

Dokumentacja promptu z chatu GPT:

napisz kod w języku python który umożliwi mi wgranie do niego zdjęcia a następnie kod ten rozpozna najjaśniejszy obszar na tym obrazie

Aby rozpoznać najjaśniejszy obszar na obrazie, możemy użyć biblioteki OpenCV do przetwarzania obrazów. OpenCV jest potężnym narzędziem do analizy obrazów i posiada funkcje, które pozwolą nam na łatwe wykonanie tego zadania. Oto przykład kodu w Pythonie, który umożliwia wgranie obrazu, a następnie rozpoznanie najjaśniejszego obszaru na tym obrazie:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    # Wczytaj obraz w odcieniach szarości
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    return image

def find_brightest_region(image, region_size=50):
    # Znajdź współrzędne najjaśniejszego punktu
    (minVal, maxVal, minLoc, maxLoc) = cv2.minMaxLoc(image)

    # Określ współrzędne obszaru wokół najjaśniejszego punktu
    x, y = maxLoc
    top_left = (max(0, x - region_size // 2), max(0, y - region_size // 2))
    bottom_right = (min(image.shape[1], x + region_size // 2),
min(image.shape[0], y + region_size // 2))

    return top_left, bottom_right

def draw_rectangle(image, top_left, bottom_right):
    # Skopiuj obraz do rysowania
    image_with_rect = image.copy()

    # Narysuj prostokąt wokół najjaśniejszego obszaru
    cv2.rectangle(image_with_rect, top_left, bottom_right, (255, 0, 0), 2)

    return image_with_rect

def display_images(original, processed):
    # Wyświetl oryginalny obraz i obraz z zaznaczonym najjaśniejszym obszarem
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.title('Oryginalny obraz')
    plt.imshow(original, cmap='gray')

    plt.subplot(1, 2, 2)
    plt.title('Najjaśniejszy obszar zaznaczony')
    plt.imshow(processed, cmap='gray')

    plt.show()

if __name__ == "__main__":
    # Ścieżka do pliku obrazu
    image_path = 'path_to_your_image.jpg'
```

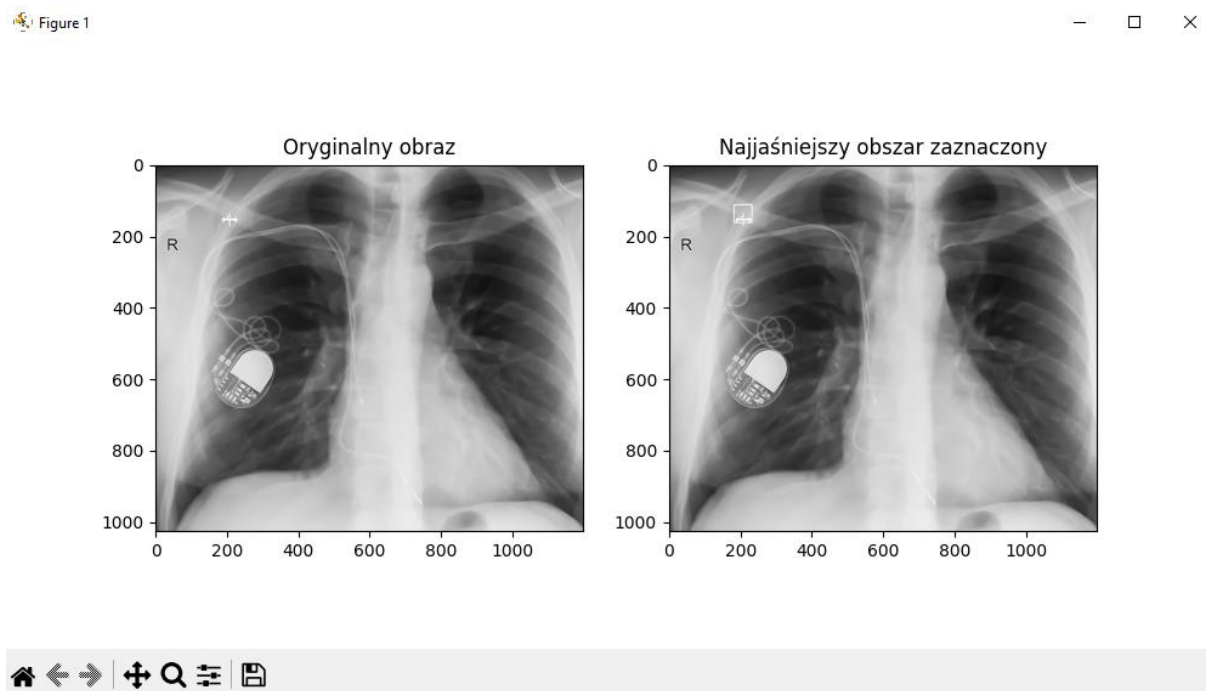
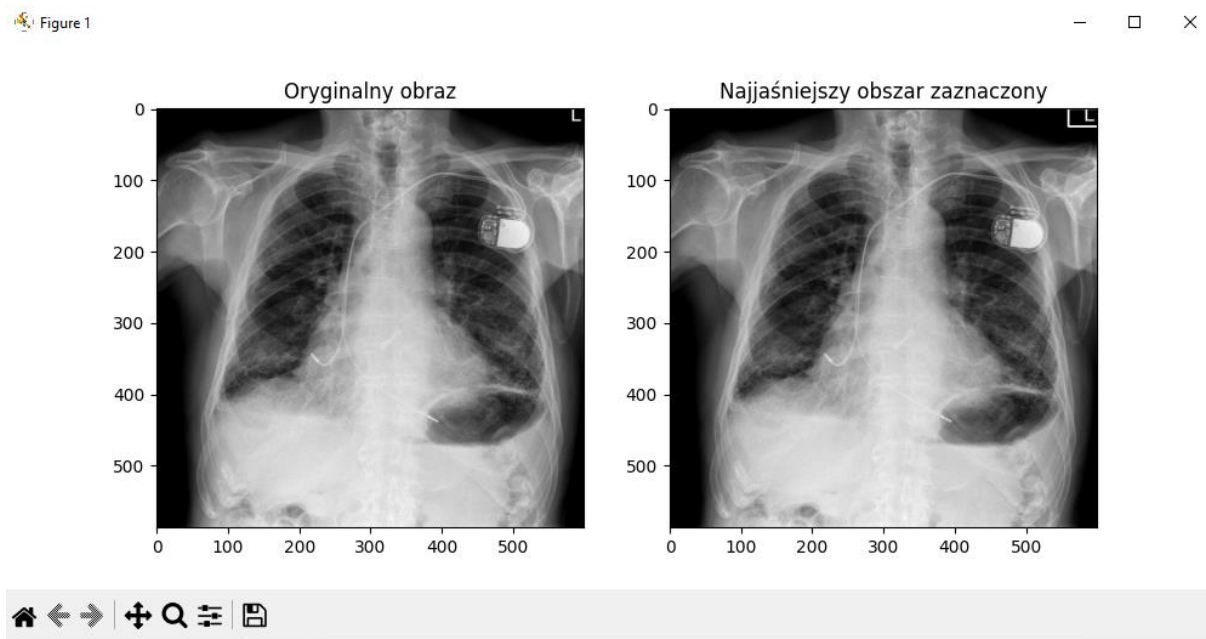
```
# Wczytaj obraz
image = load_image(image_path)

