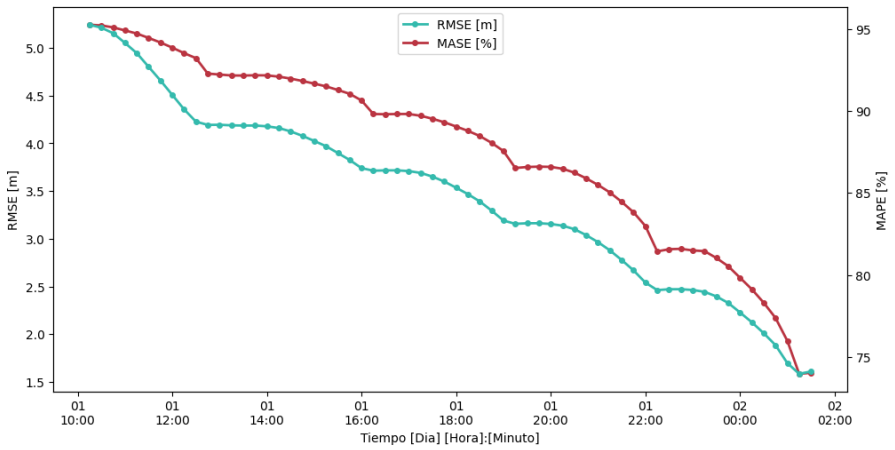


Figura 5.15: Ejemplo de visualización de *RMSE/MASE* para el caso de uso acumulativo del algoritmo de fresado cada 15 minutos para el periodo del 1 de Marzo del 2023 a las 10:00 hrs al 2 de Marzo del 2023 a las 02:00 hrs utilizando datos imprecisos.

5.2.6. Conclusiones acerca de los resultados

Por parte de las visualizaciones de *MSE*, *MAE* y *R2*, para la mayoría de los casos los gráficos eran bastante similares y entregaban la misma información, sin embargo, como la interpretación de dichas escalas es más difícil, el análisis de los resultados se vuelve más complejo. Es probable que los datos sintéticos no sean lo suficientemente sofisticados para demostrar el potencial de dichos indicadores, especialmente en las ventajas específicas de cada uno.

Concluyendo el análisis de los resultados de visualizaciones de datos, no tan solo se identifican patrones en como los datos afectan el error en el resultado final, si no que también desafía el entendimiento actual del efecto de estas configuraciones para el algoritmo de fresado. Por ejemplo, el valor por defecto del percentil parece estar incrementando el error cuando

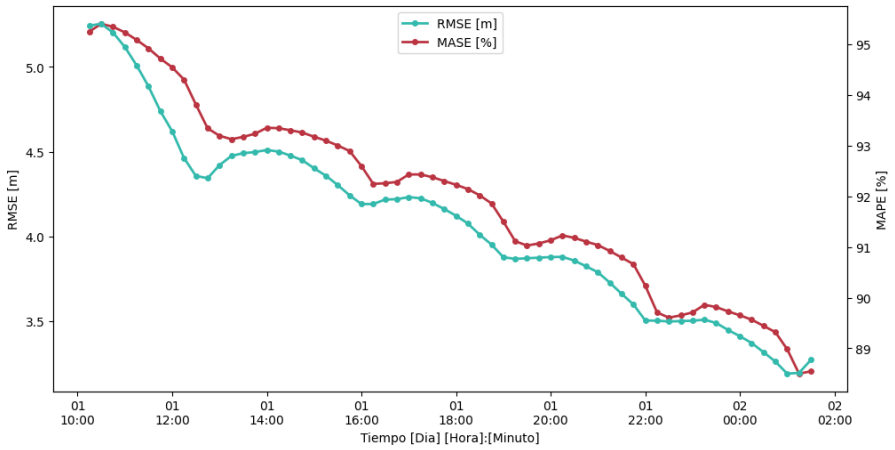


Figura 5.16: Ejemplo de visualización de *RMSE/MASE* para el caso de uso acumulativo del algoritmo de fresado cada 15 minutos para el periodo del 1 de Marzo del 2023 a las 10:00 hrs al 2 de Marzo del 2023 a las 02:00 hrs utilizando datos de mala calidad.

no existen anomalías y anteriormente esto no era fácil de notar. Esto le permite al equipo de calibración tener mejor conocimiento de cuál configuración alterar dependiendo de las propiedades de la data y poder evidenciar cómo estos parámetros generan una mejor representación de la realidad.

Cabe destacar que debido a que en la comparación se consideran ambos los frentes de extracción y la base del banco, los cambios en el error requieren más análisis, especialmente de manera visual, para poder identificar en donde específicamente se produce dicho error.

5.3. Mallas geométricas

En esta última sección de resultados, se analizan las visualizaciones de datos en mallas geométricas 3D en formato *OBJ* con variable de color representando secuencia de parámetro de configuración y diferencia en elevación con respecto a una malla geométrica de referencia. Estos son resultados de la ejecución de métodos descritos en 4.2.3.

Primero se evaluaron las mallas coloreadas por diferencia en valor Z y luego por parámetro de configuración. Para el primer caso se seleccionaron 3 configuraciones relevantes considerando la diferencia de sus efectos en los conjuntos de datos de palas utilizados los cuales también son parte de la visualización y están coloreados por tiempo. Para el caso de secuencia de parámetros de configuración, solo se mostrara el caso de mínimo de puntos en celda, ya que fallan en utilizar el espectro amplio del color, prevaleciendo solo un extremo representante de la primera ejecución.

La malla geométrica visible es una combinación entre la extracción generada por el al-

goritmo de fresado y aquella de referencia. No toda la malla está coloreada, solo aquellos valores que contienen diferencias entre ambos archivos.

5.3.1. Visualización por diferencia en valor Z

Como es mencionado en la sección 1.2, se colorean ciertas celdas de la malla geométrica en base a la diferencia en valor Z con el archivo de control. Para este caso se utiliza un mapa de calor con un espectro más notorio de 5 colores debido a la sensibilidad de la escala de elevación.

5.3.1.1. Percentil

Para el caso de la configuración de percentil en la figura 5.17 el resultado es bastante dependiente del tipo de dato utilizado. En la figura 5.17 (a) la superficie donde trabaja la pala cambia levemente en la base como se ve por el coloreado verde. Esto representa una superficie perfecta en términos de base considerando el archivo de referencia. Sin embargo, se ve que está considerando las diferencias de los bordes que es mucho mayor a la diferencia en la base del banco, perdiendo precisión.

Utilizando los datos imprecisos, la mayor diferencia está en la base del banco coloreada de rojo. La visualización en la figura 5.17 (b) se nota que el efecto del algoritmo de fresado exagero la extracción en la superficie y creó agujeros profundos en la base.

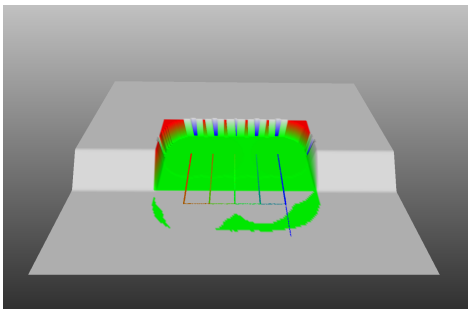
En la última visualización 5.17 (c) se utilizan datos erróneos en el algoritmo y se ve una deformación inconsistente en la superficie del banco que en la figura 5.17 (b). Sin embargo, no se distingue bien si es que las diferencias en el piso del banco son mayores en el caso de la figura 5.17(b) ó 5.17(c). La escala utilizada para colorear es lineal, por lo tanto, se pierde precisión al calcular diferencias pequeñas debido a la diferencia en los bordes de la topografía.

5.3.1.2. Mínimo de puntos en celda

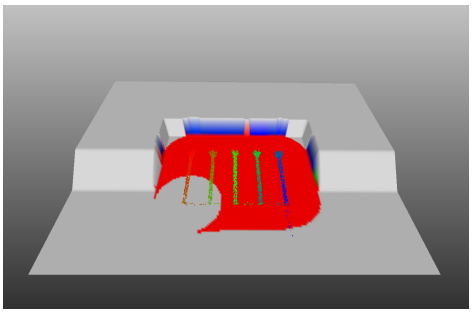
Siguiente, es la configuración de mínimo de puntos en celda en la figura 5.18. En esta oportunidad, existen ciertas diferencias en el frente de extracción como se puede notar en los tintes rojos que reducen la precisión del coloreado en la superficie del banco. Sin embargo, es el caso que presenta menores errores visuales como topografía resultante.

