INSTITUTO TECNOLÓGICO DE MORELIA

Servicio social

Trabajo final

Alumno: García González Kelvin Arath

Número de control: C19120350

Carrera: Ingeniería Electrónica

Fecha de inicio: 15 de junio de 2022

Fecha de terminación: 15 de diciembre de 2022

Índice

Introducción

El estudio y análisis de la cinética de oxidación a alta temperatura es un tema de gran importancia en diversos campos, como la ingeniería, la química y la física. Una técnica comúnmente utilizada para realizar este tipo de estudios es la termografía infrarroja, que permite medir la temperatura de un objeto a distancia. En este contexto, el uso de machine learning puede ser muy útil para analizar los datos obtenidos por termografía infrarroja y extraer información valiosa sobre la cinética de oxidación. Por ejemplo, se pueden utilizar algoritmos de machine learning para identificar patrones en los datos que permitan predecir el comportamiento de la oxidación a alta temperatura en un sistema de calentamiento por efecto Joule monitoreado con termografía infrarroja. Además, el uso de machine learning también puede ayudar a optimizar el proceso de calentamiento para lograr una oxidación más eficiente y controlada. En resumen, el uso de machine learning en el estudio y análisis de la cinética de oxidación a alta temperatura puede aportar un gran valor en diversos aspectos del proceso, desde la predicción del comportamiento de la oxidación hasta la optimización del calentamiento.

El objetivo es a partir de una imagen termográfica obtener el fondo, la barra de metal y el oxido, todo esto con un programa en Matlab.

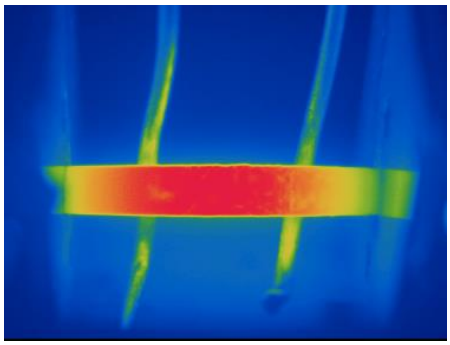

Figura 1 - Imagen original

Figura 2 - Barra de metal

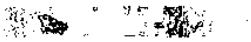

Figura 3 - Oxido

Desarrollo de actividades

Métodos de convolución

Se realizó una investigación sobre los métodos de convolución en matrices. La convolución es una técnica matemática que se utiliza en diversas áreas, como en el procesamiento de señales y en el aprendizaje automático.

En el contexto del aprendizaje automático, las convoluciones se utilizan en las redes neuronales convolucionales para identificar patrones en imágenes y videos. Estas redes utilizan filtros que se deslizan sobre la imagen en diferentes direcciones para extraer características relevantes de la misma.

Hay diferentes métodos para realizar la convolución en matrices, como la convolución directa, la fft y la convolución en tiempo discreto. Cada uno de estos métodos tiene sus propias ventajas y desventajas en términos de velocidad y precisión.

El método Sobel

Una de las técnicas que se pueden utilizar para obtener los bordes o líneas en imágenes es el método de Sobel. Este método se basa en el uso de operadores de convolución especiales que se aplican sobre la imagen para detectar cambios en la intensidad de los pixeles.

El operador de Sobel se compone de dos matrices, una para detectar cambios en la intensidad en la dirección horizontal y otra para detectar cambios en la dirección vertical. Estas matrices se aplican sobre la imagen mediante un proceso de convolución, y los valores resultantes se combinan para obtener una imagen en escala de grises que muestra los bordes o líneas presentes en la imagen original.

El método de Sobel es una técnica efectiva para detectar bordes, pero también tiene sus limitaciones. Por ejemplo, puede ser sensible a ruido y no es tan preciso como otros métodos en algunas situaciones. Sin embargo, sigue siendo una herramienta útil en el análisis de imágenes.

Figura 4 – Ejemplo del algoritmo sobel en Python

Limitaciones del método sobel

El método de Sobel tiene algunas limitaciones en cuanto a su precisión y sensibilidad a ruido. Algunas de estas limitaciones son las siguientes:

- Sensibilidad a ruido: El algoritmo de Sobel es sensible a ruido en la imagen original, lo que puede afectar negativamente la calidad del resultado obtenido. Esto se debe a que los operadores de convolución utilizados en el método pueden amplificar el ruido presente en la imagen, lo que dificulta la detección precisa de bordes.
- Precisión: El algoritmo de Sobel no es tan preciso como otros métodos en algunas situaciones. Por ejemplo, puede tener dificultades para detectar bordes en imágenes con baja resolución o en imágenes que presentan cambios bruscos en la intensidad de los pixeles.
- Tamaño del kernel: El tamaño del kernel utilizado en el operador de Sobel puede afectar la precisión y la calidad del resultado obtenido. Un kernel demasiado pequeño puede no ser capaz de detectar bordes finos, mientras que un kernel demasiado grande puede introducir bordes falsos en la imagen resultante.

