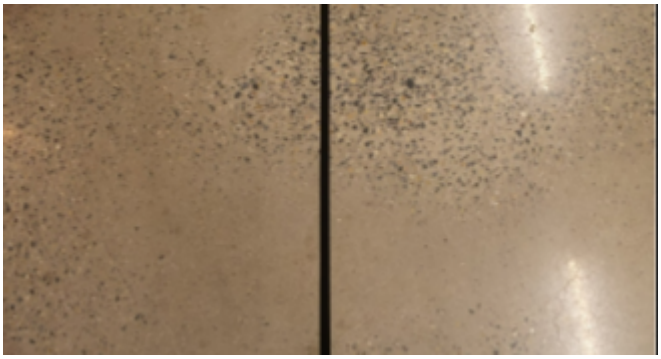
Lab 2

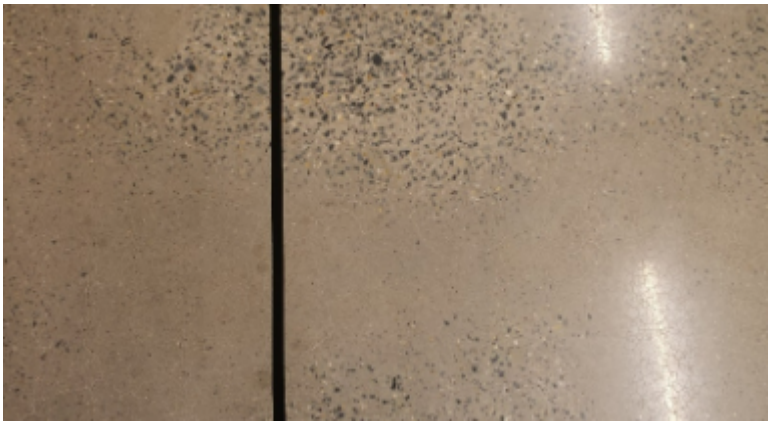
Part 1

Line Following Algorithm Using the following images

center.png



left.png



right.png

Goal

Develop and test a Matlab script to control a robot and ensure that it keeps a black line within the center of its camera image at all times. The script should generate commands for the robot to move based on where the line is on the image

Problem Statement

There are 3 input images provided by the instructor (see next slide). For each image given, the program should display one of 3 possible outputs (“go straight”, “turn right”, “turn left”). Show test cases clearly. There will be 3 test cases (the 3 images given).

Custom Function

I wrote a function to make the process of partitioning the images a bit easier. This function called `generateCommand()` takes in an image location, the number of partitions, and the commands to instruct to the robot. It's important that the commands are entered in correspondence to each partition of the image. For example, if the line is located on the left hand side of the image (partition 0 to n/num_partitions) we would instruct the robot to go right. For a "continue straight" instruction, we would have the

line be in the middle partition. For a "go left command" , we would have the line be in
the right partition.

This function uses a loop to partition the image and then sum the pixel values for each image. The function then outputs the command for the robot

```
% function to generate movement command - given an image - the
function
% divides the image into n-partitions and maps each one to a
value in
% the commands list. By applying a mask over each section and
selecting
% the section with the highest pixel sum, we return the command
that
% corresponds to selecting a given partition
% size(commands) == size(num_partitions)

function [movement_command] = generateCommand(im_loc,
num_partitions, commands)
    % preprocessing
    im = imread(im_loc);

    % binarizing
    im = imbinarize(rgb2gray(im), .2);

    % inverting
    im = ~im;

    % allocate space
    im_partitions = cell(1, num_partitions);
    pixel_sums_per_partition = zeros(1, num_partitions);
    kernel = [-1 2 -1; -1 2 -1; -1 2 -1];

    % make the height and width uniform
    [height, width] = size(im);
```

```
    % partition equally
    partition_width = floor(width / num_partitions);


    % divide each image into even partitions and sum the pixel
values of
    % each image
    for i = 1:num_partitions
        partition_start = (i-1) * partition_width + 1;
        partition_end = i * partition_width;

        % add remaining columns in we reach the last partition
        if i == num_partitions
            partition_end = width;
        end

        % save each partition to arrary
        im_partitions{i} = im(:,partition_start:partition_end);


        % summing pixel vals in partition
        filtered_partition = imfilter(im_partitions{i}, kernel);
        pixel_sums_per_partition(i) =
sum(filtered_partition(:));
    end

    % getting index of greatest pixel sum
    [~, max_par_index] = max(pixel_sums_per_partition); % ~ ->
place holder...we dont care about the actual value just the
index

    % return the movement command thats mapped to this partition
    movement_command = "[COMMAND] GO " +
commands{max_par_index};

end
```