# Znajdź najjaśniejszy obszar
top_left, bottom_right = find_brightest_region(image)

# Narysuj prostokąt wokół najjaśniejszego obszaru
image_with_rect = draw_rectangle(image, top_left, bottom_right)

# Wyświetl obrazy
display_images(image, image_with_rect)
```

Rezultaty:



kod działa ale nie wykrywa tego czego bym chciał, chce aby wykrył mi wszczep kardiologiczny na obrazie RTG takim jak ten



Aby wykryć wszczep kardiologiczny na obrazie RTG, możemy użyć bardziej zaawansowanego podejścia z wykorzystaniem detekcji obiektów. OpenCV oferuje narzędzia do detekcji obiektów, ale możemy też skorzystać z technik głębokiego uczenia, takich jak modele TensorFlow czy PyTorch do detekcji obiektów.

Jednym z popularnych podejść jest użycie modelu YOLO (You Only Look Once), który jest efektywny i szybki w detekcji obiektów. Używanie modeli pretrenowanych na takich zadaniach może być bardziej skuteczne.

Jednakże, aby zastosować prostszą metodę przy użyciu OpenCV, możemy spróbować wykryć wszczep kardiologiczny bazując na charakterystycznych kształtach i jasności.

Poniżej przedstawiam przykład wykorzystania OpenCV do detekcji wszczepu kardiologicznego na podstawie analizy kształtu i jasności:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    return image

def preprocess_image(image):
    # Konwersja do skali szarości
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Zastosowanie filtru Gaussa
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Detekcja krawędzi za pomocą Canny
    edges = cv2.Canny(blurred, 50, 150)
    return edges

def find_implant(image, edges):
    # Znajdowanie konturów
    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
```

```

cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        # Ignorowanie małych konturów
        if cv2.contourArea(contour) < 100:
            continue
        # Przybliżenie konturu prostokątem
        x, y, w, h = cv2.boundingRect(contour)
        # Możemy tu zastosować dodatkowe warunki, aby wykryć wszczep (np.
        wielkość, proporcje itp.)
        if w > 20 and h > 20: # Te warunki można dostosować do wymiarów
            wszczepu
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    return image

def display_image(image):
    # Wyświetlanie obrazu
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

if __name__ == "__main__":
    # Ścieżka do pliku obrazu
    image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane2.jpg'

    # Wczytaj obraz
    image = load_image(image_path)

    # Przetwórz obraz
    edges = preprocess_image(image)

    # Znajdź i zaznacz wszczep kardiologiczny
    image_with_implant = find_implant(image, edges)

    # Wyświetl obraz
    display_image(image_with_implant)

```

W tym kodzie:

1. Wczytujemy obraz i konwertujemy go do skali szarości.
2. Zastosowaliśmy filtr Gaussa, aby wygładzić obraz.
3. Użyliśmy algorytmu Canny do detekcji krawędzi.
4. Znaleźliśmy kontury w obrazie i zazaczyliśmy te, które mają odpowiedni rozmiar, co może odpowiadać wszczepowi kardiologicznemu.

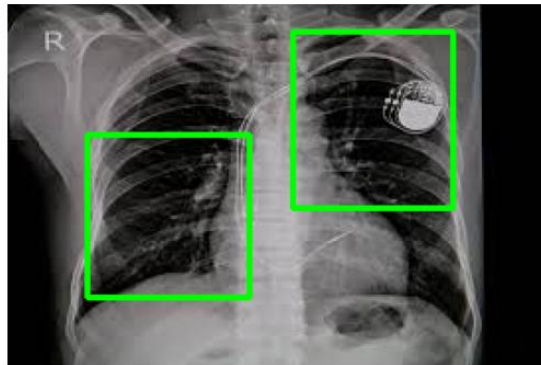
Możliwe, że będziesz musiał dostosować parametry detekcji krawędzi i warunki wykrywania konturów, aby lepiej odpowiadały twojemu obrazowi RTG i specyfikacji wszczepu kardiologicznego.

Jeśli potrzebujesz bardziej precyzyjnego wykrywania, rozważ użycie głębokiego uczenia z pretrenowanymi modelami detekcji obiektów. Modele takie jak YOLO, SSD

czy Faster R-CNN są w stanie dokładniej wykrywać obiekty na obrazach medycznych, ale wymagają większej ilości zasobów do treningu i inferencji.

kod działa ale wykrywa dwa obiekty oraz sam prostokąt jest zbyt duży

Figure 1



Dziękuję za zdjęcie i informację zwrotną. Poprawmy kod tak, aby:

1. Ulepszyć kryteria wykrywania, aby znaleźć tylko wszczep kardiologiczny.
2. Zmniejszyć rozmiar prostokąta do bardziej odpowiedniego rozmiaru.

Spróbujmy użyć bardziej specyficznych kryteriów wykrywania, takich jak filtracja konturów na podstawie proporcji boków i wielkości konturów. Oto poprawiony kod:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    return image

def preprocess_image(image):
    # Konwersja do skali szarości
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Zastosowanie filtru Gaussa
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Detekcja krawędzi za pomocą Canny
    edges = cv2.Canny(blurred, 50, 150)
    return edges

def find_implant(image, edges):
```

```

# Znajdowanie konturów
contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
for contour in contours:
    # Ignorowanie małych konturów
    if cv2.contourArea(contour) < 100:
        continue
    # Przybliżenie konturu prostokątem
    x, y, w, h = cv2.boundingRect(contour)

    # Warunki na podstawie proporcji boków i wielkości konturu
    aspect_ratio = w / float(h)
    if 0.8 < aspect_ratio < 1.2 and 20 < w < 100 and 20 < h < 100: #
Te warunki można dostosować do wymiarów wszczepu
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    return image

def display_image(image):
    # Wyświetlanie obrazu
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

if __name__ == "__main__":
    # Ścieżka do pliku obrazu
    image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane2.jpg'