En general, aunque el algoritmo de Sobel es una técnica efectiva para detectar bordes en imágenes, es importante tener en cuenta sus limitaciones para obtener resultados precisos y de alta calidad.

Implementación del algoritmo sobel en Matlab

Para el programa final se solicitó que tuviera una interfaz gráfica (GUI) y que pudiera seleccionar un directorio para obtener el fondo, la barra de metal, el óxido y finalmente una matriz de categorías que se explicará más adelante.

Sobel es un algoritmo de detección de bordes utilizado en el procesamiento de imágenes. El algoritmo de Sobel utiliza dos filtros diferentes para detectar los bordes horizontal y vertical en una imagen. Una vez que se han detectado los bordes, se combinan los resultados para obtener una imagen de bordes con una buena calidad y precisión.

En MATLAB, el algoritmo de Sobel se puede aplicar a una imagen utilizando la función edge. Esta función toma como argumentos la imagen y el tipo de detección de bordes que se desea utilizar (en este caso, "sobel").

Matrices en Matlab

Las matrices son un tipo de dato fundamental en MATLAB. Una matriz es un arreglo de números que se dispone en filas y columnas, y se utiliza en MATLAB para representar y manipular datos numéricos. Para trabajar con matrices en MATLAB, primero es necesario conocer algunas de sus funciones básicas. Por ejemplo, la función **zeros** se utiliza para crear una matriz llena de ceros, mientras que la función **ones** se utiliza para crear una matriz llena de unos. Para crear una matriz de un tamaño específico y con valores específicos, se puede utilizar la función **matrix**, que toma como argumentos una lista de números que se deben colocar en la matriz.

Procesamiento de imágenes con Matlab

El procesamiento de imágenes con matrices en MATLAB se basa en el hecho de que una imagen digital se puede representar como una matriz de números. Cada elemento de la matriz representa un pixel de la imagen, y su valor indica el color y el nivel de brillo de ese pixel.

Para trabajar con imágenes en MATLAB, primero es necesario cargar la imagen en el programa. Esto se puede hacer utilizando la función **imread**, que toma como argumento la ruta del archivo de imagen y devuelve una matriz que representa la imagen.

Una vez que se ha cargado la imagen en una matriz, se pueden realizar diversas operaciones para procesarla. Por ejemplo, se pueden aplicar filtros para mejorar el contraste o eliminar el ruido, se pueden aplicar transformaciones geométricas para cambiar la perspectiva de la imagen, o se pueden extraer características para su posterior análisis.

Guide (Graphical User Interface Development Environment)

Es una herramienta en MATLAB que permite crear interfaces gráficas de usuario (GUI) de manera sencilla y rápida. La GUI es una forma de presentar información y recibir entrada del usuario a través de elementos gráficos como botones, cajas de texto, menús, etc.

Para utilizar Guide en un proyecto de Matlab, basta con escribir **guide** en la línea de comandos de Matlab, como se muestra en la siguiente imagen:

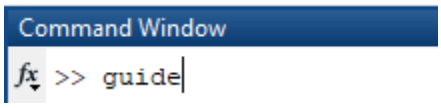

Figura 5 – Guide

Una vez que se presiona la tecla enter, se arroja la siguiente venta, en donde es posible elegir dentro de un tipo de proyectos de interfaz gráfica en Matlab, en donde existen algunos ya diseñados, pero es conveniente elegir el archivo en blanco y comenzar a trabajar desde cero, la pestaña que se muestra es la siguiente:

