```
close all;

% Read the image
image = imread('teeth_sample1.png');

% Convert to grayscale if not already
if size(image, 3) == 3
    image_gray = rgb2gray(image);
else
    image_gray = image;
end

% Enhance contrast using adaptive histogram equalization
image_enhanced = adapthisteq(image_gray);

% Divide the image into 5 vertical parts
num_vertical_parts =5;
vertical_parts = round(linspace(1, size(image_enhanced, 2), num_vertical_parts +
1));

% Initialize variables to store center points
center_points_x = [];
center_points_y = [];

% Initialize figure
figure;

% Iterate over vertical parts
for i = 1:num_vertical_parts
    % Extract the current vertical part
    part = image_enhanced(:, vertical_parts(i):vertical_parts(i+1));

    % Divide the part into horizontal blocks
    block_size = 20;
    num_blocks = size(part, 1) / block_size;

    % Initialize variables to store darkest region
    darkest_intensity = 255;
    darkest_index = 1;

    % Iterate over horizontal blocks
    for j = 1:num_blocks
        % Extract the current block
        block = part((j-1)*block_size + 1 : j*block_size, :);

        % Compute the average intensity of the block
        avg_intensity = mean(block(:));

        % Check if this block has a darker region
        if avg_intensity < darkest_intensity && j > 1 && j < num_blocks
            darkest_intensity = avg_intensity;
            darkest_index = j;
        end
    end

    % Calculate the center point of the darkest region
    center_x = mean([vertical_parts(i), vertical_parts(i+1)]);
    center_y = (darkest_index - 0.5) * block_size;
```

```
    % Store center points
    center_points_x = [center_points_x, center_x];
    center_points_y = [center_points_y, center_y];

    % Plot the current part
    subplot(1, num_vertical_parts, i);
    imshow(part);
    hold on;

    % Draw a line on the darkest region (if not too close to edges)
    if darkest_index > 1 && darkest_index < num_blocks
        line([1, size(part, 2)], [(darkest_index - 0.5) * block_size,
(darkest_index - 0.5) * block_size], 'Color', 'r', 'LineWidth', 2);
    end

    title(['Part ', num2str(i)]);
    hold off;
end

% Fit a second-degree polynomial curve to the center points
p = polyfit(center_points_x, center_points_y, 2);

% Generate x-values for the curve
curve_x = linspace(vertical_parts(1), vertical_parts(end), 100);

% Compute y-values using the polynomial curve
curve_y = polyval(p, curve_x);

% Plot the curve connecting the center points
figure;
imshow(image_enhanced);
hold on;
plot(curve_x, curve_y, 'g', 'LineWidth', 2);

% Threshold for sum of intensities
intensity_threshold1 = 20000;


% Draw lines perpendicular to the curve on upper jaw
for i = 1:6:numel(curve_x)
    % Get coordinates of current point on the curve
    x0 = curve_x(i);
    y0 = curve_y(i);


    % Compute slope of the curve at this point
    slope_curve = polyval(polyder(p), x0);

    % Compute slope of the perpendicular line
    slope_perpendicular = -1 / slope_curve;

    % Compute y-intercept of the perpendicular line
    y_intercept_perpendicular = y0 - slope_perpendicular * x0;

    % Define the length of the lines perpendicular to the curve
line_length = 150; % Adjust as needed

% Compute the change in x and y for the line length
delta_x = line_length / sqrt(1 + slope_perpendicular^2);
```

```
delta_y = slope_perpendicular * delta_x;

% Compute start and end points for the line
x_start = x0 - delta_x;
y_start = y0 - delta_y;
x_end = x0 + delta_x;
y_end = y0 + delta_y;


    % Ensure upper line stays only below the curve
    if y_start > y0
        y_start = y0;
        x_start = (y_start - y_intercept_perpendicular) / slope_perpendicular;
    end

    if y_end > y0
        y_end = y0;
        x_end = (y_end - y_intercept_perpendicular) / slope_perpendicular;
    end
    % Compute intensity along the line
  intensity_line = sum(improfile(image_enhanced, [x_start, x_end], [y_start,
y_end]));

    % Draw lines only where sum of intensities is less than the threshold
    if intensity_line < intensity_threshold1
        line([x_start, x_end], [y_start, y_end], 'Color', 'y', 'LineWidth', 2);
    end
end

intensity_threshold2 = 20000;% Adjust as needed

% Draw lines perpendicular to the curve on lower jaw
for i = 1:7:numel(curve_x)
    % Get coordinates of current point on the curve
    x0 = curve_x(i);
    y0 = curve_y(i);


    % Compute slope of the curve at this point
    slope_curve = polyval(polyder(p), x0);

    % Compute slope of the perpendicular line
    slope_perpendicular = -1 / slope_curve;

    % Compute y-intercept of the perpendicular line
    y_intercept_perpendicular = y0 - slope_perpendicular * x0;

    % Define the length of the lines perpendicular to the curve
line_length = 150; % Adjust as needed

% Compute the change in x and y for the line length
delta_x = line_length / sqrt(1 + slope_perpendicular^2);
delta_y = slope_perpendicular * delta_x;

% Compute start and end points for the line
x_start = x0 - delta_x;
y_start = y0 - delta_y;
x_end = x0 + delta_x;
y_end = y0 + delta_y;
```

```
    % Ensure lower line stays only below the curve
    if y_start < y0
        y_start = y0;
        x_start = (y_start - y_intercept_perpendicular) / slope_perpendicular;
    end

    if y_end < y0
        y_end = y0;
        x_end = (y_end - y_intercept_perpendicular) / slope_perpendicular;
    end
    % Compute intensity along the line
  intensity_line = sum(improfile(image_enhanced, [x_start, x_end], [y_start,
y_end]));

    % Draw lines only where sum of intensities is less than the threshold
    if intensity_line < intensity_threshold2
        line([x_start, x_end], [y_start, y_end], 'Color', 'r', 'LineWidth', 2);
    end
end

hold off;
```