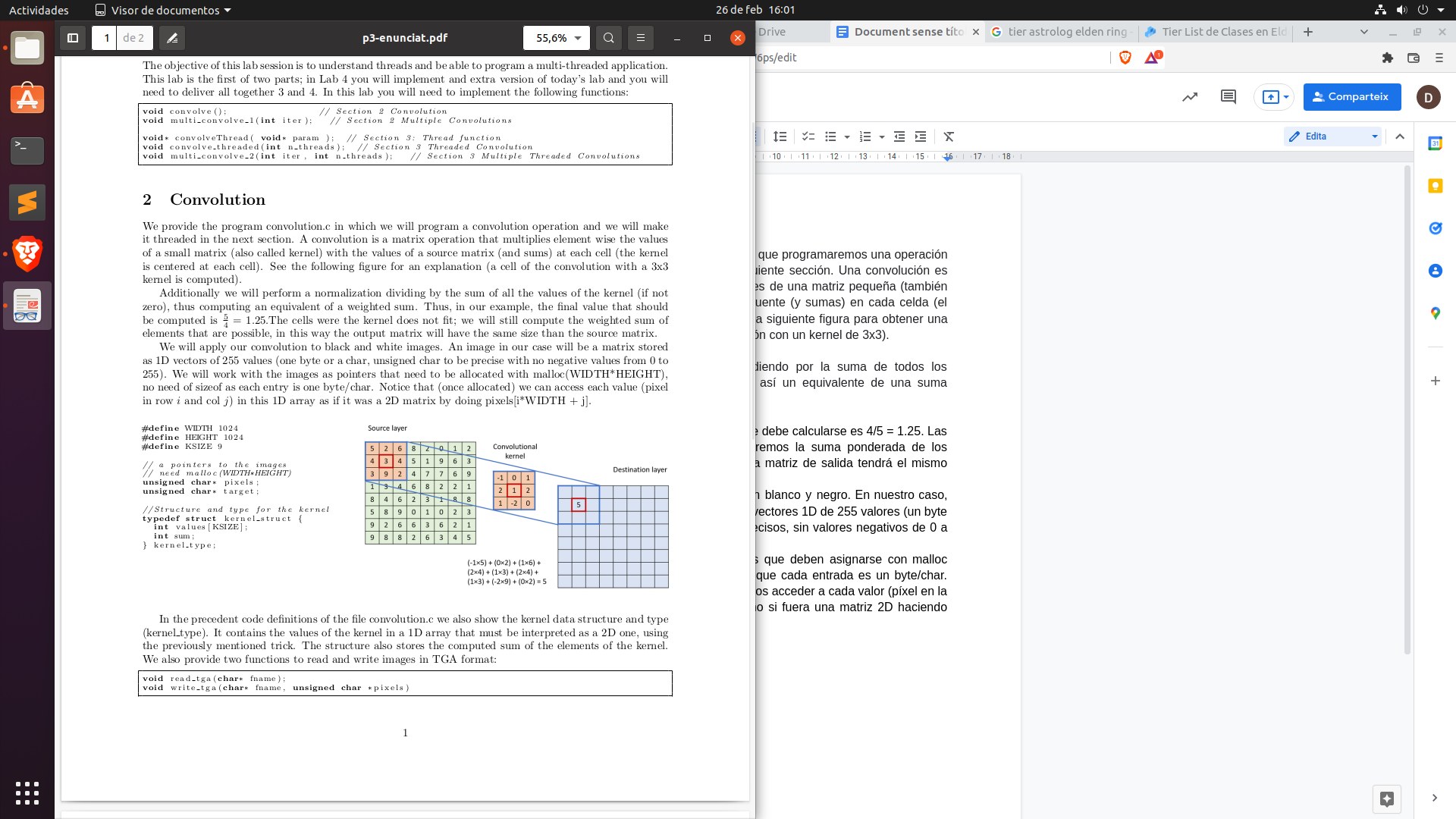
Proporcionamos el programa convolution.c en el que programaremos una operación de convolución y la haremos threads en la siguiente sección. Una convolución es una operación matricial que multiplica los valores de una matriz pequeña (también llamada kernel) con los valores de una matriz fuente (y sumas) en cada celda (el kernel está centrado en cada celda). Consulte la siguiente figura para obtener una explicación (se calcula una celda de la convolución con un kernel de 3x3).

