{

"nbformat": 4,

"nbformat\_minor": 0,

"metadata": {

"colab": {

"provenance": []

},

"kernelspec": {

"name": "python3",

"display\_name": "Python 3"

},

"language\_info": {

"name": "python"

}

},

"cells": [

{

"cell\_type": "markdown",

"source": [

"# Task 1"

],

"metadata": {

"id": "uNbk-BdfJfi8"

}

},

{

"cell\_type": "code",

"execution\_count": 19,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 529

},

"id": "n9IEoW9kD87t",

"outputId": "093c9c18-9593-43c3-ca60-a052a18da0c2"

},

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<PIL.Image.Image image mode=RGB size=512x512>"

],

"image/png": "\n",

"image/jpeg": "\n"

},

"metadata": {}

}

],

"source": [

"import cv2\n",

"import numpy as np\n",

"from google.colab.patches import cv2\_imshow\n",

"\n",

"# 1. Read the image\n",

"image = cv2.imread('/content/Image.png') # Replace 'image.jpg' with your image file\n",

"\n",

"# 2. Display the image\n",

"cv2\_imshow(image)\n"

]

},

{

"cell\_type": "code",

"source": [

"# 3. Extraction of Image size\n",

"height, width, channels = image.shape\n",

"print(f\"Image Size: Width = {width}, Height = {height}, Channels = {channels}\")"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "XrDIicsRIM4U",

"outputId": "968fc029-9009-42f1-8c24-baa535383502"

},

"execution\_count": 20,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Image Size: Width = 512, Height = 512, Channels = 3\n"

]

}

]

},

{

"cell\_type": "code",

"source": [

"# 4. Calculation of image pixels\n",

"total\_pixels = height \* width\n",

"print(f\"Total Pixels: {total\_pixels}\")"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "BWph56VaIOdl",

"outputId": "d562fd73-8c29-45e5-c7a9-a4cbfaf18c3a"

},

"execution\_count": 21,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Total Pixels: 262144\n"

]

}

]

},

{

"cell\_type": "code",

"source": [

"# 5. Convert RGB to Grayscale image\n",

"gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)\n",

"cv2\_imshow(gray\_image)\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 529

},

"id": "p\_t4QxdJISzV",

"outputId": "2ab29678-6f4c-454e-84db-ecd26aa58f9e"

},

"execution\_count": 22,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<PIL.Image.Image image mode=L size=512x512>"

],

"image/png": "\n",

"image/jpeg": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# 6. Convert RGB to Grayscale and then to Binary using a threshold\n",

"threshold = 128 # You can adjust this threshold value\n",

"\_, binary\_image = cv2.threshold(gray\_image, threshold, 255, cv2.THRESH\_BINARY)\n",

"cv2\_imshow(binary\_image)\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 529

},

"id": "b2tl112GIZPW",

"outputId": "98702886-2726-491c-f08d-cdd98256cf9a"

},

"execution\_count": 23,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<PIL.Image.Image image mode=L size=512x512>"

],

"image/png": "\n",

"image/jpeg": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Count the area of black pixels\n",

"black\_pixel\_count = np.sum(binary\_image == 0)\n",

"print(f\"Black Pixel Count: {black\_pixel\_count}\")\n",

"\n",

"# Display the image size again\n",

"print(f\"Binary Image Size: Width = {width}, Height = {height}\")\n",

"\n",

"\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "\_J6GeegVIhvT",

"outputId": "716a6cf1-12a1-4085-fc2a-70692c594c5c"

},

"execution\_count": 24,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Black Pixel Count: 130547\n",

"Binary Image Size: Width = 512, Height = 512\n"

]

}

]

},

{

"cell\_type": "markdown",

"source": [

"# Task 2"

],

"metadata": {

"id": "OXxVe2grJWwP"

}

},

{

"cell\_type": "code",

"source": [

"from skimage import filters\n",

"from skimage import feature\n",

"# Apply Sobel Operator\n",

"sobelx = cv2.Sobel(gray\_image, cv2.CV\_64F, 1, 0, ksize=3) # Horizontal edges\n",

"sobely = cv2.Sobel(gray\_image, cv2.CV\_64F, 0, 1, ksize=3) # Vertical edges\n",

"sobel = cv2.magnitude(sobelx, sobely) # Combine horizontal and vertical edges\n",