Importing Images and Testing

```
% importing the 3 images
im_locs = ["/home/aaron/Projects/matlab/week_4/center.png";
"/home/aaron/Projects/matlab/week_4/left.png" ;
"/home/aaron/Projects/matlab/week_4/right.png"]

% testing images from lecture slides
num_partitions = 3;
commands = {'Right', 'Straight', 'Left'};

disp('Line in Center:')
move = generateCommand(im_locs{1}, num_partitions, commands);
disp(move)


disp('Line in Left Side:')
move = generateCommand(im_locs{2}, num_partitions, commands);
disp(move)


disp('Line in Right Side:')
move = generateCommand(im_locs{3}, num_partitions, commands);
disp(move)
```

Output

```
im_locs = 3×1 string
"/home/aaron/Projects/matlab/week_4/center…
"/home/aaron/Projects/matlab/week_4/left.png"
"/home/aaron/Projects/matlab/week_4/right.…

Line in Center:

[COMMAND] GO Straight
```

```
Line in Left Side:

[COMMAND] GO Right

Line in Right Side:

[COMMAND] GO Left
```

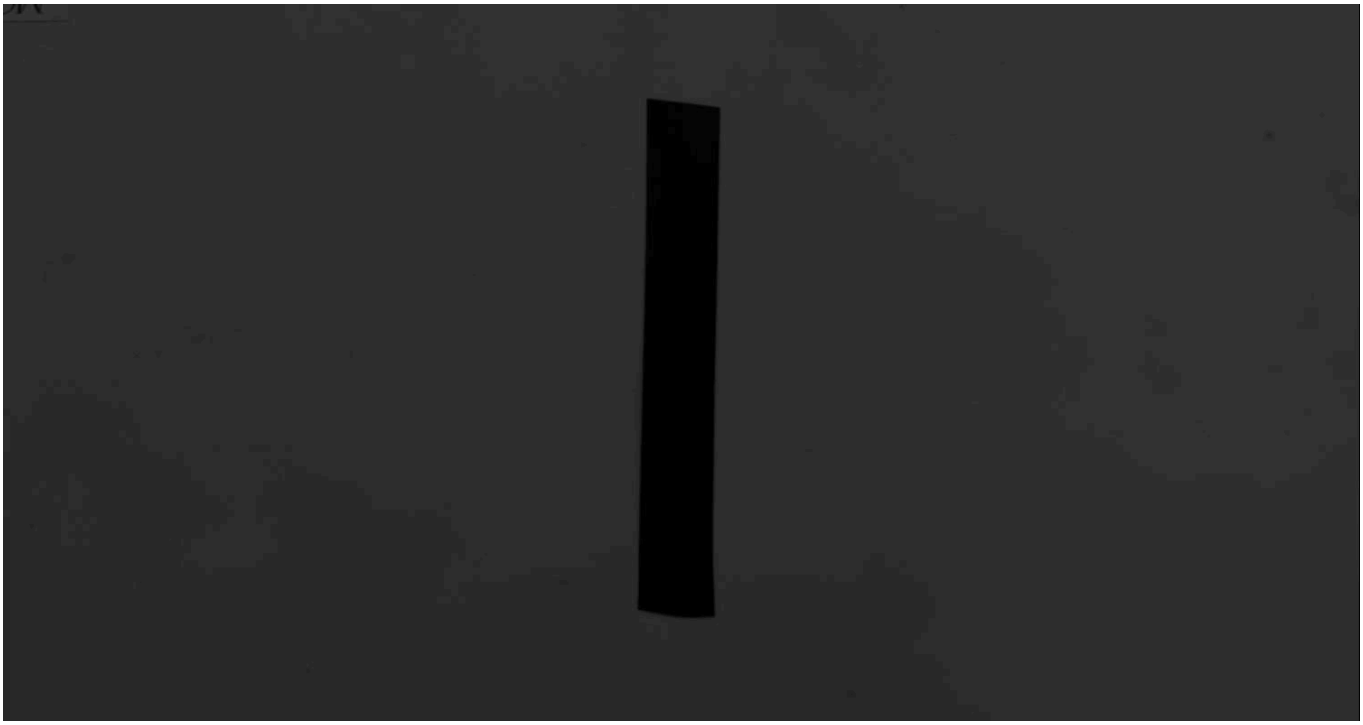
Part 2

Develop an algorithm to control a robot. Take additional pictures with phone so that the
black line (use tape) on the floor is in one of 5 regions (far right, slight right, center, slight left, hard
left). The program should display one of 5 possible outputs (“go straight”, “turn slight right”,
“turn hard right”, “turn slight left”, “turn hard left”). Show test cases clearly. There will be at least 5
test cases. (No manual/interactive adjustments allowed.) You must use the same MATLAB code for
all 5 test cases (5 input images). NOTE: you only need one black line but take 5 separate pics.