Además, realizaremos una normalización dividiendo por la suma de todos los valores del kernel (si no es cero), calculando así un equivalente de una suma ponderada.

Por lo tanto, en nuestro ejemplo, el valor final que debe calcularse es 4/5 = 1.25. Las celdas donde el núcleo no cabe; aún calcularemos la suma ponderada de los elementos que son posibles, de esta manera la matriz de salida tendrá el mismo tamaño que la matriz de origen.

Aplicaremos nuestra convolución a imágenes en blanco y negro. En nuestro caso, una imagen será una matriz almacenada como vectores 1D de 255 valores (un byte o un carácter, un carácter sin signo para ser precisos, sin valores negativos de 0 a 255).

Trabajaremos con las imágenes como punteros que deben asignarse con malloc (ANCHO \* ALTO), sin necesidad de sizeof ya que cada entrada es un byte/char. Tenga en cuenta que (una vez asignado) podemos acceder a cada valor (píxel en la fila i y en la columna j) en esta matriz 1D como si fuera una matriz 2D haciendo píxeles [i\*WIDTH + j].



En las definiciones de código precedentes del archivo convolution.c también mostramos la estructura y el tipo de datos del núcleo (tipo de núcleo). Contiene los valores del kernel en un arreglo 1D que debe interpretarse como uno 2D, usando el truco mencionado anteriormente. La estructura también almacena la suma calculada de los elementos del kernel. También proporcionamos dos funciones para leer y escribir imágenes en formato TGA:

Ahora estamos listos para comenzar a implementar una convolución simple haciendo un bucle doble en el que calculamos cada píxel de la imagen de destino, haciendo la suma multiplicativa explicada (normalizada al final) del kernel y la submatriz relevante de la imagen de origen ( píxeles). Primero compile y ejecute el programa provisto que necesita completar, luego siga los pasos provistos.

g c c c o n v o l u t i o n . c m y u t i l s . c = p t h r e a d = o conv

1. Mire dónde asignamos la memoria para las imágenes: píxeles y destino.

2. Vamos a utilizar un Kernel gaussiano. Rellene el núcleo utilizando el filtro gaussiano().

3. Mire dónde leímos el archivo de imagen tga proporcionado. Probamos que funciona copiando la memoria de "píxeles" a "objetivo" llamando: memcpy (destino, fuente, tamaño) y luego escribimos la imagen y verificamos que se escriben las mismas imágenes.

4. La función convolve() ya está implementada para usted. Recorre todos los píxeles de la imagen aplicando el píxel objetivo de cálculo en cada píxel.

5. Prueba que todo funciona correctamente escribiendo la imagen resultante. 6. Implemente ahora la función multi convolve 1(int iter) que realiza un número de convoluciones especificadas seguidas. Asegúrese de que la imagen de destino esté llena, luego intercambie punteros y llame a convolve en un bucle.

7. Pruebe varias iteraciones (10, 20, 30) de la imagen resultante; ¿lo que sucede?

3. Convolución roscada

Ahora vamos a hacer una versión "\_threaded" del programa anterior.

1. Ahora implemente la función convolucionar subprocesos creando un solo subproceso. Crea un hilo y espéralo. Escriba la función de hilo convolucionarHilo. Puede recibir como parámetros el id del hilo y cuantos hilos hay (en el caso actual 0 y 1 respectivamente). Puede usar una matriz para eso (recuerde que cada subproceso necesitará su espacio único reservado para parámetros, así que use malloc y free). La función calcular el píxel objetivo puede permanecer sin cambios. Pruebe que la convolución funciona correctamente y que genera la misma imagen un poco más borrosa que la original (como en la sección anterior).