Para los datos erróneos en la figura 5.18 (c), la superficie se ve verde, parecido a los datos ideales en la figura 5.18 (c). Mientras tanto, en la imagen 3D se visualizan errores en la superficie del banco quedando varios agujeros a lo largo del área de extracción. Se debería usar una distinta escala al colorear las diferencias para que se aproveche mejor el espectro del coloreado.

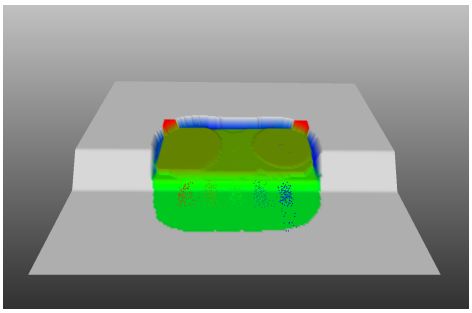
Ya que en la figura 5.18 (b) no hubo mayores diferencias en los bordes de la topografía, se nota mejor precisión en la base del banco. Destacando las diferencias que tiene la base



(a) Datos Ideales.



(b) Datos Imprecisos.



(c) Datos Erróneos.

Figura 5.17: Visualización coloreada por diferencia en elevación de percentil.

con la topografía de control. También, en donde la densidad de los puntos es mayor, es decir, cuando la pala está detenida, el efecto es distinto y el error se reduce.

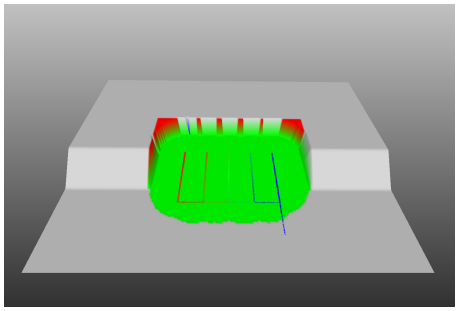
5.3.1.3. Radio

Finalmente para las visualizaciones de datos con diferencias en valor de elevación en la figura 5.19, se ilustran los efectos de la configuración de radio de extracción.

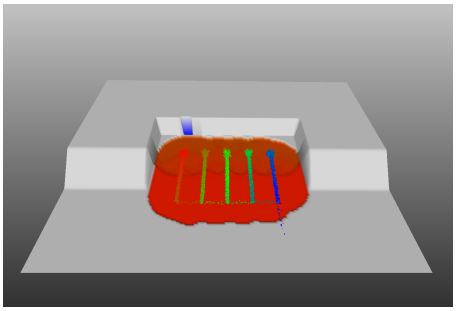
En esta oportunidad el cambio en altura es más evidente ya que se puede ver el archivo de referencia coloreado en los alrededores de la extracción debido a que el radio utilizado es mayor para el que la topografía está diseñada.

El problema de distinguir el efecto de la generación realizada por el algoritmo de fresado se puede visualizar nuevamente ya que los colores son bastante distintos y en este caso no se logra representar bien el hecho de que los datos erróneos están mucho más por debajo de la superficie real. El cálculo de la superficie no ha cambiado en este caso, solo el radio de extracción. Por lo tanto, la superficie del banco tiene un gran efecto en el coloreado.

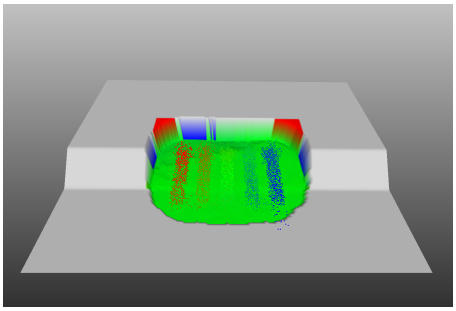
No obstante, visualmente la diferencia horizontal de donde termina lo extraído por el algoritmo de fresado es notorio aunque este sea solo un color sólido. El ángulo de la topografía de control causa un tinte verde, representando bien donde comienza este ángulo en el archivo de referencia. Por lo tanto, se puede diferir la altura de lo real comparado con la superficie



(a) Datos Ideales.



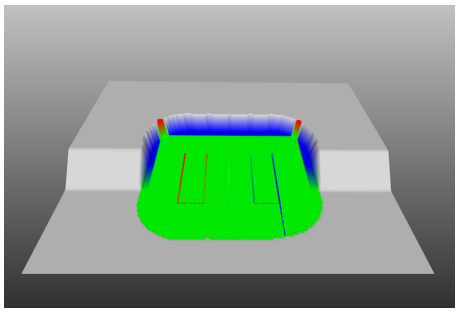
(b) Datos Imprecisos.



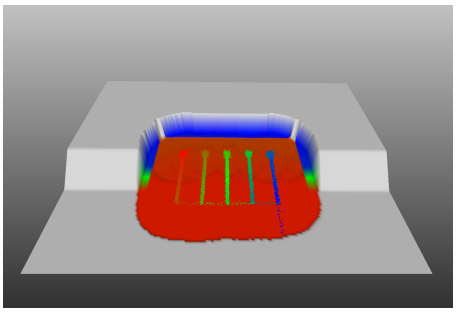
(c) Datos Erróneos.

Figura 5.18: Visualización coloreada por diferencia en elevación de mínimo de puntos en celda.

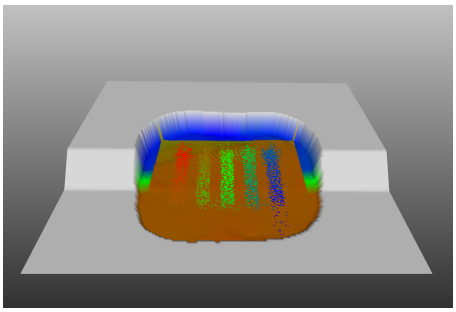
plana generada por *MineTiler*.



(a) Datos Ideales.



(b) Datos Imprecisos.



(c) Datos Erróneos.

Figura 5.19: Visualización coloreada por diferencia en elevación de radio de extracción.

5.3.2. Visualización coloreada por de parámetro de configuración

Los resultados de las visualizaciones por valor de parámetro de configuración no aprovechan bien el espectro de color utilizado ya que el cambio que se colorea ocurre dentro de pocas celdas, con la excepción de la configuración del radio. En la figura 5.21 podemos notar que la mayor parte de la malla esta coloreada de color azul debido a que es la primera ejecución realizada, por lo tanto, no se distingue bien cual es el efecto de cambiar el valor de configuración utilizando este tipo de coloreado.

Esto se debe a que la mayor parte del efecto que cambia la topografía es en una primera ejecución. Después de esto, queda poco espacio para hacer uso del espectro del mapa de calor como se muestra en la figura 5.20. Incluso si no se coloreara tanto la superficie, en el tinte rojo no se alcanza a distinguir cómo cambia la malla geométrica cuando cambian los parámetros.

Este resultado se repite en las demás configuraciones. En el caso del percentil, incluso no se divisa ningún otro tinte aparte de la superficie azul.