    # Wczytaj obraz
    image = load_image(image_path)

    # Przetwórz obraz
    edges = preprocess_image(image)

    # Znajdź i zaznacz wszczep kardiologiczny
    image_with_implant = find_implant(image, edges)

    # Wyświetl obraz
    display_image(image_with_implant)

```

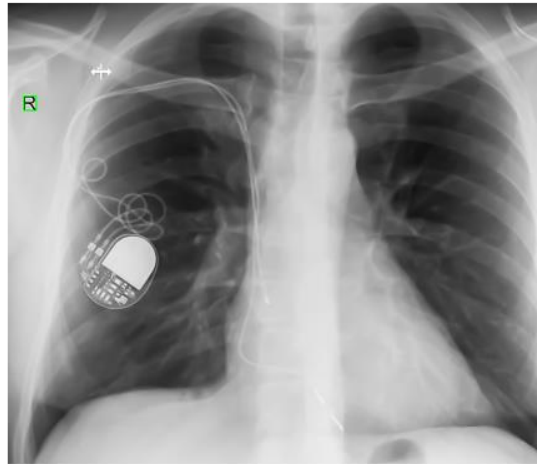
W tym kodzie:

1. Dodaliśmy warunki filtracji na podstawie proporcji boków (**aspect_ratio**) oraz wielkości prostokąta (**w** i **h**), aby lepiej dopasować się do kształtu i rozmiaru wszczepu kardiologicznego.
2. Zmniejszyliśmy rozmiar prostokąta, aby bardziej precyzyjnie otaczał wszczep.

Jeśli potrzebne będzie dalsze dostosowanie, możesz zmieniać wartości w warunkach **aspect_ratio**, **w**, i **h** w zależności od specyficznych cech wszczepu na obrazie RTG.

kod dalej nie działa najlepiej, musisz zastosować inne podejście, w obrazie tego typu zaznacza kompletnie zły fragment

Figure 1



Zrozumiałe, że podejście oparte na prostej analizie kształtu i jasności może nie być wystarczające do precyzyjnego wykrywania wszczepów kardiologicznych na obrazach RTG. Lepszym podejściem byłoby wykorzystanie algorytmów głębokiego uczenia, takich jak YOLO (You Only Look Once) lub SSD (Single Shot Multibox Detector), które są bardziej zaawansowane i mogą lepiej radzić sobie z takim zadaniem.

W tej sekcji przedstawię, jak można użyć modelu YOLO do detekcji wszczepu kardiologicznego. Wykorzystamy pretrenowany model YOLOv3, który jest powszechnie stosowany do detekcji obiektów.

Poniżej znajduje się kod wykorzystujący bibliotekę `opencv-python` do detekcji obiektów za pomocą modelu YOLOv3.

Kroki:

1. Pobierz pliki konfiguracyjne YOLOv3 (`yolov3.cfg`, `yolov3.weights`) oraz plik klas (`coco.names`).
2. Umieść je w odpowiednim katalogu.
3. Zmodyfikuj poniższy kod zgodnie ze ścieżkami do plików konfiguracyjnych i wag.

YOLOv3 Files:

- [yolov3.cfg](#)

- yolov3.weights
- [coco.names](#)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def load_yolo_model(config_path, weights_path, classes_path):
    net = cv2.dnn.readNet(weights_path, config_path)
    with open(classes_path, "r") as f:
        classes = [line.strip() for line in f.readlines()]
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]
    return net, classes, output_layers

def detect_objects(net, output_layers, image):
    height, width, channels = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0),
True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)
    return outs, width, height

def get_boxes(outs, width, height, conf_threshold=0.5, nms_threshold=0.4):
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > conf_threshold:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
    indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,
nms_threshold)
```

chciałbym jednak żebyśmy pozostali przy w miarę prostym kodzie w języku python

Jeśli chcesz pozostać przy prostszym kodzie w Pythonie, spróbujmy zastosować bardziej specyficzne podejście do przetwarzania obrazu, takie jak wykrywanie jasnych okrągłych kształtów, które mogą odpowiadać wszczepom kardiologicznym. Użyjemy tutaj techniki wykrywania okręgów za pomocą transformacji Hougha.

Oto jak możesz to zrobić:

1. Wczytaj obraz i przekonwertuj go do skali szarości.
2. Zastosuj filtr Gaussa, aby wygładzić obraz.
3. Użyj funkcji `HoughCircles`, aby wykryć okręgi, które mogą odpowiadać wszczepom.

Poniżej znajduje się przykładowy kod:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    return image

def preprocess_image(image):
    # Konwersja do skali szarości
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Zastosowanie filtru Gaussa
    blurred = cv2.GaussianBlur(gray, (9, 9), 2)
    return blurred

def find_implant(image, gray_image):
    # Wykrywanie okręgów za pomocą transformacji Hougha
    circles = cv2.HoughCircles(gray_image, cv2.HOUGH_GRADIENT, dp=1.2,
minDist=50, param1=50, param2=30, minRadius=10,
maxRadius=100)

    if circles is not None:
        circles = np.round(circles[0, :]).astype("int")
        for (x, y, r) in circles:
            # Rysowanie okręgów wokół wykrytych okręgów
            cv2.circle(image, (x, y), r, (0, 255, 0), 4)
            # Rysowanie prostokąta wokół wykrytych okręgów
            cv2.rectangle(image, (x - r, y - r), (x + r, y + r), (0, 128,
255), 2)
        return image

def display_image(image):
    # Wyświetlanie obrazu
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

if __name__ == "__main__":
    # Ścieżka do pliku obrazu
    image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane2.jpg'

    # Wczytaj obraz
```

```

image = load_image(image_path)

# Przetwórz obraz
gray_image = preprocess_image(image)

# Znajdź i zaznacz wszczep kardiologiczny
image_with_implant = find_implant(image, gray_image)

# Wyświetl obraz
display_image(image_with_implant)

```

W tym kodzie:

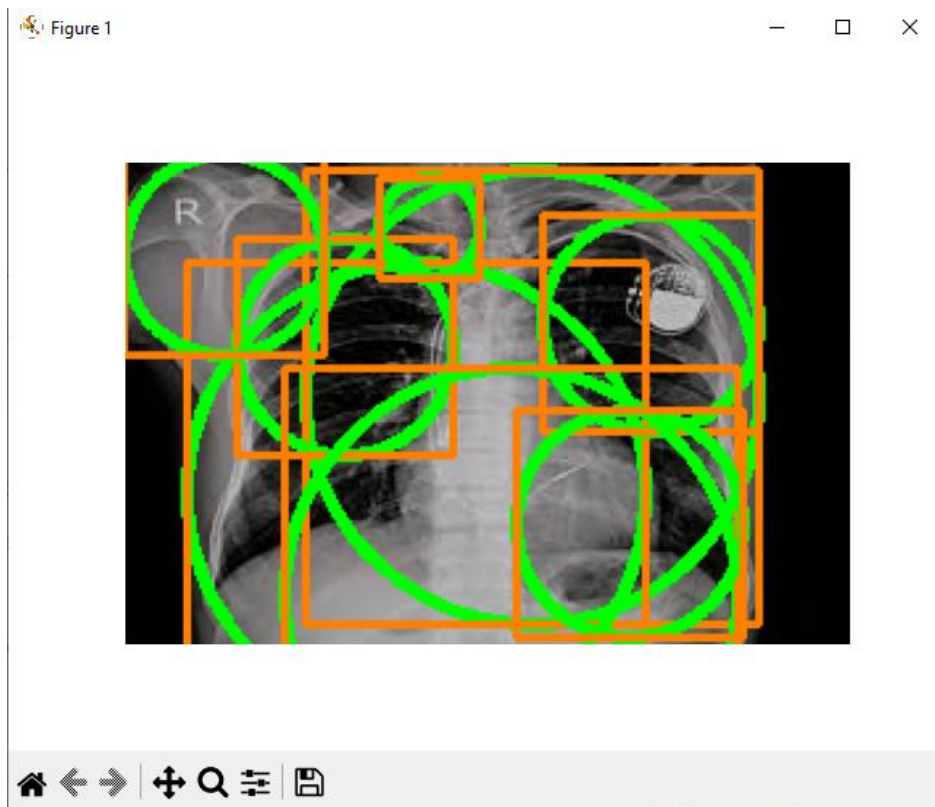
1. Wczytujemy obraz i konwertujemy go do skali szarości.
2. Stosujemy filtr Gaussa, aby wygładzić obraz.
3. Używamy transformacji Hougha do wykrywania okręgów, które mogą odpowiadać wszczepom kardiologicznym.
4. Jeśli wykryjemy okręgi, rysujemy wokół nich okręgi i prostokąty.

Możesz dostosować parametry funkcji **HoughCircles**, takie jak **dp**, **minDist**, **param1**, **param2**, **minRadius**, i **maxRadius**, aby lepiej dopasować wykrywanie do swojego obrazu.

Rezultaty:

Figure 1





Nowy Prompt:

napisz kod w języku python który będzie na obrazie wykrywał najjaśniejszy półokrąg

Aby napisać kod w języku Python, który wykrywa najjaśniejszy półokrąg na obrazie, możemy użyć bibliotek takich jak OpenCV i NumPy. Proces ten można podzielić na kilka kroków:

1. Wczytanie obrazu.
2. Przekształcenie obrazu na skali szarości.
3. Wykrycie krawędzi na obrazie.
4. Zastosowanie algorytmu HoughCircles do wykrywania okręgów.
5. Przefiltrowanie wykrytych okręgów do półokręgów i znalezienie najjaśniejszego.

Poniżej znajduje się przykładowy kod realizujący te kroki:

```
import cv2
import numpy as np

def detect_brightest_semi_circle(image_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Wykrywanie krawędzi
```

```

edges = cv2.Canny(gray, 50, 150, apertureSize=3)

# Wykrywanie okręgów za pomocą transformacji Hougha
circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, dp=1, minDist=20,
param1=50, param2=30, minRadius=0, maxRadius=0)

if circles is not None:
    circles = np.uint16(np.around(circles))
    max_avg_intensity = -1
    brightest_semi_circle = None

    for circle in circles[0, :]:
        x, y, r = circle
        mask = np.zeros_like(gray)
        cv2.circle(mask, (x, y), r, 255, thickness=-1)

        # Mask for upper semi-circle
        mask_upper = np.zeros_like(mask)
        cv2.ellipse(mask_upper, (x, y), (r, r), 0, 0, 180, 255,
thickness=-1)
        mask_upper = cv2.bitwise_and(mask, mask_upper)

        # Mask for lower semi-circle
        mask_lower = np.zeros_like(mask)
        cv2.ellipse(mask_lower, (x, y), (r, r), 0, 180, 360, 255,
thickness=-1)
        mask_lower = cv2.bitwise_and(mask, mask_lower)

        # Calculate average intensity for upper and lower semi-circles
        mean_upper = cv2.mean(gray, mask=mask_upper)[0]
        mean_lower = cv2.mean(gray, mask=mask_lower)[0]

        if mean_upper > max_avg_intensity:
            max_avg_intensity = mean_upper
            brightest_semi_circle = (x, y, r, 'upper')

        if mean_lower > max_avg_intensity:
            max_avg_intensity = mean_lower
            brightest_semi_circle = (x, y, r, 'lower')

    if brightest_semi_circle:
        x, y, r, position = brightest_semi_circle
        color = (0, 255, 0) # zielony kolor
        thickness = 2

        # Rysowanie najjaśniejszego półokręgu
        if position == 'upper':
            cv2.ellipse(image, (x, y), (r, r), 0, 0, 180, color,
thickness)
        else:
            cv2.ellipse(image, (x, y), (r, r), 0, 180, 360, color,
thickness)

        cv2.imshow('Brightest Semi-Circle', image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

# Przykładowe użycie
detect_brightest_semi_circle('sciezka_do_obrazka.jpg')

```

Wyjaśnienie kroków:

1. **Wczytanie obrazu:** Obraz jest wczytywany przy użyciu funkcji `cv2.imread()`, a następnie konwertowany do skali szarości.
2. **Wykrywanie krawędzi:** Krawędzie są wykrywane za pomocą algorytmu Canny'ego.
3. **Wykrywanie okręgów:** Okręgi są wykrywane za pomocą transformacji Hougha.
4. **Przefiltrowanie do półokręgów:** Każdy wykryty okrąg jest podzielony na półokręgi (górny i dolny) i obliczana jest średnia jasność dla każdego z nich.
5. **Wybór najjaśniejszego półokręgu:** Spośród wszystkich półokręgów wybierany jest ten o najwyższej średniej jasności i rysowany na oryginalnym obrazie.

Pamiętaj, aby zmienić `'sciezka_do_obrazka.jpg'` na ścieżkę do obrazu, na którym chcesz wykrywać półokręgi.

Kod nie działał

Wygląda na to, że problem jest związany z funkcją `cv2.imshow()`, która może nie działać poprawnie w niektórych środowiskach, takich jak w przypadku korzystania z niektórych instalacji OpenCV bez odpowiednich bibliotek GUI.

Możesz spróbować zapisać wynikowy obraz do pliku zamiast wyświetlać go bezpośrednio. Poniżej znajduje się zmodyfikowany kod, który zapisuje wynikowy obraz do pliku:

```
import cv2
import numpy as np

def detect_brightest_semi_circle(image_path, output_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    if image is None:
        print(f"Nie można wczytać obrazu z {image_path}")
        return
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Wykrywanie krawędzi
    edges = cv2.Canny(gray, 50, 150, apertureSize=3)

    # Wykrywanie okręgów za pomocą transformacji Hougha
    circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, dp=1, minDist=20,
param1=50, param2=30, minRadius=0, maxRadius=0)

    if circles is not None:
        circles = np.uint16(np.around(circles))
        max_avg_intensity = -1
        brightest_semi_circle = None
```

```

for circle in circles[0, :]:
    x, y, r = circle
    mask = np.zeros_like(gray)
    cv2.circle(mask, (x, y), r, 255, thickness=-1)

    # Mask for upper semi-circle
    mask_upper = np.zeros_like(mask)
    cv2.ellipse(mask_upper, (x, y), (r, r), 0, 0, 180, 255,
thickness=-1)
    mask_upper = cv2.bitwise_and(mask, mask_upper)

    # Mask for lower semi-circle
    mask_lower = np.zeros_like(mask)
    cv2.ellipse(mask_lower, (x, y), (r, r), 0, 180, 360, 255,
thickness=-1)
    mask_lower = cv2.bitwise_and(mask, mask_lower)

    # Calculate average intensity for upper and lower semi-circles
    mean_upper = cv2.mean(gray, mask=mask_upper)[0]
    mean_lower = cv2.mean(gray, mask=mask_lower)[0]

    if mean_upper > max_avg_intensity:
        max_avg_intensity = mean_upper
        brightest_semi_circle = (x, y, r, 'upper')

    if mean_lower > max_avg_intensity:
        max_avg_intensity = mean_lower
        brightest_semi_circle = (x, y, r, 'lower')

    if brightest_semi_circle:
        x, y, r, position = brightest_semi_circle
        color = (0, 255, 0) # zielony kolor
        thickness = 2

        # Rysowanie najjaśniejszego półokręgu
        if position == 'upper':
            cv2.ellipse(image, (x, y), (r, r), 0, 0, 180, color,
thickness)
        else:
            cv2.ellipse(image, (x, y), (r, r), 0, 180, 360, color,
thickness)

        # Zapisz obraz do pliku
        cv2.imwrite(output_path, image)
        print(f"Wynik zapisany jako {output_path}")

# Przykładowe użycie
image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane3.jpg'
output_path = r'C:\Users\jajko\Desktop\jpdzb\wynik.jpg'
detect_brightest_semi_circle(image_path, output_path)