"\n",

"# Convert to uint8\n",

"sobel = np.uint8(np.absolute(sobel))\n",

"\n",

"# Display the Sobel Edge Detection Result\n",

"plt.figure(figsize=(8, 6))\n",

"plt.imshow(sobel, cmap='gray')\n",

"plt.title('Sobel Edge Detection')\n",

"plt.axis('off')\n",

"plt.show()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 521

},

"id": "EeNQCn6LNM1s",

"outputId": "bcb8782c-125b-4ad0-e660-075d32be5029"

},

"execution\_count": 40,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 800x600 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Apply Prewitt Operator using scikit-image\n",

"prewitt\_edges = filters.prewitt(gray\_image)\n",

"\n",

"# Display the Prewitt Edge Detection Result\n",

"plt.figure(figsize=(8, 6))\n",

"plt.imshow(prewitt\_edges, cmap='gray')\n",

"plt.title('Prewitt Edge Detection')\n",

"plt.axis('off')\n",

"plt.show()"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 521

},

"id": "EuaJs7jvNag9",

"outputId": "4c660df9-4de7-42d6-e015-49f0be6ac798"

},

"execution\_count": 41,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 800x600 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Apply Roberts Cross Operator using scikit-image\n",

"roberts\_edges = filters.roberts(gray\_image)\n",

"\n",

"# Display the Roberts Cross Edge Detection Result\n",

"plt.figure(figsize=(8, 6))\n",

"plt.imshow(roberts\_edges, cmap='gray')\n",

"plt.title('Roberts Cross Edge Detection')\n",

"plt.axis('off')\n",

"plt.show()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 521

},

"id": "TFcTQGL1NdFN",

"outputId": "5cfcbed7-4756-4308-a196-c90deaf27eae"

},

"execution\_count": 42,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 800x600 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Apply Canny Edge Detector\n",

"# Parameters: (image, threshold1, threshold2)\n",

"# You may need to adjust the thresholds based on your image\n",

"canny\_edges = cv2.Canny(gray\_image, 100, 200)\n",

"\n",

"# Display the Canny Edge Detection Result\n",

"plt.figure(figsize=(8, 6))\n",

"plt.imshow(canny\_edges, cmap='gray')\n",

"plt.title('Canny Edge Detection')\n",

"plt.axis('off')\n",

"plt.show()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 521

},

"id": "OlCC7IqPNhbn",

"outputId": "07d0690c-3347-4892-c53e-908692eb2b03"

},

"execution\_count": 43,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 800x600 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"from skimage import filters, segmentation, color, morphology, feature\n",

"from skimage.util import img\_as\_ubyte\n",

"# Global Thresholding\n",

"# Define a threshold value\n",

"global\_thresh = 127 # You can adjust this value\n",

"\n",

"# Apply global thresholding\n",

"\_, global\_thresh\_image = cv2.threshold(gray\_image, global\_thresh, 255, cv2.THRESH\_BINARY)\n",

"\n",

"# Display the Result\n",

"plt.figure(figsize=(6, 6))\n",

"plt.imshow(global\_thresh\_image, cmap='gray')\n",

"plt.title(f'Global Thresholding (Threshold = {global\_thresh})')\n",

"plt.axis('off')\n",

"plt.show()"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 521

},

"id": "\_msiku2PNkHD",

"outputId": "309374ab-fb68-45a1-8ff0-811347ff663c"

},

"execution\_count": 44,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 600x600 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Adaptive Thresholding\n",

"# Parameters for adaptiveThreshold:\n",

"# - src: Source image (should be grayscale)\n",

"# - maxValue: Non-zero value assigned to pixels exceeding the threshold\n",

"# - adaptiveMethod: Adaptive thresholding algorithm to use\n",

"# - thresholdType: Type of thresholding\n",

"# - blockSize: Size of the pixel neighborhood used to calculate the threshold\n",