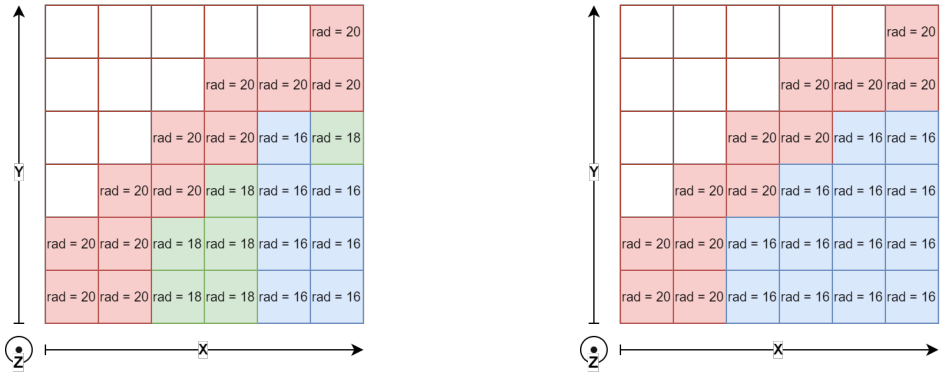
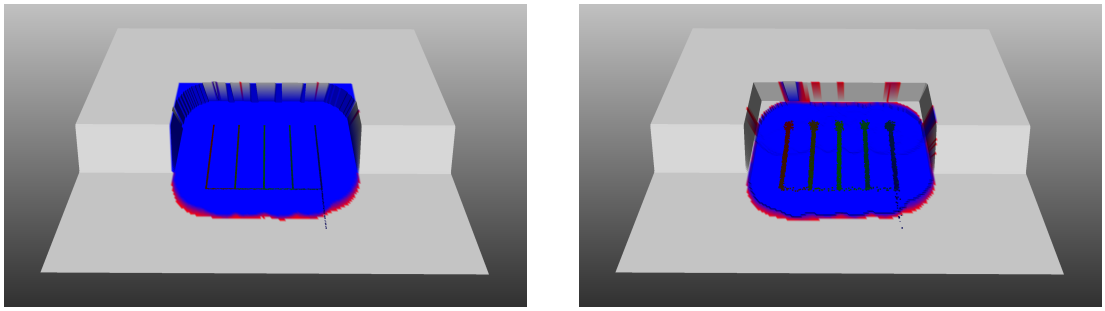
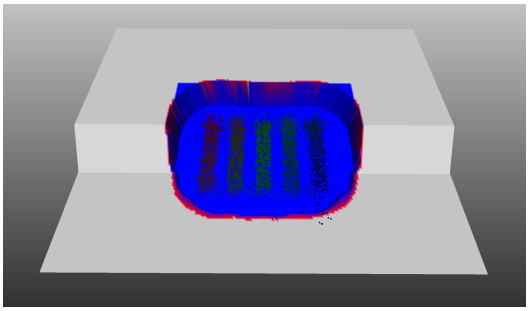


Figura 5.20: Ilustración de coloreado por parámetro de configuración. En la izquierda está el esperado utilizando más del espectro. En la derecha está el caso real donde los resultados pasan de un extremo del espectro al otro en pocas celdas.



(a) Datos Ideales.

(b) Datos Imprecisos.



(c) Datos Erróneos.

Figura 5.21: Visualización coloreada por parámetro de configuración de mínimo de puntos en celda.

5.3.3. Conclusiones acerca de los resultados

Por parte de las visualizaciones de *MSE*, *MAE* y *R2*, para la mayoría de los casos los gráficos eran bastante similares y entregaban la misma información, sin embargo, como la interpretación de dichas escalas es más difícil, el análisis de los resultados se vuelve más complejo. Es probable que los datos sintéticos no sean lo suficientemente sofisticados para demostrar el potencial de dichos indicadores, especialmente en las ventajas específicas de cada uno.

Concluyendo el análisis de los resultados de visualizaciones de datos, no tan solo se identifican patrones en como los datos afectan el error en el resultado final, si no que también desafía el entendimiento actual del efecto de estas configuraciones para el algoritmo de fresado. Por ejemplo, el valor por defecto del percentil parece estar incrementando el error cuando no existen anomalías y anteriormente esto no era fácil de notar. Esto le permite al equipo de calibración tener mejor conocimiento de cuál configuración alterar dependiendo de las propiedades de la data y poder evidenciar cómo estos parámetros generan una mejor representación de la realidad.

Cabe destacar que debido a que en la comparación se consideran ambos los frentes de extracción y la base del banco, los cambios en el error requieren más análisis, especialmente de manera visual, para poder identificar en donde específicamente se produce dicho error.

Por otro lado, para las visualizaciones geométricas, los resultados son inconclusos. En la mayoría de los casos cuando *MineTiler* extrae de manera vertical es de menor magnitud que lo extraído en los frentes de extracción. Por lo tanto, los detalles de la superficie coloreada no son visibles, nuevamente dependiendo solo de la visualización de la malla geométrica para evaluar el resultado como lo hace el equipo actualmente.

En el caso de coloración por valor de parámetro, la superficie se pinta de un solo color mayoritariamente debido a la primera ejecución de fresado como se ilustra en la figura 5.20. Por lo tanto, es difícil distinguir el efecto al cambiar los parámetros de manera paulatina.

Debido a la escala lineal utilizada, si se busca tener un buen entendimiento del efecto del algoritmo, especialmente en la superficie de la topografía, se debería separar el análisis de lo extraído horizontalmente versus verticalmente para hacer uso del espectro del mapa de calor. Esto también aplica para las visualizaciones de métricas de error en cierto grado aunque solo aporten valor para ver los efectos del cambio de las configuraciones. También se propone probar distintas escalas, por ejemplo, logarítmicas, para utilizar mejor el espectro.

Capítulo 6

Conclusiones

En la última sección de esta investigación, se discute cuáles fueron los objetivos alcanzados, cuáles tomaron más trabajo y las situaciones que ocurrieron a lo largo de su avance que cambiaron el enfoque. Se describe también como la solución propuesta ataca el problema de fondo al instalar *MineTiler*, pasando por su automatización y las distintas visualizaciones de los resultados de esta. Dadas estas visualizaciones se discute el impacto que podrían tener si es que fuesen utilizadas para el propósito que fueron diseñadas, es decir, agilizar las labores del equipo de calibración al momento de instalar el software, durante el proceso de estudiar los datos y el efecto del algoritmo de fresado en las topografías de un cliente. Se destacan las lecciones aprendidas a lo largo del trabajo realizado. Y finalmente, se aborda el trabajo a futuro que se podría realizar para mejorar la solución propuesta.

Parte de los objetivos involucran el contacto constante con el equipo que instala *MineTiler*, incluyendo iterar la solución en base a su participación en ver los resultados de lo propuesto. Por otro lado, parte de la solución podría ser validada mediante su uso con datos reales de las faenas sin comprometer la privacidad. Por ejemplo, los gráficos representan solo el trabajo realizado por *MineTiler*, sin comprometer fotografías o datos geoespaciales de un cliente. Sin embargo, y debido al término de contrato entre el estudiante y *TIMining*, no tan solo se perdió la prioridad de atacar este problema si no también el contacto con el equipo involucrado y las herramientas propias que tenía *TIMining* para llevar a cabo ciertos deseables como por ejemplo el uso de su visualizador propio o la recopilación de imágenes de *TIMA* en acción. Por lo tanto, el último objetivo fue cambiado.

De todos modos, el diseño de la solución propuesta se pensó para ser desarrollada de manera independiente por el estudiante en caso de que existiera una solución más expedita dentro del equipo de calibración o cambiará la lógica de cómo se instala *MineTiler*, considerando que incluso el uso de la API podría haber cambiado a lo largo de esta memoria.

Previo a la aplicación en sí, está la generación de datos sintéticos para sustituir la necesidad de datos reales que comprometan la privacidad de los clientes. Por el lado de las topografías, aíslan bien el efecto de fresado a un solo banco, pero en términos de escala, las faenas reales tienen una mayor cantidad de baldosas que procesar. Para mejorar estos datos se propone utilizar volúmenes más similares a lo visto en procesos de extracción reales. Por otro lado, el *LHS* encargado de entregar los datos *GPS* de pala cumplieron con su objetivo

de responder como *MineTiler* espera para poder responder a las consultas. Sin embargo, el movimiento era limitado y se podría estudiar mejor cuales son los movimientos de las palas y generar algo más cercano a como trabajan las palas. Como por ejemplo, los tiempos que se detienen a extraer material o las velocidades en que se mueven.

Por parte de la solución en sí y pasando a los objetivos concretos, en primera instancia está la automatización de *MineTiler* que fue en parte realizada en conjunto con el equipo de desarrollo de *MineTiler*. La automatización se logró con éxito y es el pilar fundamental de la mayoría de los resultados de este documento. Ya que no se utilizaron instancias o datos reales, la limpieza de los datos una vez hechas las consultas es más completa de lo que se planeaba, ya que elimina gran parte de los datos intermedios al cambiar las configuraciones. Se despriorizo también el análisis del tiempo requerido por dicha solución debido a que el mayor cuello de botella sería una instancia real de *LHS*, a lo cual no se tuvo acceso durante la generación de resultados.