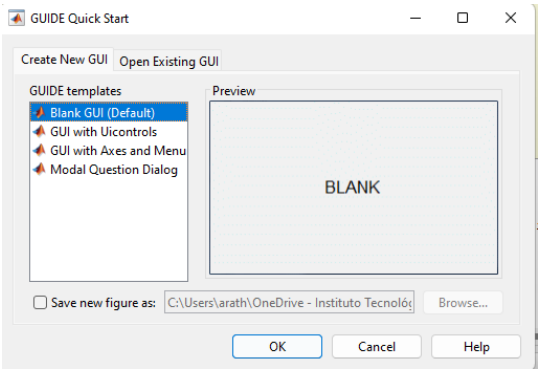

Figura 6 – Guide proyectos

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
path = get(handles.edit1,'String');
cd(path);
%read files .tiff in folder
files = dir('*.tiff');


% loop through each file in the folder
for o = 1:numel(files)
    Image{o} = imread(files(o).name);
    ImGray{o} = rgb2gray(Image{o});
    ImSobel = edge(ImGray{o},'Sobel');
    [m, n, r2] = size(Image{o});
    xInitial = [0,0];, x2Initial = 0;,yInitial = 0;,y2Initial = 0;
    xFinal = 0;, yFinal = 0;
    folder_name = files(o).name(1:end-5);
    mkdir(folder_name)
    categories = [m n];
    topLeft = [m n]; %initial point
    lowRight = [0 0]; % final point
    mask1 = [m n];
    mask2 = [m n];
    mask3 = [m n];
```

Figura 7 – Lecturas de ruta y código en botón

Una vez que se creo una interfaz gráfica es posible elegir entre cajas de texto, botones y otras vistas disponibles, pero para el proyecto se utilizaron dos botones y una caja de texto. Las líneas de código anterior son las encargadas de primeramente, cambiar al **path** que se ha elegido como ruta, y files = dir(‘*.tiff); lee cada archivo con extensión .tiff en la ruta seleccionada, posteriormente, mediante un ciclo, se recorre cada archivo con extensión .tiff en la carpeta, y comienza el procesamiento de imágenes.

```
%% Code to detect edges in aeach image
          for i = 1: m % rows = number of rows in ImSobel image
            for j = 1: n % cols = number of columns in ImSobel image
                if ( ImSobel(i,j) == 1 && ImSobel(i,j+2)== 1 && ImSobel(i,j+4)== 1 && ImSobel(i,j+6)== 1 && ImSobel(i,j+10) == 1)
                    if ( j < topLeft(1,2) ) % x resolution
                      yInitial = i; %i axe y
                      xInitial = j; %j axe x
                      topLeft(1,1) = yInitial;
                      topLeft(1,2) = xInitial;
                    end
                    if (j > lowRight(1,2) && i + 5 > topLeft(1,1) && i - 5 > topLeft(1,1))
                         yFinal = i;
                         xFinal = j;
                         lowRight(1,1) = yFinal;
                         lowRight(1,2) = xFinal;
                    end
                    if( i == lowRight(1,1) && j > lowRight(1,2))
                         xFinal = j;
                         lowRight(1,2) = xFinal;
                    end
                end
            end
          end
```

Figura 8 – Detección de bordes

El código que se muestra en la figura 8, es el encargado de hacer la detección de los bordes, ya que al saber que se trata de una figura rectangular, se aplico cierta lógica que permitiera detectar líneas rectas en una imagen, y con esto determinar los puntos iniciales y finales de los bordes.

Hasta este momento el código realiza un proceso que permite detectar líneas rectas de esta imagen (Figura 9), y conocer los valores de las esquinas del rectángulo, para de esta forma hacer un análisis posteriormente únicamente a la zona de interés que es la parte rectangular correspondiente a la probeta.

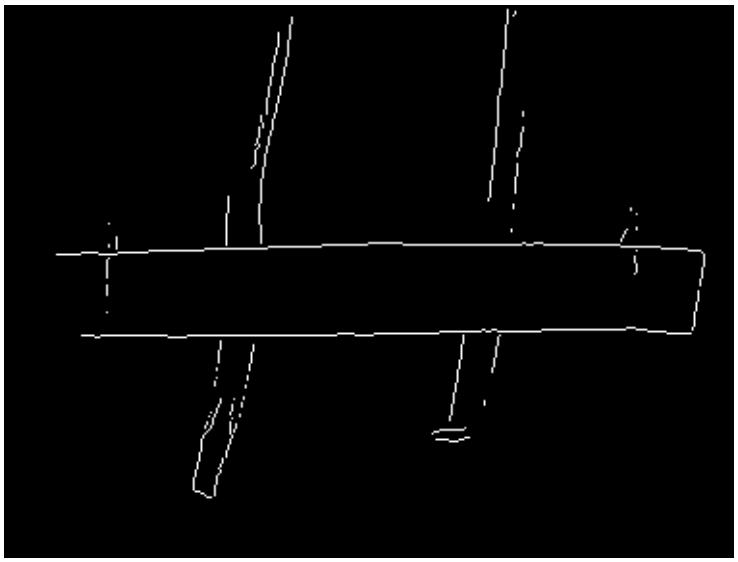

Figura 9 – Procesamiento de imagen con función Sobel

Ahora que ya se tiene la zona de interés identificada, se creó una matriz llamada categoría, esta matriz guarda un 1 en las zonas de la imagen correspondiente al fondo y dado que ya conocemos la zona de interés, todo lo que no esté dentro de esa zona de interés se considerará como fondo. De otro modo, se guarda un 2 en las posiciones de la imagen que contienen la zona de interés, y dado que ya se conocen los puntos de inicio y fin de la zona de interés, basta con rellenar con 2 la matriz todos los puntos que están dentro de esta zona. Y por último se guarda un 3 en las zonas que se detecta oxidación, esta zona se identifica, ya que existen cambios de color en la imagen original, en donde se aprecia un color que contrasta respecto a la otra zona, por lo que se hizo uso de un valor de umbral, el cual considera la zona de interés, y todos aquellos puntos en que la zona de interés tenga cambios de color considerables, los tomará como zona de óxido, y por lo tanto rellenará estos puntos con un valor 3. Además, en esta sección se crean 3 matrices que corresponden a las 3 diferentes máscaras (fondo, probeta y oxidación), y para estas matrices se sigue la misma lógica anterior, únicamente que, para estas máscaras, se rellena con 0 todo lo que no sea fondo, probeta y oxidación respectivamente.

La implementación de este código se puede observar en la figura 10.