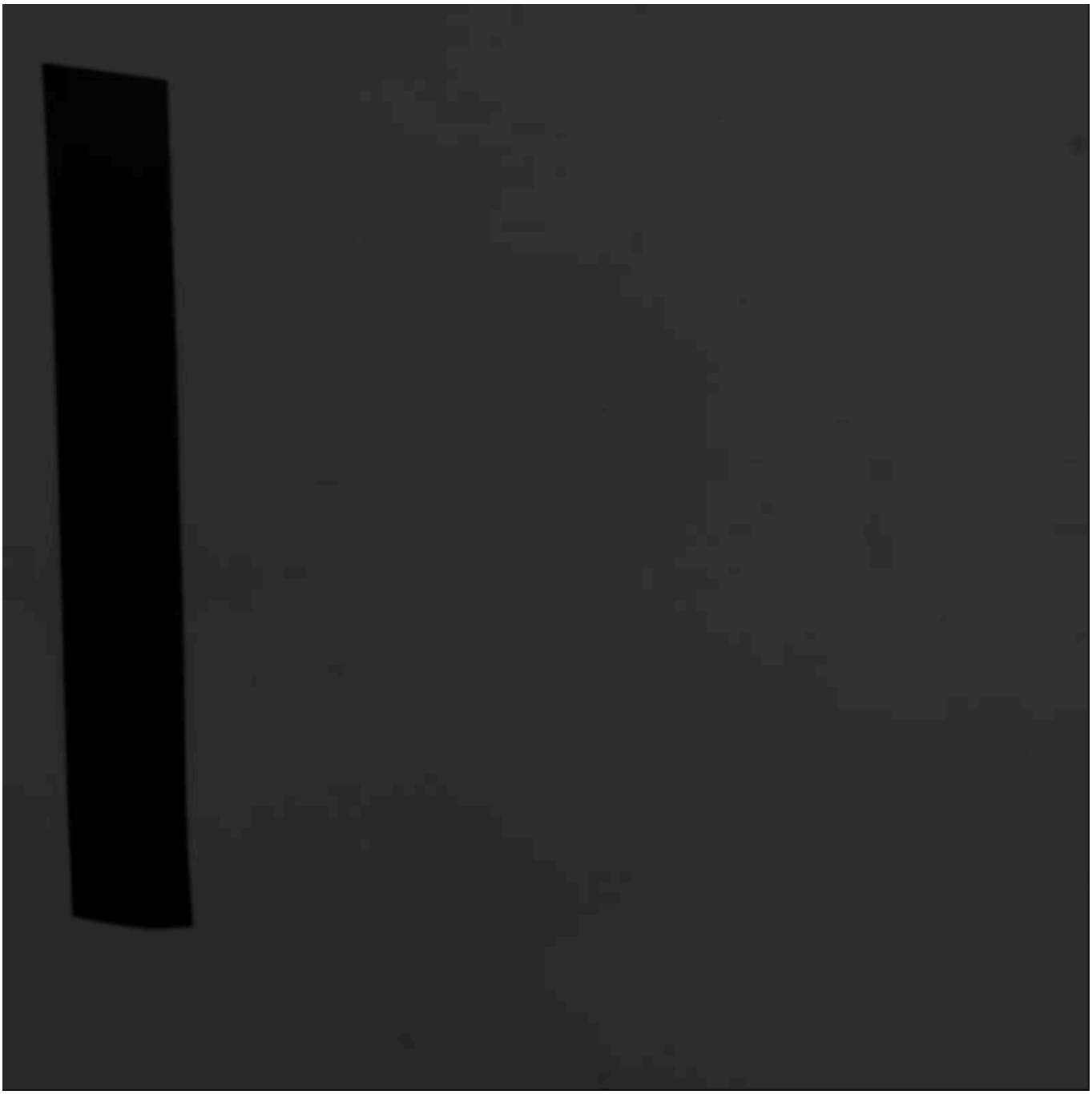
Test Photos

I took the photos on Iphone and converted them from the default .heic format to .jpeg. This for some reason made them into the following color scheme.