Como segundo objetivo está comparar las geometrías utilizando indicadores que explicitan el error de la simulación de la topografía que en concreto consiste de los distintos gráficos y las métricas de error implementadas. Estos gráficos fueron realizados con éxito no tan solo por parte de su desarrollo sino también en las hipótesis que se buscaba responder utilizando las visualizaciones de datos en gráficos de línea presentadas.

El aporte de las métricas *MAE*, *MSE* y *R2* fue limitado debido a que los datos sintéticos de prueba generaban resultados muy similares con aquella visualización principal de *RMSE* y *MASE*. No se logró aprovechar las ventajas de dichos indicadores con los datos presentes. Por lo tanto, queda inconcluso si es que aportan valor al agilizar el proceso de calibración o no. Sin embargo, considerando los resultados de la visualización principal, se desafían los conocimientos preconcebidos sobre el efecto de las configuraciones y sus valores por defecto. Dicho conocimiento es de gran valor para futuras implementaciones. También, cabe destacar que el aporte de valor de estas visualizaciones aplica para todas las configuraciones evaluadas.

Por otro lado, las visualizaciones geométricas requieren más trabajo debido a que en la mayoría de los casos el coloreado no aporta gran valor o entendimiento del comportamiento del algoritmo de fresado. Esto debido a las distintas escalas entre los frentes de extracción y la superficie del banco. Se propone utilizar distintas escalas para aprovechar mejor el espectro del coloreado e incorporar transparencias para mostrar la topografía de referencia además de la simulada.

Las lecciones aprendidas durante el trabajo realizado consisten principalmente en entender el problema que afrontaba el equipo de calibración. Fue una preocupación desde el equipo de desarrollo la cantidad de tiempo que se invertía en poder entender el algoritmo de fresado, entonces comenzaron una serie de discusiones para profundizar en cómo corregir eso ya que datos erróneos o deformaciones en la topografía tomaron un largo tiempo en ser corregidas, usualmente generando desconfianza en el cliente. Considerando esto se sabía que si solo se solucionaba el problema para el cliente actual, el problema continuaría si no se entendía a cabalidad y se enfrentaba de una manera escalable para cuando existan más clientes.

Finalmente, el trabajo a futuro a realizar consiste principalmente en dividir el estudio de frentes de extracción con la superficie del banco minero. Esto es debido a que ambos

son problemas con escalas distintas. Esto requiere desarrollo en identificar dichos espacios en la topografía, lo cual se podría realizar calculando las normales de cada celda y haciendo una máscara para considerar sólo los resultados son normales en algún ángulo, por sobre el banco o en la superficie por donde se mueve la pala. También, buscar mejores escalas para el coloreado de las mallas geométricas y utilizar transparencias para apoyar la identificación de diferencias, especialmente en la base del banco.

Sin embargo, por parte de los objetivos alcanzados, se considera un éxito el poder tener una solución que genera más entendimiento del funcionamiento de *MineTiler* para su óptimo uso en *TIMA*.

Bibliografía

[1] Mohamed Abdelaziz, Banan Jamil Awrahman, Chia Aziz Fatah, and Mzhda Yasin Hamaamin. A review of the role and challenges of big data in healthcare informatics and analytics. *Computational Intelligence and Neuroscience*, 2022:5317760, 2022.

[2] Abbas Aghajani Bazzazi, Morteza osanloo, and Behrooz Karimi. Optimal open pit mining equipment selection using fuzzy multiple attribute decision making approach. *Archive of mining science*, 54:301–320, 04 2009.

[3] Paul J. Bartos. Is mining a high-tech industry?: Investigations into innovation and productivity advance. *Resources Policy*, 32(4):149–158, 2007.

[4] F.s Beretta, Henrique Shibata, Rodrigo Cordova, Rodrigo Peroni, Jeremias Azambuja, and João Costa. Topographic modelling using uavs compared with traditional survey methods in mining. *Rem Revista Escola de Minas*, 71:463–470, 07 2018.

[5] Michelle Blom, Adrian Pearce, and Peter Stuckey. Short-term planning for open pit mines: A review. *International Journal of Mining, Reclamation and Environment*, 33, 03 2018.

[6] Centre National d'Etudes Spatiales (CNES). Demcompare, a dem comparison tool. `https://demcompare.readthedocs.io/`, 2023.

[7] P. Hartlieb F. Sánchez. Innovation in the mining industry: Technological trends and a case study of the challenges of disruptive innovation. *Mining, Metallurgy & Exploration*, page 1385–1399, July 2020.

[8] J.L Broadhurst H.-M Stander. Understanding the opportunities, barriers, and enablers for the commercialization and transfer of technologies for mine waste valorization: A case study of coal processing wastes in south africa. *Resources Policy*, 32(4):149–158, 2007.

[9] Huma Hassan, Syed Amer Mahmood, Saira Batool, Areeba Amer, Mareena Khurshid, Hina Yaqub, Dr Ali Gill, Marina Khurshid, and Imran Imran. Generation of digital surface model (dsm) using uav/ quadcopter. *International Journal of Innovations in Science and Technology*, 2:89–107, 09 2020.

[10] Wei Hu, Kendrik Yan Hong Lim, and Yiyu Cai. Digital twin and industry 4.0 enablers in building and construction: A survey. *Buildings*, 12:2004, 11 2022.

[11] IBM. What is industry 4.0? https://www.ibm.com. https://www.ibm.com/topics/industry-4-0.

[12] Sergio Jiménez-Jiménez, Waldo Ojeda, Mariana Marcial, and Juan Enciso. Digital terrain models generated with low-cost uav photogrammetry: Methodology and accuracy. *ISPRS International Journal of Geo-Information*, 10:285, 04 2021.

[13] J. Joaquín Jara, Patricio Pérez, and Pablo Villalobos. Good deposits are not enough: Mining labor productivity analysis in the copper industry in chile and peru 1992–2009. *Resources Policy*, 35(4):247–256, 2010. Resources Policy in South America.

[14] Tahera Kalsoom, Naeem Ramzan, Shehzad Ahmed, and Masood Ur Rehman. Advances in sensor technologies in the era of smart factory and industry 4.0. *Sensors*, 20:6783, 11 2020.

[15] B. Johansson L. Abrahamsson and J. Johansson. Future of metal mining: Sixteen predictions. *Int. J. Mining and Mineral Engineering*, 1(3):304–312, 2019.

[16] MapFreeGlobalRisks. Mining 4.0, an evolving industry. https://www.mapfreglobalrisks.com/en/risks-insurance-management/article/mining-4-0-an-evolving-industry/, February 2021.

[17] José Mesa-Mingorance and Francisco Ariza-Lopez. Accuracy assessment of digital elevation models (dems): A critical review of practices of the past three decades. *Remote Sensing*, 12:2630, 08 2020.

[18] MinergyConnect. Transformación digital para la minería 4.0 en Perú. https://www.minergyconnect.pe/2022/05/transformacion-digital-para-la-mineria-4-0-en-peru/, May 2022.

[19] Julian Müller, Daniel Kiel, and Kai-Ingo Voigt. What drives the implementation of industry 4.0? the role of opportunities and challenges in the context of sustainability. *Sustainability*, 10, January 2018.

[20] Pham Duc Thang Nguyen Ngoc Minh. Tendencies of mining technology development in relation to deep mines. *Mining Science and Technology*, 4(1):16–22, 2019.

[21] Laurent Polidori and Mhamad El Hage. Digital elevation model quality assessment methods: A critical review. *Remote. Sens.*, 12:3522, 2020.

[22] Javad Shahmoradi, Elaheh Talebi, Pedram Roghanchi, and Mostafa Hassanalian. A comprehensive review of applications of drone technology in the mining industry. *Drones*, 4, 07 2020.

[23] Jan-Frederik Uhlenkamp, Karl Hribernik, Stefan Wellsandt, and Klaus-Dieter Thoben. Digital twin applications : A first systemization of their dimensions. In *Conference: 2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–8, 06 2019.