```
%% categories, categories is a matrix that containe three numbers, 1 -> background, 2 -> metal, 3 -> oxide zone
%mask1 -> background, mask2 -> metal
            for i = 1: m
                  for j = 1: n
                    if( j < xFinal && i < yFinal && j > xInitial && i > yInitial )
                        categories(i,j) = 2;
                        mask1(i,j) = 0;
                        mask2(i,j) = 1;
                    else
                        categories(i,j) = 1;
                        mask1(i,j) = 1;
                        mask2(i,j) = 0;
                    end
                  end
            end
%% code to detect oxide zone
            oxideImage = (Image{o}(:,:,1) > 231) & (Image{o}(:,:,2) > 60 & Image{o}(:,:,2) < 90) & (Image{o}(:,:,3) < 45);
            for i = 1: m
                for j = 1: n
                    if(oxideImage(i,j) == 1)
                        categories(i,j) = 3;
                        mask3(i,j) = 1;
                    else mask3(i,j) = 0;
                    end
                end
            end
```

Figura 10 – Matriz de categorías y creación de máscaras

Hasta este momento ya tenemos las matrices de categorías, mask1, mask2 y mask3, ahora solo se necesita guardar estos archivos dentro de cada carpeta, en donde la matriz categorías se guardará como un archivo .xlsx, y mask1, mask2 y mask3 como archivos .png. Este proceso se muestra en la figura 11.

```
%% Create a new file .xlsx in folder
    file_name = fullfile(folder_name, strcat('categories',num2str(o),'.xlsx'));
    datos = categories;
    xlswrite(file_name, datos)

