Andrew Kozempel

Professor Avanzato

CMPSC 497

21 September 2023

**LAB #2: Line Detection using Computer Vision**

**Part 1:**

**Objective:** For each of the 3 test cases, create a program that will detect whether a line is to the left, right, or in the center.

**MATLAB Script:**

| **%{ Andrew Kozempel CMPSC 497 Fall 2023 LAB #2: Line Detection using Computer Vision %}  % % PART 1 %  fprintf('\nPart 1\n');  % load images images = {'center.bmp', 'left.bmp', 'right.bmp'};  % loop through images for i = 1:length(images)   % load original image  RGB = imread(images{i});   % convert to grayscale  gray = rgb2gray(RGB);   % get rid of "salt and pepper"  filtered = medfilt2(gray, [15 15]);  figure, imshow(filtered);   % create binary image  bw = ~im2bw(filtered, 100/255);  figure, imshow(bw);   % set regions: left (40%), center (20%), right (40%)  [rows, cols] = size(bw);  left\_region = bw(:, 1:floor(2\*cols/5));  center\_region = bw(:, floor(2\*cols/5)+1:floor(3\*cols/5));  right\_region = bw(:, floor(3\*cols/5)+1:end);    % calculate sum of regions  left\_sum = sum(left\_region(:));  center\_sum = sum(center\_region(:));  right\_sum = sum(right\_region(:));   % print totals  fprintf('\nLeft Sum: %d, Center Sum: %d, Right Sum: %d\n', left\_sum, center\_sum, right\_sum);   % determine which region the line is in and give command  if center\_sum > left\_sum && center\_sum > right\_sum  direction = 'Go Straight';  elseif left\_sum > center\_sum && left\_sum > right\_sum  direction = 'Turn Left';  else  direction = 'Turn Right';  end   % print direction  fprintf('%s: %s\n', images{i}, direction); end** |
| --- |

**Results:**

| **center.bmp** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **center.bmp: Go Straight** |

| **left.bmp** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **left.bmp: Turn Left** |

| **right.bmp** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **right.bmp: Turn Right** |

**Part 2:**

**Objective:** For each of the 5 test cases, create a program that will detect whether a line is to the far left, slightly left, far right, slightly right, or in the center.

**MATLAB Script:**

| % % PART 2 %  fprintf('\nPart 2\n');  new\_images = {'farleft.jpg', 'slightleft.jpg', 'center.jpg', 'slightright.jpg', 'farright.jpg', };  % loop through images for i = 1:length(new\_images)   % load original image  RGB = imread(new\_images{i});  figure, imshow(RGB);   % convert to grayscale  gray = rgb2gray(RGB);   % get rid of "salt and pepper"  filtered = medfilt2(gray, [15 15]);   % create binary image ( less than 90 = 0(black))  bw = ~im2bw(filtered, 90/255);  figure, imshow(bw);   % set regions: 20% each  [rows, cols] = size(bw);  farleft\_region = bw(:, 1:floor(cols/5));  slightleft\_region = bw(:, floor(cols/5)+1:floor(2\*cols/5));  center\_region = bw(:, floor(2\*cols/5)+1:floor(3\*cols/5));  slightright\_region = bw(:, floor(3\*cols/5)+1:floor(4\*cols/5));  farright\_region = bw(:, floor(4\*cols/5)+1:end);    % calculate sum of regions  farleft\_sum = sum(farleft\_region(:));  slightleft\_sum = sum(slightleft\_region(:));  center\_sum = sum(center\_region(:));  slightright\_sum = sum(slightright\_region(:));  farright\_sum = sum(farright\_region(:));   % print totals  fprintf(['\nFar Left Sum: %d, Slight Left Sum: %d, Center Sum: %d, ' ...  'Slight Right Sum: %d, Far Right Sum: %d\n'], farleft\_sum, ...  slightleft\_sum, center\_sum, slightright\_sum, farright\_sum);   % find which is greatest  sums = [farleft\_sum, slightleft\_sum, center\_sum, slightright\_sum, farright\_sum];  max\_sum = max(sums);   % give direction  if max\_sum == farleft\_sum  direction = 'Turn Hard Left';  elseif max\_sum == slightleft\_sum  direction = 'Turn Slight Left';  elseif max\_sum == center\_sum  direction = 'Go Straight';  elseif max\_sum == slightright\_sum  direction = 'Turn Slight Right';  elseif max\_sum == farright\_sum  direction = 'Turn Hard Right';  end   % print direction  fprintf('%s: %s\n', new\_images{i}, direction); end |
| --- |

**Results:**

| **farleft.jpg** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **farleft.jpg: Turn Hard Left** |

| **slightleft.jpg** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **slightleft.jpg: Turn Slight Left** |

| **center.jpg** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **center.jpg: Go Straight** |

| **slightright.jpg** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **slightright.jpg: Turn Slight Right** |

| **farright.jpg** | |
| --- | --- |
| **Original Image** |  |
| **Binary Image (Inverted)** |  |
| **Results** | **farright.jpg: Turn Hard Right** |

**CONCLUSION:**

Overall, I was able to successfully identify the direction the robot should go based on where the line is. The basic process was to load the image, convert to grayscale, filter out the noise, convert to binary, and invert it. I inverted the binary image because of part 1. The left side picture was a little too close to the center, so I had to make the center region smaller. Had I not inverted the image, it would have added the pixels that are not the line (because the line would be black), and throw off the results. After this was done, I was able to add up the pixel values in each region and determine where the line was located and which way the robot should turn. I had to play around with different values for medfilt2() and im2bw(), but I found a good sweet spot that allowed everything to work properly (part 1 and 2 have different values for im2bw()).