[24] J.W van der Merwe and D.C Andersen. Applications and benefits of 3D laser scanning for the mining industry. *Journal of the Southern African Institute of Mining and Metallurgy*, 113:00 – 00, 03 2013.

[25] Mining Weekly. Timining brings situational awareness to mining operations from any smart device, anytime, anywhere. https://www.miningweekly.com/article/timining-brings-situational-awareness-to-mining-operations-from-any-smart-device-anytime-anywhere-2021-03-29, March 2021.

[26] John P. Wilson and John C. Gallant. *Terrain Analysis: Principles and Applications*. Wiley, New York, 1997.

[27] Li Da Xu, Eric L. Xu, and Ling Li. Industry 4.0: state of the art and future trends. *International Journal of Production Research*, 56(8):2941–2962, 2018.

Anexo

Código Fuente

```
import os
import subprocess
import shutil
import json
import bson
import time
import requests
import datetime as dt
import numpy as np
from dateutil.parser import parse
from pymongo import MongoClient

import dtm



# MineTiler
# PabloS - Feb 2023
#
# Python script which provides an enviroment to simplify and automate
# certain interactions with the MineTiler API.
#
# Key name in files variable definitions CANNOT be changed, as they are
# MineTilers way to receive said file.
#
# All file uploads are made using POST multipart/formdata.



######################### Change as you see fit ##############################
## Server configurations for script
DATE = "2023-01-01T10:00:00.000Z"
START_DATE = "2023-01-01T10:00:00.000Z"
END_DATE = "2023-01-02T02:00:00.000Z"
PASTE_PATH = "D://Users//TIM//Documents//TESIS"

SERVER_HOST = "localhost"
SERVER_PORT = "8500"
MONGO_URI = 'mongodb://localhost:27017'
MONGO_DB = 'test_cc6908'
GEOM_SERVER_FOLDER = 'D:/DeltaRepo/terraformer/geometry-server'

## Tiles Configuration parameters
CELLSIZE = 1.0
TILESIZE = 200

## Digger Configuration parameters
HEIGHTTHRESHOLD = 50
PERCENTILE = 10
MINPOINTSINCELL = 5

##############################################################################
```

Parte del *script* **minetiler_automation.py**.

```python
# Build post variables
headers = {'Accept': 'application/json'}
minetiler_url = "http://" + SERVER_HOST + ":" + SERVER_PORT + "/jobs"

# Mongo Functions
client = MongoClient(MONGO_URI)
db = client[MONGO_DB]

# Convert bdtm to dtm using bdtm2obj.exe
FNULL = open(os.devnull, 'w')
EXE = "D:\\Users\\TIM\\Documents\\TESIS\\geometry-calculators\\bdtm2obj.exe"

def turnFilesInFolderToDTM(folder):
    for file in os.listdir(os.path.join(os.getcwd(), folder + "/bdtms/")):
        in_file_path = folder + "/bdtms/" + file
        out_file_path = folder + "/dtms/" + os.path.splitext(file)[0] + ".dtm"
        args = EXE + " -d " + in_file_path + " " + out_file_path
        print(args)

        subprocess.call(args, stdout=FNULL, stderr=FNULL, shell=False)


## Change value of config key in mongo
def changeVal(config, key, value):
    configQuery = { '_id': config }
    old_val = db['configuration'].find_one(configQuery)

    log_str = 'Swapped Mongo key: {0}, with curr val: {1} to val: {2}'.format(key, old_val[key], va
    print(log_str)

    db['configuration'].update_one(configQuery, {'$set': {key: value}})

    return old_val[key]

## Change value of config key in mongo
def changeShovelVal(shovel_id, key, value):
    configQuery = { '_id': shovel_id }
    old_val = db['shovels'].find_one(configQuery)

    log_str = 'Swapped Mongo key: {0}, with curr val: {1} to val: {2}'.format(key, old_val[key], va
    print(log_str)

    db['shovels'].update_one(configQuery, {'$set': {key: value}})

    return old_val[key]


## Change value of config key in mongo
def changeSmoothingVal(key, value):
    configQuery = {'_id': 'diggerConfiguration'}
    old_val = db['configuration'].find_one(configQuery)
```

Parte del *script* **minetiler_automation.py**.

```python
    old_val = db['configuration'].find_one(configQuery)

    log_str = 'Swapped Mongo key: {0}, with curr val: {1} to val: {2}'.format(key, old_val['smoothi
    print(log_str)

    old_val['smoothingConfiguration'][key] = value
    smoothing_val = { "$set": { 'smoothingConfiguration': old_val['smoothingConfiguration'] } }

    db['configuration'].update_one(configQuery, smoothing_val)

    return old_val['smoothingConfiguration']


def ClearCollections():
    db['tiledTopographies'].drop()
    db['topographies'].drop()


# MineTiler API
## http://{SERVER_HOST}:{SERVER_PORT}/jobs/{job}
def statusFromRequest(request):
    # Get JobID from input POST request
    job_id = request['jobId']
    url = minetiler_url + "/status" + "/" + job_id

    print("Requesting status: ", url)
    response = requests.post(url, headers=headers).json()

    # Wait until task has finished and print response JSON
    while response['status'] != "Finished":
        if response['status'] == "Created":
            print("Task created, not processing: ", url)
        elif response['status'] == "Failed":
            return "Task failed."
        else:
            print("Processing, waiting for finished: ", url)

        time.sleep(1)
        response = requests.post(url, headers=headers).json()

    print("Finished request.", '\n')
    return response


## http://{SERVER_HOST}:{SERVER_PORT}/jobs/newtopography/{DATE}
def newTopography(filename, date):
    url = minetiler_url + "/newtopography" + "/" + date
    files = {'file1': open(filename,'rb')}

    print("Requesting newtopography: ", url)
    return requests.post(url, files=files, headers=headers).json()
```

Parte del *script* **minetiler_automation.py**.

```python
# REQUESTS
## http://{SERVER_HOST}:{SERVER_PORT}/jobs/{request_job}/{DATE}
def iRequestJob(request_job, date):
    url = minetiler_url + "/" + request_job + "/" + date
    print("Requesting " + request_job + ": ", url)
    return url

## http://{SERVER_HOST}:{SERVER_PORT}/jobs/tiledtopography/{frequency}/{DATE}
def tiledTopography(date, fresh=True):
    if fresh is True:
        url = iRequestJob("freshtiledtopography", date)
    else:
        url = iRequestJob("tiledtopography/", date)

    return requests.get(url, headers=headers).json()


def getBDTMFile(folder, filename):
    for name in os.listdir(GEOM_SERVER_FOLDER):
        if name == folder:
            for file in os.listdir(GEOM_SERVER_FOLDER + "/" + name + "/output"):
                if os.path.splitext(file)[1] == ".bdtm":
                    copy_path = GEOM_SERVER_FOLDER + "/" + name + "/output/" + file
                    shutil.copyfile(copy_path, PASTE_PATH + "/bdtms/" + filename)


def generateControlDTMs():
    req_date = parse(START_DATE)
    end_date = parse(END_DATE)

    json_output = {}

    json_output['control_topo'] = statusFromRequest(newTopography("actualinitial.obj", req_date.iso
    json_output['control_dtm'] = statusFromRequest(tiledTopography(end_date.isoformat()))

    folder_name = json_output['control_dtm']['jobType'] + "-" + json_output['control_dtm']['jobId']
    out_f_name = "control_dtm" + ".bdtm"
    getBDTMFile(folder_name, out_f_name)


    return json_output


def generateConfigDTMs():
    req_date = parse(START_DATE)
    end_date = parse(END_DATE)

    json_output = {}

    def ProcessResult(key, config, val):
        keyname = config + "_" + str(round(val, 5))
```