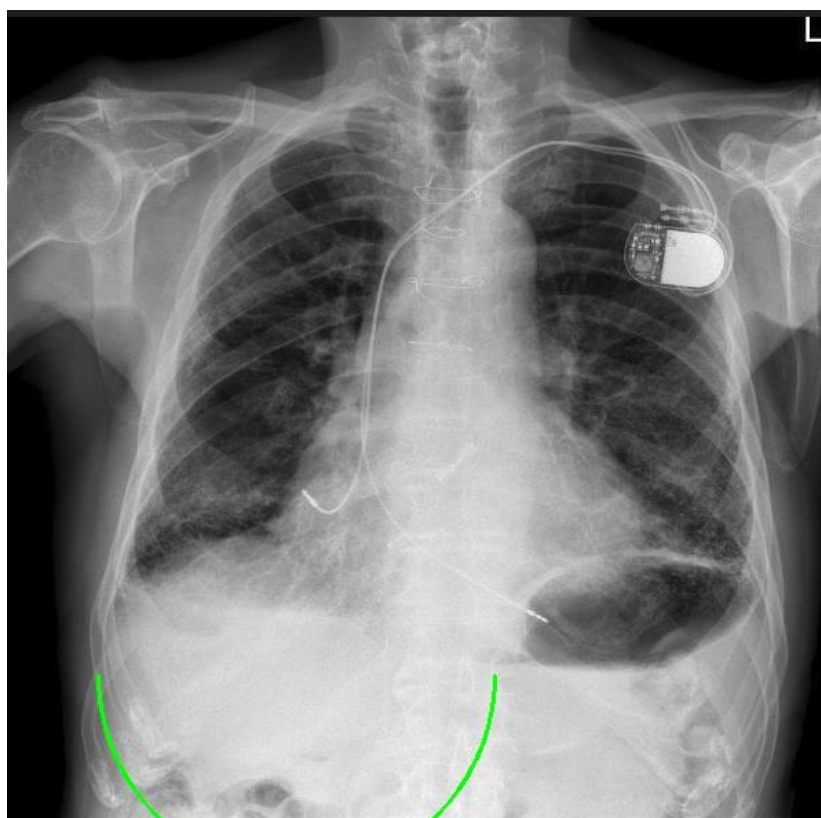
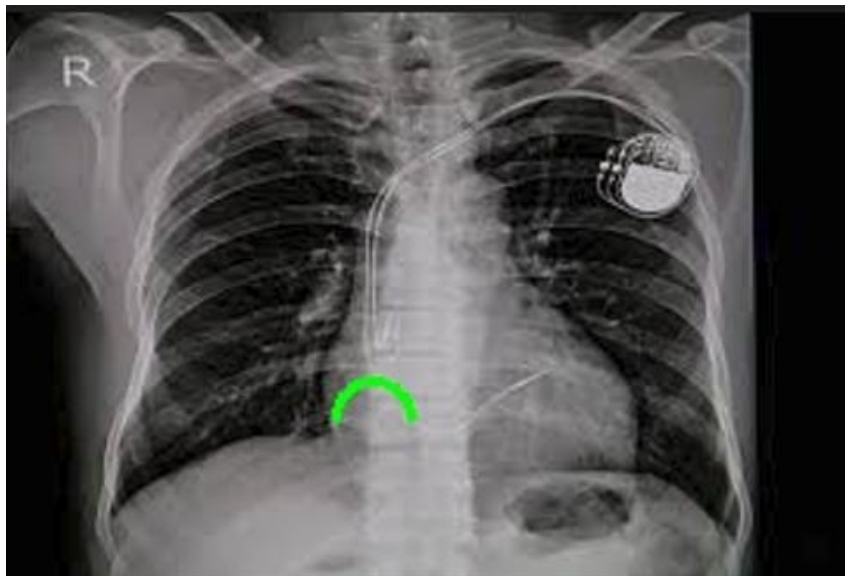
```

Co się zmieniło:

1. **Dodano parametr output_path:** Określa ścieżkę, gdzie zapisany będzie wynikowy obraz.
2. **Zapis obrazu do pliku:** Zamiast wyświetlać obraz, wynikowy obraz jest zapisywany za pomocą funkcji `cv2.imwrite(output_path, image)`.

Teraz kod zapisze wynikowy obraz z najjaśniejszym półokręgiem do podanego pliku, co powinno rozwiązać problem z funkcją `cv2.imshow()`.

Rezultaty:



a teraz napisz kod który wykrywa najjaśniejszy półokrąg ale taki który jest wokół czarnego tła

Aby wykryć najjaśniejszy półokrąg wokół czarnego tła, należy najpierw zidentyfikować obszary obrazu, które mają czarne tło, a następnie wykryć półokręgi, które otaczają te obszary. Następnie można znaleźć półokrąg o najwyższej średniej jasności. Poniżej znajduje się przykładowy kod, który realizuje ten proces:

```
import cv2
import numpy as np

def detect_brightest_semi_circle_around_black(image_path, output_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    if image is None:
        print(f"Nie można wczytać obrazu z {image_path}")
        return
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Binarizacja obrazu, aby znaleźć czarne tło
    _, binary = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY_INV)

    # Wykrywanie konturów
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    max_avg_intensity = -1
    brightest_semi_circle = None

    for contour in contours:
        # Dopasowanie okręgu minimalnego obejmującego kontur
        (x, y), radius = cv2.minEnclosingCircle(contour)
        center = (int(x), int(y))
        radius = int(radius)

        mask = np.zeros_like(gray)
        cv2.circle(mask, center, radius, 255, thickness=-1)

        # Mask for upper semi-circle
        mask_upper = np.zeros_like(mask)
        cv2.ellipse(mask_upper, center, (radius, radius), 0, 0, 180, 255,
thickness=-1)
        mask_upper = cv2.bitwise_and(mask, mask_upper)

        # Mask for lower semi-circle
        mask_lower = np.zeros_like(mask)
        cv2.ellipse(mask_lower, center, (radius, radius), 0, 180, 360, 255,
thickness=-1)
        mask_lower = cv2.bitwise_and(mask, mask_lower)

        # Calculate average intensity for upper and lower semi-circles
        mean_upper = cv2.mean(gray, mask=mask_upper)[0]
        mean_lower = cv2.mean(gray, mask=mask_lower)[0]

        if mean_upper > max_avg_intensity:
            max_avg_intensity = mean_upper
            brightest_semi_circle = (center, radius, 'upper')

        if mean_lower > max_avg_intensity:
            max_avg_intensity = mean_lower
```



```

        brightest_semi_circle = (center, radius, 'lower')

    if brightest_semi_circle:
        center, radius, position = brightest_semi_circle
        color = (0, 255, 0) # zielony kolor
        thickness = 2

        # Rysowanie najjaśniejszego półokręgu
        if position == 'upper':
            cv2.ellipse(image, center, (radius, radius), 0, 0, 180, color,
thickness)
        else:
            cv2.ellipse(image, center, (radius, radius), 0, 180, 360,
color, thickness)

        # Zapisz obraz do pliku
        cv2.imwrite(output_path, image)
        print(f"Wynik zapisany jako {output_path}")

# Przykładowe użycie
image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane3.jpg'
output_path = r'C:\Users\jajko\Desktop\jpdzb\wynik.jpg'
detect_brightest_semi_circle_around_black(image_path, output_path)