%% Create a .png files in folder for matrix mask1,mask2,mask3

    imwrite(mask1, strcat(file_name(1:end-16),'mask1','.png'));
    imwrite(mask2, strcat(file_name(1:end-16),'mask2','.png'));
    imwrite(mask3, strcat(file_name(1:end-16),'mask3','.png'));
```

Figura 11 – Guardado de archivos con extensión .xlsx y .png

El código que se muestra en la figura 11, almacena los 4 archivos generados anteriormente, dentro de una carpeta que se crea con la función mkdir, en donde esta carpeta tiene el mismo nombre de cada imagen dentro de la ruta seleccionada por medio de la interfaz gráfica, por lo tanto, al estar dentro de un ciclo for, este código generará la carpeta, y los 4 archivos dentro de cada carpeta para todas las imágenes con la extensión .tiff en la carpeta.

Como se mencionó anteriormente, la interfaz gráfica de usuario tiene un diseño sencillo, pero que permite que el código pueda procesar todas las imágenes contenidas dentro de una carpeta, accediendo a ella desde una manera sencilla. La interfaz cuenta con un botón (Browse) el cual permite desplegar la pestaña de file explorer, y al seleccionar cualquier archivo y dar clic en aceptar, la ruta se copia en la caja de texto (file), con esto, tenemos ahora el path con el que queremos trabajar, y al presionar el botón OK, este copia la ruta que está contenida dentro de la caja de texto, y con la función cd(), se cambia a este directorio, y el código comienza a trabajar directamente en esta ruta.

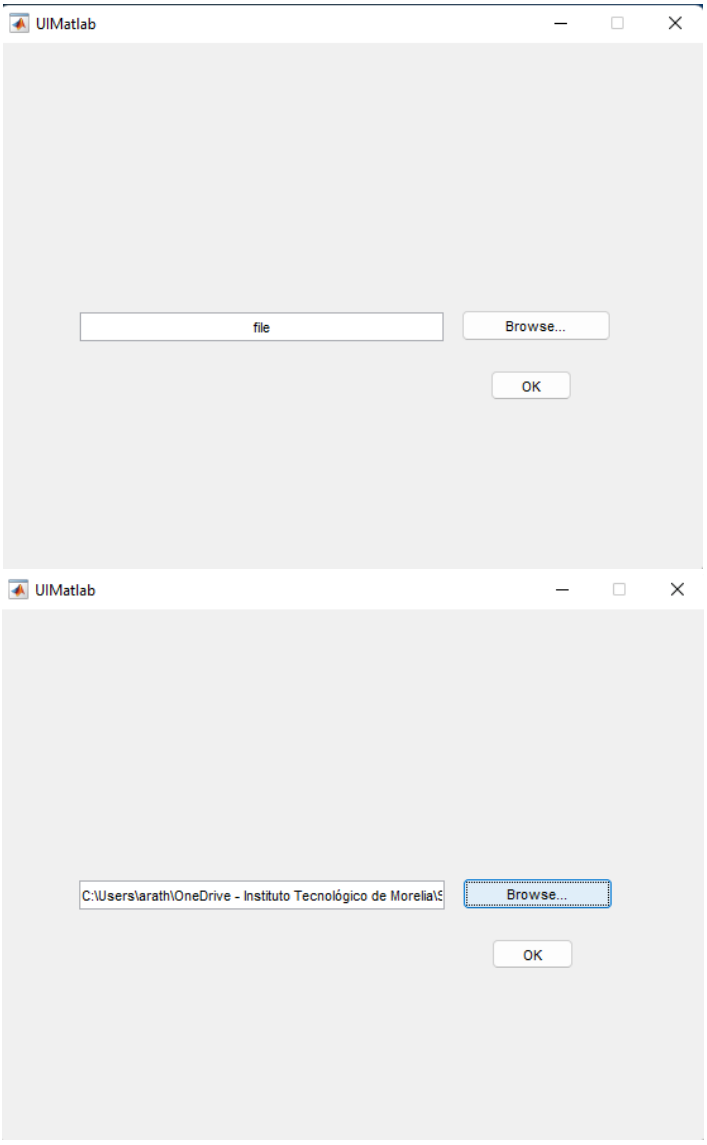

Figura 12 – Interfaz gráfica de usuario

Resultados

El programa funcionó satisfactoriamente, el usuario puede utilizar el programa para seleccionar una carpeta que contenga imágenes con extensión tiff, la cuál es la extensión utilizada por la cámara termográfica con la que se estuvo trabajando. Seguido a esto cuando el usuario presiona el botón generar el programa escanea el directorio en busca de imágenes con la extensión ya mencionada para crear una carpeta con su mismo nombre y dentro de ella generar las imágenes con el fondo, la barra de metal y el óxido, además de una matriz de categorías.

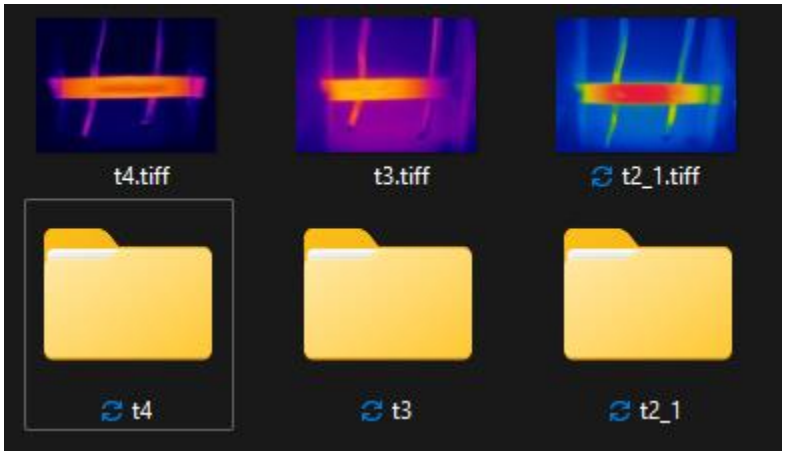

Figura 13 – Creación de carpeta por cada imagen

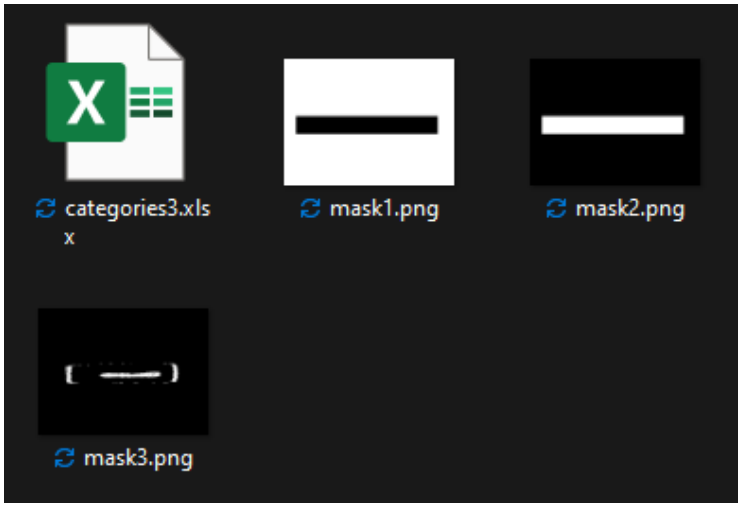

Figura 14 – Creación de las imágenes y matriz de categorías

Recomendaciones y anexos

Una vez realizado el trabajo sobre el uso de machine learning en el estudio y análisis de la cinética de oxidación a alta temperatura, se pueden hacer las siguientes recomendaciones:

1. Continuar explorando y profundizando en el uso de machine learning en el estudio de la oxidación a alta temperatura. Esto incluye la investigación de diferentes algoritmos de machine learning y su aplicación a diferentes situaciones y escenarios.
2. Mejorar y optimizar el programa desarrollado en Matlab, hacer uso de funciones que permitan llevar un mejor orden y estructura de código, así como la mejora en cuanto a la detección de los bordes que se generan a través del código.
3. Realizar pruebas y evaluaciones del programa desarrollado en diferentes conjuntos de datos y en diferentes condiciones para validar su eficacia y precisión.
4. Compartir los resultados y las conclusiones del trabajo con la comunidad científica y técnica a través de publicaciones en revistas especializadas y en eventos académicos.
5. Colaborar con otros investigadores y expertos en el tema para desarrollar aplicaciones prácticas del uso de machine learning en el estudio de la oxidación a alta temperatura y promover su uso en la industria.

Conclusiones

El desarrollo de un algoritmo, que se aplica a un caso real, y con objetivos específicos bien definidos, es lo más próximo a un trabajo real que se ha desarrollado a lo largo de la carrera, lo que ha permitido desarrollar habilidades de análisis y experimentación a lo largo del proyecto, en donde además del desarrollo del algoritmo, se ha trabajado directamente con las herramientas para el análisis termográfico, haciendo uso de software de carácter ingenieril, una planta de alta temperatura, uso de una cámara termográfica, entre otras actividades. Lo que ha permitido desarrollar un sentido de profesionalidad y responsabilidad con el uso del equipo, y enfrentar desafíos, como el cumplimiento en tiempo y forma de los avances solicitados por el profesor a cargo, siendo esta una de las mayores utilidades y beneficios que se han obtenido en el período del servicio social.

El algoritmo desarrollado contempla todos los requisitos fundamentales para el procesamiento de una probeta calentada a alta temperatura, en donde los requisitos se cumplen, sin embargo, es importante mencionar que el código se vería beneficiado haciendo una optimización del mismo, haciendo uso de funciones que permitan hacerlo más funcional y estructurado.

Cuadro de firmas

NOMBRE DEL RESPONSABLE DEL PROGRAMA		FIRMA DEL ALUMNO
FIRMA DEL RESPONSABLE		Vo.BO. OFICINA DEL SERVICIO SOCIAL ITM