Parte del *script* **minetiler_automation.py**.

```python
    def ProcessResult(key, config, val):
        keyname = config + "_" + str(round(val, 5))
        orig_val = changeVal(key, config, val)
        json_output[keyname] = statusFromRequest(tiledTopography(end_date.isoformat()))
        changeVal(key, config, orig_val)

        folder_name = json_output[keyname]['jobType'] + "-" + json_output[keyname]['jobId']
        out_f_name = keyname + ".bdtm"
        getBDTMFile(folder_name, out_f_name)

    def ProcessTemporalResult():
        i_date = parse(START_DATE)
        i = 0
        while i_date < end_date:
            keyname = "temporal_" + str(i)
            json_output[keyname] = statusFromRequest(tiledTopography(i_date.isoformat(), fresh=Fals
            folder_name = json_output[keyname]['jobType'] + "-" + json_output[keyname]['jobId']
            out_f_name = keyname + ".bdtm"
            getBDTMFile(folder_name, out_f_name)

            i_date = i_date + dt.timedelta(minutes=15)
            i = i + 1

    def ProcessShovelResult(shovel_id, config, val):
        keyname = config + "_" + str(round(val, 5))
        orig_val = changeShovelVal(shovel_id, config, val)
        json_output[keyname] = statusFromRequest(tiledTopography(end_date.isoformat()))
        changeShovelVal(shovel_id, config, orig_val)

        folder_name = json_output[keyname]['jobType'] + "-" + json_output[keyname]['jobId']
        out_f_name = keyname + ".bdtm"
        getBDTMFile(folder_name, out_f_name)

    def ProcessShovelRadiusResult(shovel_id, outer, inner):
        keyname = "radius_" + str(round(outer, 5)) + "_" + str(round(inner, 5))
        orig_outer_val = changeShovelVal(shovel_id, 'outerRadius', outer)
        orig_inner_val = changeShovelVal(shovel_id, 'innerRadius', inner)
        json_output[keyname] = statusFromRequest(tiledTopography(end_date.isoformat()))
        changeShovelVal(shovel_id, 'outerRadius', orig_outer_val)
        changeShovelVal(shovel_id, 'innerRadius', orig_inner_val)

        folder_name = json_output[keyname]['jobType'] + "-" + json_output[keyname]['jobId']
        out_f_name = keyname + ".bdtm"
        getBDTMFile(folder_name, out_f_name)


    ## Process results
    # minPointsInCell changes
    for i in np.linspace(60, 100, 21):
        ProcessResult('diggerConfiguration', 'minPointsInCell', i)

    # percentile changes
```

Parte del *script* **minetiler_automation.py**.

```
    for i in np.linspace(0, 100, 21):
        ProcessResult('diggerConfiguration', 'percentile', i)

    # extractionOffset changes
    for i in np.linspace(0, 4, 21):
        ProcessShovelResult('0', 'extractionOffset', i)

    # Same inner and outer radius changes
    for i in np.linspace(15, 25, 21):
        ProcessShovelRadiusResult('0', i, i)

    return json_output


def generateTemporalDTMs():
    req_date = parse(START_DATE)
    end_date = parse(END_DATE)

    json_output = {}

    def ProcessTemporalResult():
        i_date = parse(START_DATE)
        i = 0
        while i_date < end_date:
            keyname = "temporal_" + str(i)
            json_output[keyname] = statusFromRequest(tiledTopography(i_date.isoformat(), fresh=Fals
            folder_name = json_output[keyname]['jobType'] + "-" + json_output[keyname]['jobId']
            out_f_name = keyname + ".bdtm"
            getBDTMFile(folder_name, out_f_name)

            i_date = i_date + dt.timedelta(minutes=15)
            i = i + 1

    ProcessTemporalResult()

    return json_output


# Script execution
if __name__ == "__main__":
    # Make script ejecution folder
    folder_name = "minetiler_automation" + dt.datetime.now().strftime("%d%m%Y-%H%M%S")
    PASTE_PATH = os.path.join(os.getcwd(), folder_name)
    os.mkdir(PASTE_PATH)
    if not os.path.exists(PASTE_PATH + "/dtms"):
        os.mkdir(PASTE_PATH + "/dtms")
    if not os.path.exists(PASTE_PATH + "/bdtms"):
        os.mkdir(PASTE_PATH + "/bdtms")

    json_output = {}

    print("Generating control BDTM")
```

Parte del *script* **minetiler_automation.py**.

```
# Script execution
if __name__ == "__main__":
    # Make script ejecution folder
    folder_name = "minetiler_automation" + dt.datetime.now().strftime("%d%m%Y-%H%M%S")
    PASTE_PATH = os.path.join(os.getcwd(), folder_name)
    os.mkdir(PASTE_PATH)
    if not os.path.exists(PASTE_PATH + "/dtms"):
        os.mkdir(PASTE_PATH + "/dtms")
    if not os.path.exists(PASTE_PATH + "/bdtms"):
        os.mkdir(PASTE_PATH + "/bdtms")

    json_output = {}

    print("Generating control BDTM")
    json_output.update(generateControlDTMs())
    print("Generating digger run BDTMs")
    json_output.update(generateConfigDTMs())

    # Clear DB
    print("Clearing")
    ClearCollections()

    generateControlDTMs()
    json_output.update(generateTemporalDTMs())

    # Converting BDTM to DTM
    print("Start bdtm2obj.exe")
    turnFilesInFolderToDTM(folder_name)
    print("Done using bdtm2obj.exe")

    #Writing output json file
    json_object = json.dumps(json_output, indent=4)
    with open("minetilerpy_output.json", "w") as outfile:
        outfile.write(json_object)

    # Clear DB
    ClearCollections()
```

Parte del *script* **minetiler_automation.py**.

```python
import os
import numpy as np
from dateutil.parser import parse
import datetime as dt
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pandas as pd
import time
import dtm
import usebdtm2obj


######################## Change as you see fit ##############################
# Folder containing DTM files from control_automation.py
DATA_FOLDER = "minetiler_automationRND12/"
# Data for result export
OUT_FOLDER = "results12-05-03-1835"
# Dates
START_DATE = "2023-01-01T10:00:00.000Z"
END_DATE = "2023-01-02T02:00:00.000Z"
# DTM to measure against
CONTROL_DTM = "actualfinal.dtm"

#############################################################################



## Metrics calculation
# Calculate error metrics between a contro DTM and a list of DTMs
def CalcMetrics(dtm_control, dtm_l):
    rmse_l = []
    mase_l = []
    mse_l = []
    mae_l = []
    r2_l = []

    t1_start = time.perf_counter()
    print("Started calculating metrics. Calculating for {0} DTMs".format(len(dtm_l)))
    for a_dtm in dtm_l:
        rmse_l.append(dtm_control.rmse(a_dtm))
        mase_l.append(dtm_control.mase(a_dtm))
        mse_l.append(dtm_control.mse(a_dtm))
        mae_l.append(dtm_control.mae(a_dtm))
        r2_l.append(dtm_control.r2(a_dtm))

    t1_stop = time.perf_counter()
    print("Finished calculating metrics. Time elapsed: ", t1_stop-t1_start)
    return [rmse_l, mase_l, mse_l, mae_l, r2_l]
```

Inicio del *script* **compare_dtms.py**, variables globales y calculo de métricas.

```python
# Generate RMSE & MAPE data for a configuration in range_l
def GenData(dtm_control, config, range_l):
    dtm_l = []

    for i in range_l:
        dtm_filename = DATA_FOLDER + "dtms/" + config + "_" + str(round(i,5)) + ".dtm"
        dtm_l.append(dtm.readDtm(dtm_filename, divisor=1))

    return CalcMetrics(dtm_control, dtm_l)

# Generate RMSE & MAPE data for an extraction configuration for an inner and outter radius
def GenRadiusData(dtm_control, range_outer, range_inner):
    dtm_l = []

    for i, j in zip(range_outer, range_inner):
        dtm_filename = DATA_FOLDER + "dtms/radius_" + str(round(i,5)) + "_" + str(round(j,5)) + ".dtm"
        dtm_l.append(dtm.readDtm(dtm_filename, divisor=1))

    return CalcMetrics(dtm_control, dtm_l)

# Generate RMSE & MAPE data for a sequential executions in a date interval
def GenTemporalData(dtm_control):
    dtm_l = []

    i = 1
    dtm_filename = DATA_FOLDER + "dtms/temporal_" + str(i) + ".dtm"
    while os.path.isfile(dtm_filename):
        dtm_l.append(dtm.readDtm(dtm_filename, divisor=1))

        i = i + 1
        dtm_filename = DATA_FOLDER + "dtms/temporal_" + str(i) + ".dtm"

    result = CalcMetrics(dtm_control, dtm_l)
    result.append(i)
    return result
```