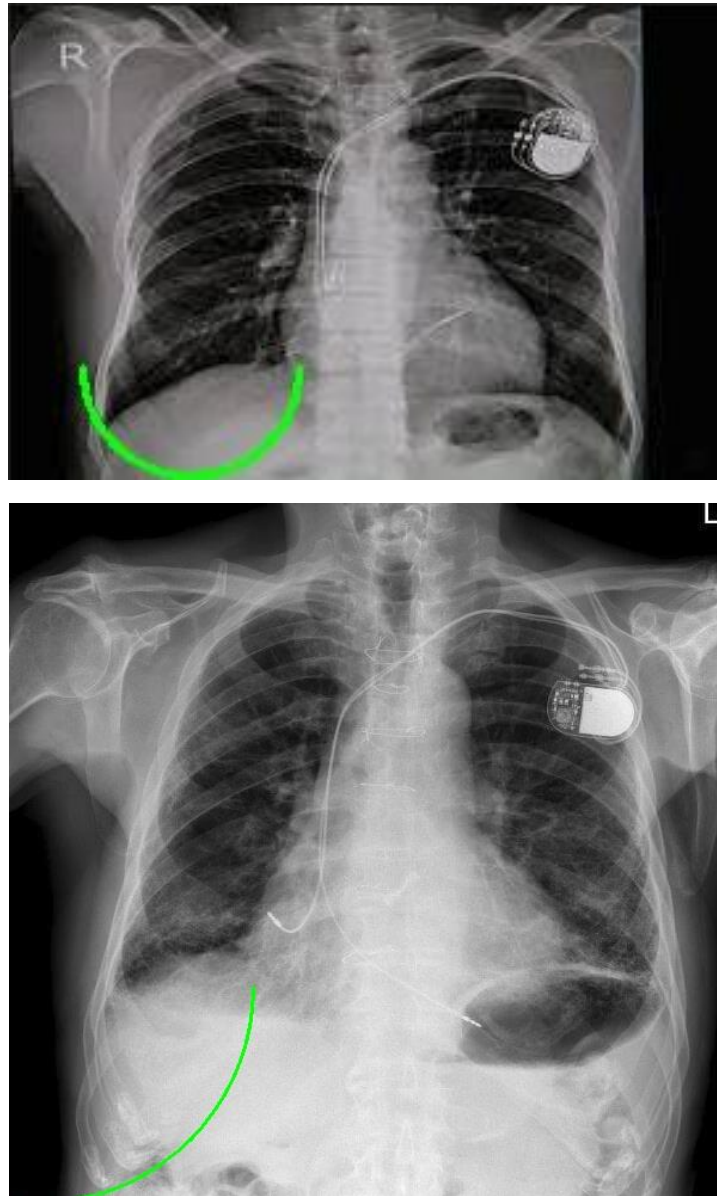
```

Co robi ten kod:

1. **Wczytuje obraz:** Wczytuje obraz z podanej ścieżki.
2. **Binaryzacja obrazu:** Przekształca obraz na czarno-biały, aby zidentyfikować czarne tło.
3. **Wykrywanie konturów:** Znajduje kontury czarnych obszarów.
4. **Dopasowanie okręgu:** Dopasowuje minimalny okrąg obejmujący każdy kontur.
5. **Tworzenie masek półokręgów:** Tworzy maski dla górnych i dolnych półokręgów.
6. **Oblicza średnią jasność:** Oblicza średnią jasność dla półokręgów i wybiera najjaśniejszy.
7. **Rysowanie i zapisywanie:** Rysuje najjaśniejszy półokrąg na oryginalnym obrazie i zapisuje wynik.

Teraz kod wykrywa najjaśniejszy półokrąg wokół czarnego tła i zapisuje wynikowy obraz do pliku.

Rezultaty:



Nowy Prompt:

```
import cv2 import numpy as np from matplotlib import pyplot as plt def
load_image(image_path): # Wczytaj obraz image = cv2.imread(image_path) return
image def preprocess_image(image): # Konwersja do skali szarości gray =
cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Zastosowanie filtru Gaussa blurred =
cv2.GaussianBlur(gray, (5, 5), 0) # Detekcja krawędzi za pomocą Canny edges =
cv2.Canny(blurred, 50, 150) return edges def find_implant(image, edges): #
Znajdowanie konturów contours, _ = cv2.findContours(edges.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) for contour in contours: #
Ignorowanie małych konturów if cv2.contourArea(contour) < 100: continue #
Przybliżenie konturu prostokątem x, y, w, h = cv2.boundingRect(contour) # Możemy tu
zastosować dodatkowe warunki, aby wykryć wszczęp (np. wielkość, proporcje itp.) if w
> 20 and h > 20: # Te warunki można dostosować do wymiarów wszczępu
```

```

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2) return image def
display_image(image): # Wyświetlanie obrazu plt.imshow(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)) plt.axis('off') plt.show() if __name__ == "__main__": # Ścieżka do
pliku obrazu image_path =
r'C:\Users\jajko\Desktop\programowanie_dyrka\pobrane.jpg' # Wczytaj obraz image =
load_image(image_path) # Przetwórz obraz edges = preprocess_image(image) # Znajdź
i zaznacz wszczep kardiologiczny image_with_implant = find_implant(image, edges) #
Wyświetl obraz display_image(image_with_implant)

```

przepisz ten kod tak aby wykrywał tylko jeden element

Oczywiście, możemy zmodyfikować funkcję `find_implant` tak, aby zwracała tylko jeden element. Możemy na przykład zwrócić pierwszy znaleziony wszczep, który spełnia warunki wielkości.

```

import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    return image

def preprocess_image(image):
    # Konwersja do skali szarości
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Zastosowanie filtru Gaussa
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Detekcja krawędzi za pomocą Canny
    edges = cv2.Canny(blurred, 50, 150)
    return edges

def find_implant(image, edges):
    # Znajdowanie konturów
    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Inicjalizacja flagi oznaczającej, czy znaleziono wszczep
    implant_found = False

    for contour in contours:
        # Ignorowanie małych konturów
        if cv2.contourArea(contour) < 100:
            continue
        # Przybliżenie konturu prostokątem
        x, y, w, h = cv2.boundingRect(contour)
        # Sprawdzenie warunków na wszczep
        if w > 20 and h > 20: # Te warunki można dostosować do wymiarów
wszczepu
            # Zaznaczenie prostokąta na obrazie
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
            # Ustawienie flagi, że wszczep został znaleziony
            implant_found = True

```

```

        # Przerwij pętlę po znalezieniu pierwszego wszczepu
        break

    # Jeśli wszczep został znaleziony, zwróć zaznaczony obraz, w przeciwnym
    razie zwróć oryginalny obraz
    if implant_found:
        return image
    else:
        return None

def display_image(image):
    # Wyświetlanie obrazu
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

if __name__ == "__main__":
    # Ścieżka do pliku obrazu
    image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane3.jpg'

    # Wczytaj obraz
    image = load_image(image_path)

    # Przetwórz obraz
    edges = preprocess_image(image)

    # Znajdź i zaznacz wszczep kardiologiczny
    image_with_implant = find_implant(image, edges)

    if image_with_implant is not None:
        # Wyświetl obraz z zaznaczonym wszczepem
        display_image(image_with_implant)
    else:
        print("Nie znaleziono wszczepu.")