"# - C: Constant subtracted from the mean or weighted mean\n",

"\n",

"# Apply Adaptive Mean Thresholding\n",

"adaptive\_mean = cv2.adaptiveThreshold(\n",

" gray\_image,\n",

" 255,\n",

" cv2.ADAPTIVE\_THRESH\_MEAN\_C,\n",

" cv2.THRESH\_BINARY,\n",

" blockSize=11,\n",

" C=2\n",

")\n",

"\n",

"# Apply Adaptive Gaussian Thresholding\n",

"adaptive\_gaussian = cv2.adaptiveThreshold(\n",

" gray\_image,\n",

" 255,\n",

" cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\n",

" cv2.THRESH\_BINARY,\n",

" blockSize=11,\n",

" C=2\n",

")\n",

"\n",

"# Display the Results\n",

"plt.figure(figsize=(12, 6))\n",

"\n",

"plt.subplot(1, 2, 1)\n",

"plt.imshow(adaptive\_mean, cmap='gray')\n",

"plt.title('Adaptive Mean Thresholding')\n",

"plt.axis('off')\n",

"\n",

"plt.subplot(1, 2, 2)\n",

"plt.imshow(adaptive\_gaussian, cmap='gray')\n",

"plt.title('Adaptive Gaussian Thresholding')\n",

"plt.axis('off')\n",

"\n",

"plt.show()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 482

},

"id": "p\_-0WDYZNnp\_",

"outputId": "097b80e8-2fc8-4204-e5ad-9a53c9b225ca"

},

"execution\_count": 45,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 1200x600 with 2 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Canny Edge Detection\n",

"# Parameters: (image, threshold1, threshold2, apertureSize, L2gradient)\n",

"# threshold1 and threshold2 are the lower and upper thresholds for the hysteresis procedure\n",

"\n",

"# Apply Canny Edge Detector\n",

"canny\_edges = cv2.Canny(gray\_image, threshold1=100, threshold2=200)\n",

"\n",

"# Display the Result\n",

"plt.figure(figsize=(6, 6))\n",

"plt.imshow(canny\_edges, cmap='gray')\n",

"plt.title('Canny Edge Detection')\n",

"plt.axis('off')\n",

"plt.show()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 521

},

"id": "KyqT1\_LYNpgc",

"outputId": "9df49405-c958-4181-d68c-cf8f3c104485"

},

"execution\_count": 46,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 600x600 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

},

{

"cell\_type": "code",

"source": [

"# Watershed Algorithm for Image Segmentation\n",

"\n",

"# Step 1: Noise Reduction\n",

"# Apply a threshold to get the foreground\n",

"ret, thresh = cv2.threshold(gray\_image, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)\n",

"\n",

"# Step 2: Morphological Operations\n",

"# Remove noise and separate touching objects\n",

"# Morphological opening (remove small objects from the foreground)\n",

"kernel = np.ones((3,3), np.uint8)\n",

"opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)\n",

"\n",

"# Step 3: Sure Background Area\n",

"sure\_bg = cv2.dilate(opening, kernel, iterations=3)\n",

"\n",

"# Step 4: Sure Foreground Area\n",

"# Use distance transform and threshold\n",

"dist\_transform = cv2.distanceTransform(opening, cv2.DIST\_L2, 5)\n",

"ret, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)\n",

"\n",

"# Step 5: Unknown Region\n",

"sure\_fg = np.uint8(sure\_fg)\n",

"unknown = cv2.subtract(sure\_bg, sure\_fg)\n",

"\n",

"# Step 6: Marker Labelling\n",

"ret, markers = cv2.connectedComponents(sure\_fg)\n",

"\n",

"# Add one to all labels so that sure background is not 0, but 1\n",

"markers = markers + 1\n",

"\n",

"# Mark the unknown region with zero\n",

"markers[unknown == 255] = 0\n",

"\n",

"# Step 7: Apply Watershed\n",

"rgb\_image = cv2.cvtColor(gray\_image, cv2.COLOR\_GRAY2BGR)\n",

"markers = cv2.watershed(rgb\_image, markers)\n",

"\n",

"# Step 8: Mark Boundaries in Red\n",

"segmented\_image = rgb\_image.copy()\n",

"segmented\_image[markers == -1] = [255, 0, 0] # Boundaries marked in red\n",

"\n",

"# Display the Results\n",

"plt.figure(figsize=(12, 6))\n",

"\n",

"plt.subplot(1, 2, 1)\n",

"plt.imshow(rgb\_image)\n",

"plt.title('Original RGB Image')\n",

"plt.axis('off')\n",

"\n",

"plt.subplot(1, 2, 2)\n",

"plt.imshow(segmented\_image)\n",

"plt.title('Watershed Segmentation')\n",

"plt.axis('off')\n",

"\n",

"plt.show()\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 482

},

"id": "fNjQY6xeNzgW",

"outputId": "fc2761cd-a646-4c37-f89d-732a75fe97ae"

},

"execution\_count": 50,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 1200x600 with 2 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

}

]

}