Parte del *script* **compare_dtms.py**, lectura de *DTMs* de *script* de automatización.

```
def ColorizeByParam(dtm_control, config, range_1):
    dtm_diff_history = dtm.crearDtm(
        dtm_control.ncols, dtm_control.nrows,
        dtm_control.xcenter, dtm_control.ycenter,
        dtm_control.cellsize,
        dtm_control.nodata)
    last_dtm = dtm.readDtm(DATA_FOLDER + 'dtms/' + config + "_" + str(round(range_1[-1], 5)) + ".dtm")

    tolerance = 1e-3
    count = 0
    for i in range_1:
        in_filename = config + '_' + str(round(i,5))
        diff_dtm = last_dtm - dtm.readDtm(DATA_FOLDER + "dtms/" + in_filename + ".dtm")
        count = count + 1
        for i in range(diff_dtm.ncols):
            for j in range(diff_dtm.nrows):
                if abs(diff_dtm.Z[j,i]) > tolerance and abs(diff_dtm.Z[j,i] - diff_dtm.nodata) > tolerance:
                    if abs(dtm_diff_history.Z[j,i] - dtm_diff_history.nodata) < tolerance or dtm_diff_history.Z[j,i] < count:
                        dtm_diff_history.Z[j,i] = count


    out_path = OUT_FOLDER + '/objs/' + in_filename + ".obj"
    last_dtm.writeObjColorizeByValueParam(dtm_diff_history, len(range_1), out_path)

def ColorizeByRadiusParam(dtm_control, range_inner, range_outter):
    dtm_diff_history = dtm.crearDtm(
        dtm_control.ncols, dtm_control.nrows,
        dtm_control.xcenter, dtm_control.ycenter,
        dtm_control.cellsize,
        dtm_control.nodata)
    last_dtm = dtm.readDtm(DATA_FOLDER + "dtms/radius_" + str(round(range_inner[-1],5)) + "_" + str(round(range_outter[-1],5)) + ".dtm")

    tolerance = 1e-3
    count = 0
    for i in range_inner:
        in_filename = "radius_" + str(round(i,5)) + "_" + str(round(i,5))
        diff_dtm = last_dtm - dtm.readDtm(DATA_FOLDER + "dtms/" + in_filename + ".dtm")
        count = count + 1
        for i in range(diff_dtm.ncols):
            for j in range(diff_dtm.nrows):
                if abs(diff_dtm.Z[j,i]) > tolerance and abs(diff_dtm.Z[j,i] - diff_dtm.nodata) > tolerance:
                    if abs(dtm_diff_history.Z[j,i] - dtm_diff_history.nodata) < tolerance or dtm_diff_history.Z[j,i] < count:
                        dtm_diff_history.Z[j,i] = count


    out_path = OUT_FOLDER + '/objs/' + in_filename + ".obj"
    last_dtm.writeObjColorizeByValueParam(dtm_diff_history, len(range_inner), out_path)
```

Parte del *script* **compare_dtms.py**, definición de diferencias para coloración de *OBJ*.

```
def ColorizeByObjDiffs(dtm_control, last_file):
    diff_dtm = dtm_control - dtm.readDtm(last_file)
    last_dtm = dtm.readDtm(last_file)
    last_dtm.writeObjColorizeByDiff(diff_dtm, "radius_2.obj")


# Script execution
if __name__ == "__main__":

    ## Initializations
    dtm_control = dtm.readDtm(CONTROL_DTM, divisor=1)
    results_list = {}

    # Config data generation
    results_list['minPointsInCell'] = GenData(dtm_control, 'minPointsInCell', np.linspace(60, 100, 21))
    results_list['percentile'] = GenData(dtm_control, 'percentile', np.linspace(0, 100, 21))
    results_list['extractionOffset'] = GenData(dtm_control, 'extractionOffset', np.linspace(0, 4, 21))
    results_list['radius'] = GenRadiusData(dtm_control, np.linspace(15, 25, 21), np.linspace(15, 25, 21))
    results_list['temporal'] = GenTemporalData(dtm_control) # np.linspace(10, results_list[2]-1 + 10, 63)

    ## Plot definitions
    if not os.path.exists(OUT_FOLDER):
        os.mkdir(OUT_FOLDER)
    if not os.path.exists(OUT_FOLDER + '/objs'):
        os.mkdir(OUT_FOLDER + '/objs')
```

Parte del *script* **compare_dtms.py**, coloración de *OBJ* y definición de resultados de métricas.

```python
# Plot RMSE/MAPE main graph
def PlotMainConfig(config, xlabel, range_l):

    # Initializations
    data = results_list[config]
    fig, host = plt.subplots(figsize=(10,5), layout='constrained')
    ax_mase = host.twinx()

    color1, color2 = '#34BAAD', '#BA3441'

    # Labels
    host.set_xlabel(xlabel)
    if isinstance(range_l[0], dt.date):
        host.xaxis.set_major_formatter(mdates.DateFormatter('%d\n%H:%M'))

    host.set_ylabel('RMSE [m]')
    ax_mase.set_ylabel('MAPE [%]')

    host.set_zorder(ax_mase.get_zorder() + 1)
    host.patch.set_visible(False)

    # Plotting
    p1 = host.plot(range_l, data[0], color=color1, label="RMSE [m]", linewidth=2, marker='o', markersize=4)
    p2 = ax_mase.plot(range_l, data[1], color=color2, label="MASE [%]", linewidth=2, marker='o', markersize=4)

    # Legend
    host.legend(handles=p1+p2, loc='upper center')

    # Axis limits
    bleed_x = (max(range_l) - min(range_l))*0.05
    bleed_host_y = (max(data[0]) - min(data[0]))*0.05
    bleed_ax_mase_y = (max(data[1]) - min(data[1]))*0.05

    host.set_xlim(min(range_l) - bleed_x, max(range_l) + bleed_x)
    host.set_ylim(min(data[0]) - bleed_host_y, max(data[0]) + bleed_host_y)
    ax_mase.set_ylim(min(data[1]) - bleed_ax_mase_y, max(data[1]) + bleed_ax_mase_y)

    # Save resulting file
    plt.savefig(OUT_FOLDER + "/main" + config, bbox_inches='tight')
    #plt.show()
```

Parte del *script* **compare_dtms.py**, generación de gráfico de linea principal.

```python
# Plot RMSE/MAPE main graph
def PlotConfig(config, xlabel, range_l):
    # Initializations
    data = results_list[config]
    fig, host = plt.subplots(3, figsize=(10,5), layout='constrained')

    color1, color2, color3 = '#34BAAD', '#BA3441', '#C59849'

    # Labels
    host[2].set_xlabel(xlabel)
    if isinstance(range_l[0], dt.date):
        host[2].xaxis.set_major_formatter(mdates.DateFormatter('%d\n%H:%M'))
    host[0].set_ylabel('MAE [m]')
    host[1].set_ylabel('MAE [m\u00b2]')
    host[2].set_ylabel('R2 [I]')

    # Plotting
    host[0].plot(range_l, data[2], color=color1, label="MAE [m]", linewidth=2, marker='o', markersize=4)
    host[1].plot(range_l, data[3], color=color2, label="MSE [m\u00b2]", linewidth=2, marker='o', markersize=4)
    host[2].plot(range_l, data[4], color=color3, label="R2 [I]", linewidth=2, marker='o', markersize=4)

    # Legend
    host[0].legend(loc='upper center')
    host[1].legend(loc='upper center')
    host[2].legend(loc='lower center')

    # Save resulting file
    plt.savefig(OUT_FOLDER + "/" + config, bbox_inches='tight')
    #plt.show()


# Plot
def PlotTemporal(xlabel):
    # Date axis list
    date_l = []

    i_date = parse(START_DATE) + dt.timedelta(minutes=15)
    while i_date < parse(END_DATE):
        date_l.append(i_date)
        i_date = i_date + dt.timedelta(minutes=15)

    PlotMainConfig('temporal', xlabel, date_l[:-1])
    PlotConfig('temporal', xlabel, date_l[:-1])
```