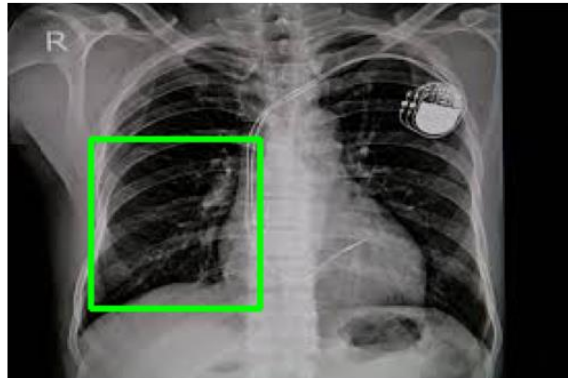
```

W tej modyfikacji dodałem blok `if` w funkcji `__main__`, który sprawdza, czy zwrócony obraz `image_with_implant` nie jest pusty (`None`). Jeśli obraz nie jest pusty, to znaczy, że wszczep został znaleziony i wyświetlamy obraz z zaznaczonym wszczepem. W przeciwnym razie, jeśli obraz jest pusty, wypisujemy komunikat informujący, że wszczep nie został znaleziony.

Rezultaty:

Figure 1

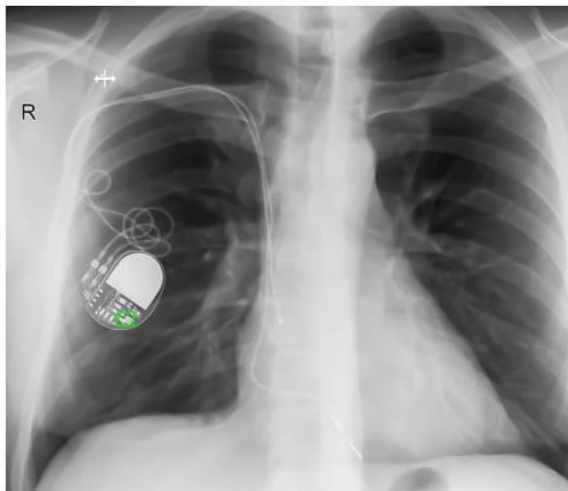
— □ ×



🏠 ⬅ ➡ 🔍 📏 📄

Figure 1

— □ ×



🏠 ⬅ ➡ 🔍 📏 📄

Figure 1

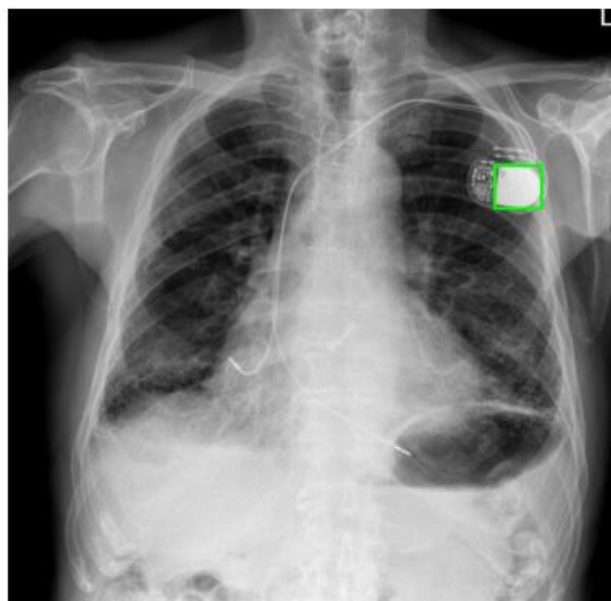
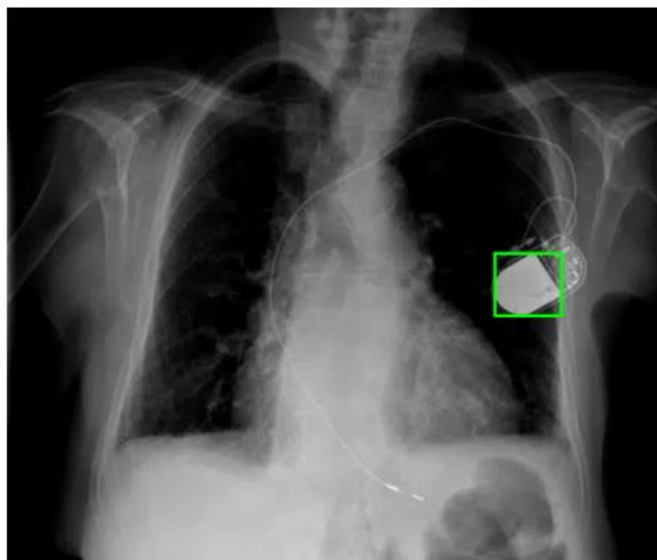


Figure 1



Ostatecznie użyty kod:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def load_image(image_path):
    # Wczytaj obraz
    image = cv2.imread(image_path)
    return image

def preprocess_image(image):
    # Konwersja do skali szarości
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Zastosowanie filtru Gaussa
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Detekcja krawędzi za pomocą Canny
    edges = cv2.Canny(blurred, 50, 150)
    return edges

def find_implant(image, edges):
    # Znajdowanie konturów
    contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        # Ignorowanie małych konturów
        if cv2.contourArea(contour) < 100:
            continue
        # Przybliżenie konturu prostokątem
        x, y, w, h = cv2.boundingRect(contour)
        # Możemy tu zastosować dodatkowe warunki, aby wykryć wszczep (np.
wielkość, proporcje itp.)
        if w > 20 and h > 20: # Te warunki można dostosować do wymiarów
wszczepu
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    return image

def display_image(image):
    # Wyświetlanie obrazu
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

if __name__ == "__main__":
    # Ścieżka do pliku obrazu
    image_path = r'C:\Users\jajko\Desktop\jpdzb\pobrane2.jpg'

    # Wczytaj obraz
    image = load_image(image_path)

    # Przetwórz obraz
    edges = preprocess_image(image)

    # Znajdź i zaznacz wszczep kardiologiczny
    image_with_implant = find_implant(image, edges)
```

```
# Wyświetl obraz  
display_image(image_with_implant)
```

Rezultaty:

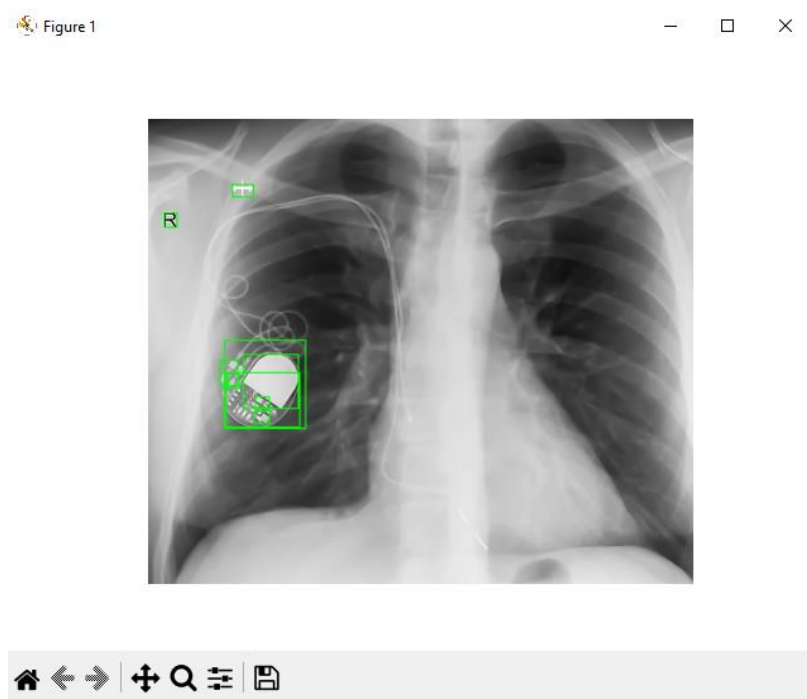
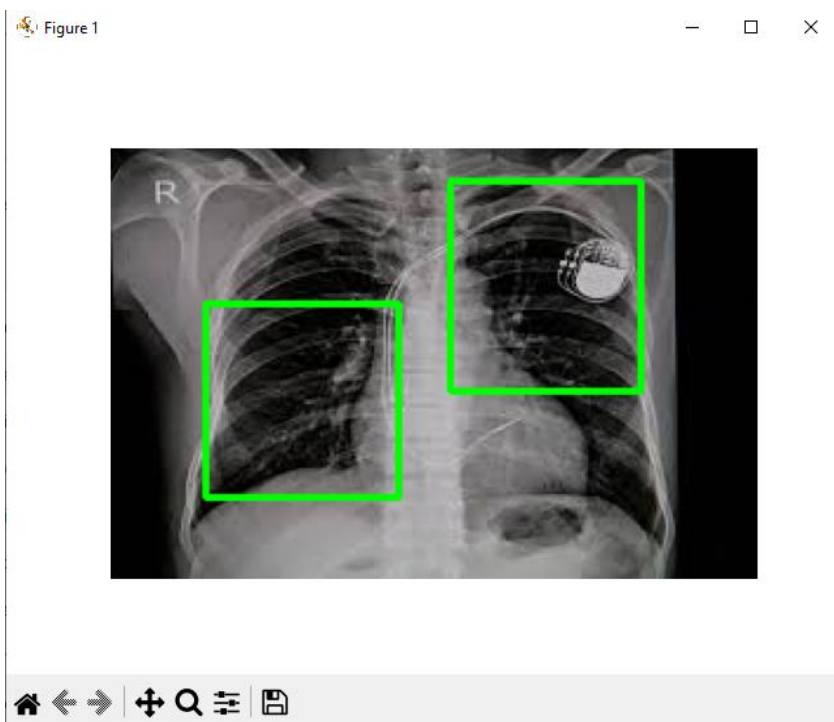


Figure 1

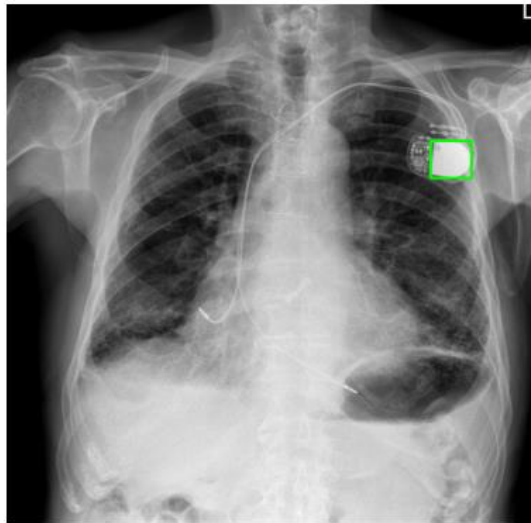
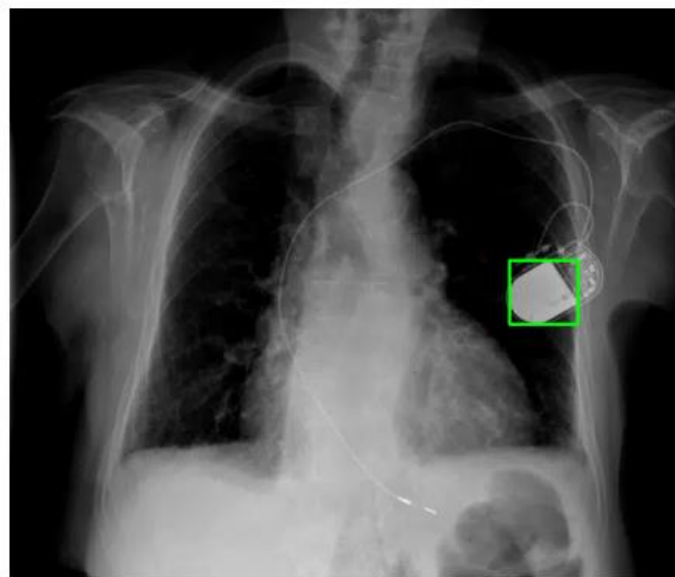


Figure 1



Wyjaśnienie działania kodu:

Importowanie bibliotek

- **cv2**: OpenCV, biblioteka do przetwarzania obrazów.

- **np**: NumPy, biblioteka do operacji na tablicach wielowymiarowych.
- **plt**: Matplotlib, biblioteka do wizualizacji danych.

Funkcja load_image

- **Funkcja load_image**: Wczytuje obraz z podanej ścieżki `image_path` przy użyciu funkcji `cv2.imread`, która zwraca obraz jako macierz NumPy.

Funkcja preprocess_image

- **Konwersja do skali szarości**: `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` konwertuje obraz z przestrzeni kolorów BGR (domyślna dla OpenCV) na skalę szarości.
- **Zastosowanie filtru Gaussa**: `cv2.GaussianBlur(gray, (5, 5), 0)` stosuje filtr Gaussa do wygładzenia obrazu, co pomaga w redukcji szumu i poprawie detekcji krawędzi.
- **Detekcja krawędzi za pomocą Canny**: `cv2.Canny(blurred, 50, 150)` używa algorytmu detekcji krawędzi Canny'ego do znalezienia krawędzi na przetworzonym obrazie. Parametry 50 i 150 to dolny i górny próg histerezy.

Funkcja find_implant

- **Znajdowanie konturów**: `cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)` znajduje kontury na obrazie krawędzi. `cv2.RETR_EXTERNAL` oznacza, że zwracane są tylko zewnętrzne kontury, a `cv2.CHAIN_APPROX_SIMPLE` upraszcza kontury, przechowując tylko punkty narożne.
- **Iteracja po konturach**: Pętla `for contour in contours` iteruje po znalezionych konturach.
- **Ignorowanie małych konturów**: `if cv2.contourArea(contour) < 100` ignoruje kontury o małej powierzchni.
- **Przybliżenie konturu prostokątem**: `x, y, w, h = cv2.boundingRect(contour)` oblicza prostokąt otaczający kontur.
- **Sprawdzenie warunków na wszczep**: `if w > 20 and h > 20` sprawdza, czy wymiary prostokąta są wystarczająco duże.
- **Zaznaczenie prostokąta na obrazie**: `cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)` rysuje zielony prostokąt na obrazie wokół wykrytego wszczepu.

Funkcja display_image

- **Wyświetlanie obrazu**: `plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))` konwertuje obraz z BGR na RGB (potrzebne, ponieważ Matplotlib używa

przestrzeni kolorów RGB) i wyświetla go. `plt.axis('off')` wyłącza osie, a `plt.show()` wyświetla obraz.

Wywołanie programu

- **Ścieżka do pliku obrazu:** `image_path` przechowuje ścieżkę do pliku obrazu, który ma być przetworzony.
- **Wczytaj obraz:** `image = load_image(image_path)` wczytuje obraz z podanej ścieżki.
- **Przetwórz obraz:** `edges = preprocess_image(image)` przetwarza obraz, aby uzyskać krawędzie.
- **Znajdź i zaznacz wszczep kardiologiczny:** `image_with_implant = find_implant(image, edges)` znajduje wszczep na obrazie i zaznacza go prostokątem.
- **Wyświetl obraz:** `display_image(image_with_implant)` wyświetla przetworzony obraz z zaznaczonym wszczepem.