Parte del *script* **compare_dtms.py**, generación de gráfico de linea secundario.

```python
# Plot
def PlotTemporal(xlabel):
    # Date axis list
    date_l = []

    i_date = parse(START_DATE) + dt.timedelta(minutes=15)
    while i_date < parse(END_DATE):
        date_l.append(i_date)
        i_date = i_date + dt.timedelta(minutes=15)

    PlotMainConfig('temporal', xlabel, date_l[:-1])
    PlotConfig('temporal', xlabel, date_l[:-1])


ColorizeByParam(dtm_control, 'minPointsInCell', np.linspace(60, 100, 21))
ColorizeByParam(dtm_control, 'percentile', np.linspace(0, 100, 21))
ColorizeByParam(dtm_control, 'extractionOffset', np.linspace(0, 4, 21))
ColorizeByRadiusParam(dtm_control, np.linspace(15, 25, 21), np.linspace(15, 25, 21))

PlotMainConfig('minPointsInCell', 'Minimo de puntos en celda', np.linspace(60, 100, 21))
PlotMainConfig('percentile', 'Percentil', np.linspace(0, 100, 21))
PlotMainConfig('extractionOffset', 'Desplazamiento de extraccion [m]', np.linspace(0, 4, 21))
PlotMainConfig('radius', 'Radio de extraccion [m]', np.linspace(15, 25, 21))

PlotConfig('minPointsInCell', 'Minimo de puntos en celda', np.linspace(60, 100, 21))
PlotConfig('percentile', 'Percentil', np.linspace(0, 100, 21))
PlotConfig('extractionOffset', 'Desplazamiento de extraccion [m]', np.linspace(0, 4, 21))
PlotConfig('radius', 'Radio de extraccion [m]', np.linspace(15, 25, 21))

PlotTemporal('Tiempo [Dia] [Hora]:[Minuto]')
```

Parte del *script* **compare_dtms.py**, generación de gráfico de linea temporal.

```python
def mse(self, dtm):
    z_l = list(filter(lambda z: z != self.nodata, self.Z.reshape(-1).tolist()))
    dtm_z_l = list(filter(lambda z: z != self.nodata, dtm.Z.reshape(-1).tolist()))

    sqr_sum = 0
    for (z,dtm_z) in zip(z_l, dtm_z_l):
        sqr_sum = sqr_sum + (z-dtm_z)**2

    return sqr_sum/len(z_l)

def rmse(self, dtm):
    return math.sqrt(self.mse(dtm))

def mae(self, dtm):
    z_l = list(filter(lambda z: z != self.nodata, self.Z.reshape(-1).tolist()))
    dtm_z_l = list(filter(lambda z: z != self.nodata, dtm.Z.reshape(-1).tolist()))

    abs_sum = 0
    for (z,dtm_z) in zip(z_l, dtm_z_l):
        abs_sum = abs_sum + abs(z-dtm_z)

    return abs_sum/len(z_l)

def mase(self, dtm):
    z_l = list(filter(lambda z: z != self.nodata, self.Z.reshape(-1).tolist()))
    dtm_z_l = list(filter(lambda z: z != self.nodata, dtm.Z.reshape(-1).tolist()))

    c=(len(z_l)-1)/len(z_l)

    top, bot = 0, 0
    for (z,dtm_z) in zip(z_l, dtm_z_l):
        top = top + abs(z-dtm_z)

    for (z,dtm_z) in zip(z_l[1:], dtm_z_l[:-1]):
        bot = bot + abs(z-dtm_z)

    return c*(top/bot)

def r2(self, dtm):
    z_l = list(filter(lambda z: z != self.nodata, self.Z.reshape(-1).tolist()))
    dtm_z_l = list(filter(lambda z: z != self.nodata, dtm.Z.reshape(-1).tolist()))

    mean_of_z = sum(z_l)/len(z_l)

    residual_sum_of_squares = 0
    total_sum_of_squares = 0
    for (z,dtm_z) in zip(z_l, dtm_z_l):
        residual_sum_of_squares = residual_sum_of_squares + (z-dtm_z)**2
        total_sum_of_squares = total_sum_of_squares + (z-mean_of_z)**2

    return 1 - (residual_sum_of_squares/total_sum_of_squares)
```

Métodos que calculan las métricas de error.

```
def writeObjColorizeByValueParam(self, diffs, curr_max, f_out):
    out=open(f_out, 'w')
    ncols=self.ncols
    nrows=self.nrows
    x0=self.xcenter
    y0=self.ycenter
    d=self.cellsize
    nodata=self.nodata

    X=np.arange(ncols*nrows).reshape(ncols, nrows)*0
    Y=np.arange(ncols*nrows).reshape(ncols, nrows)*0
    Z=self.Z

    for i in range(ncols):
        for j in range(nrows):
            X[i,j]=x0+i*d
            Y[i,j]=y0+(nrows-j-1)*d

    epsilon = 1e-3
    for i in range(ncols):
        for j in range(nrows):
            if abs(diffs.Z[j,i] - nodata) > epsilon:
                rgb = FiveColorHeatMap(1 - diffs.Z[j,i]/curr_max)
                out.write('v %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f\n' % (X[i,j], Y[i,j], Z[j,i], rgb[0], rgb[1], rgb[2]))
            else:
                out.write('v %10.2f %10.2f %10.2f\n' % (X[i,j], Y[i,j], Z[j,i]))

    for i in range(ncols-1):
        for j in range(nrows-1):
            if abs(Z[j,i] - nodata) > epsilon and abs(Z[j,i+1] - nodata) > epsilon and abs(Z[j+1,i] - nodata) > epsilon:
                out.write('f %d %d %d\n' % (self.getId(i,j+1), self.getId(i+1,j), self.getId(i,j)))
            if abs(Z[j,i+1] - nodata) > epsilon and abs(Z[j+1,i+1] - nodata) > epsilon and abs(Z[j+1,i] - nodata) > epsilon:
                out.write('f %d %d %d\n' % (self.getId(i,j+1), self.getId(i+1,j+1), self.getId(i+1,j)))
```

Método que crea el OBJ coloreado por DTM de secuencias de parámetros.

```
def writeObjColorizeByDiff(self, diffs, f_out):
    out=open(f_out, 'w')
    ncols=self.ncols
    nrows=self.nrows
    x0=self.xcenter
    y0=self.ycenter
    d=self.cellsize
    nodata=self.nodata

    X=np.arange(ncols*nrows).reshape(ncols, nrows)*0
    Y=np.arange(ncols*nrows).reshape(ncols, nrows)*0
    Z=self.Z

    for i in range(ncols):
        for j in range(nrows):
            X[i,j]=x0+i*d
            Y[i,j]=y0+(nrows-j-1)*d

    max_z_distance = diffs.getMaxZ() - diffs.getMinZ()
    epsilon = 1e-3
    for i in range(ncols):
        for j in range(nrows):
            if diffs.Z[j,i] > epsilon:
                rgb = FiveColorHeatMap((diffs.getMaxZ() - diffs.Z[j,i])/max_z_distance)
                out.write('v %10.2f %10.2f %10.2f %10.2f %10.2f %10.2f\n' % (X[i,j], Y[i,j], Z[j,i], rgb[0], rgb[1], rgb[2]))
            else:
                out.write('v %10.2f %10.2f %10.2f\n' % (X[i,j], Y[i,j], Z[j,i]))

    for i in range(ncols-1):
        for j in range(nrows-1):
            if abs(Z[j,i] - nodata) > epsilon and abs(Z[j,i+1] - nodata) > epsilon and abs(Z[j+1,i] - nodata) > epsilon:
                out.write('f %d %d %d\n' % (self.getId(i,j+1), self.getId(i+1,j), self.getId(i,j)))
            if abs(Z[j,i+1] - nodata) > epsilon and abs(Z[j+1,i+1] - nodata) > epsilon and abs(Z[j+1,i] - nodata) > epsilon:
                out.write('f %d %d %d\n' % (self.getId(i,j+1), self.getId(i+1,j+1), self.getId(i+1,j)))
```

Método que crea el OBJ coloreado por DTM de diferencias de valor Z.