center.jpg

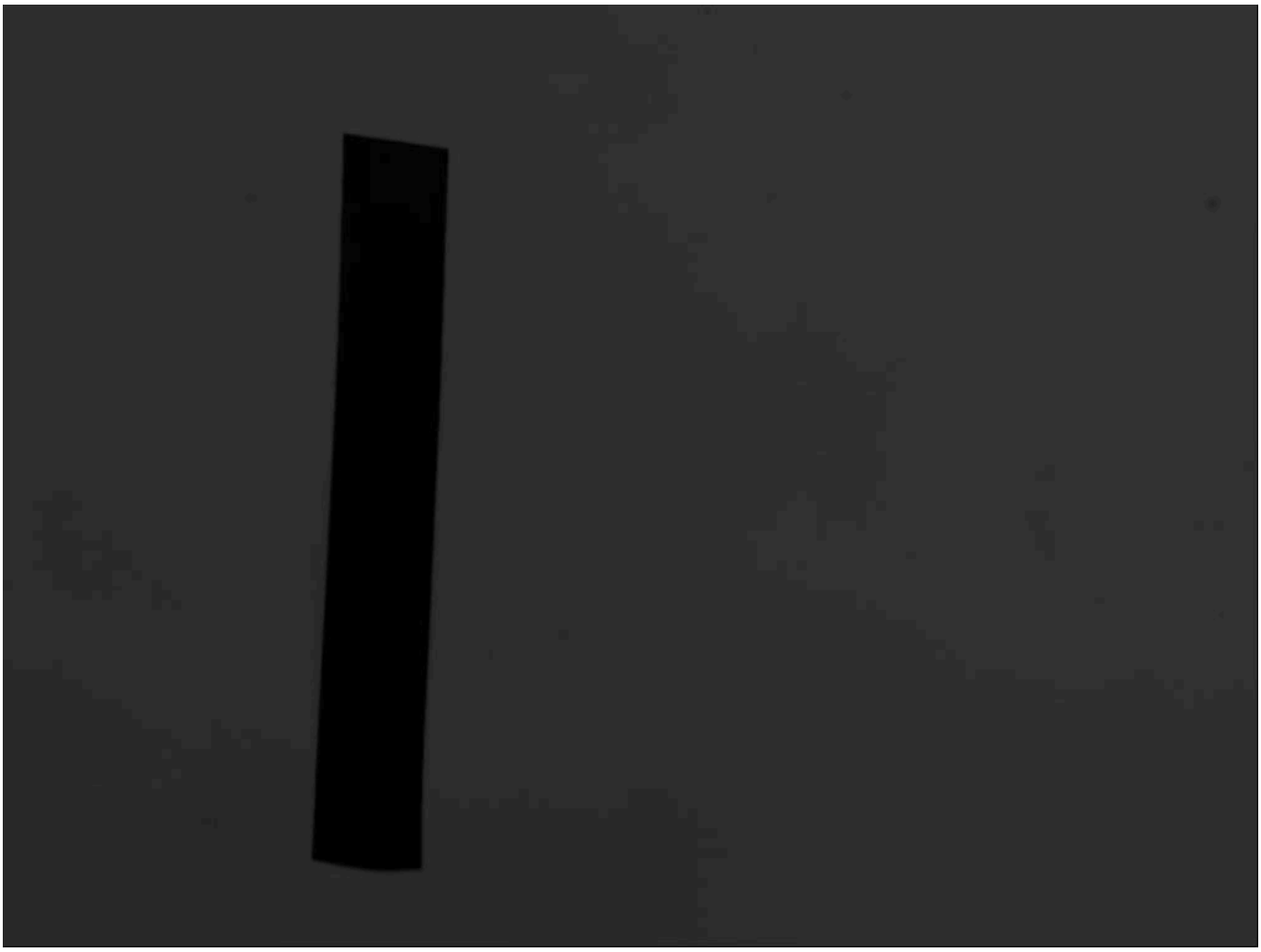


hardleft.jpg



hardright.jpg

left.jpg



right.jpg

Code

The code for this portion is largely similar to the code in part 1 besides a few minor adjustments. In order to compensate for the weird colors, I removed the custom graypoint from the imbinarize function

```
function [movement_command] = generateCommand(im_loc,
num_partitions, commands)
    % preprocessing
    im = imread(im_loc);

    % smoothing histogram of image...possibly

    % binarizing
    im = imbinarize(rgb2gray(im));

    % inverting
    im = ~im;

    % allocate space
    im_partitions = cell(1, num_partitions);
    pixel_sums_per_partition = zeros(1, num_partitions);
    kernel = [-1 2 -1; -1 2 -1; -1 2 -1];

    % make the height and width uniform
    [height, width] = size(im);

    % partition equally
    partition_width = floor(width / num_partitions);


    for i = 1:num_partitions
        partition_start = (i-1) * partition_width + 1;
        partition_end = i * partition_width;

        % add remaining columns in we reach the last partition
        if i == num_partitions
            partition_end = width;
        end
```

```
        % save each partition to arrary
        im_partitions{i} = im(:,partition_start:partition_end);

        % summing pixel vals in partition
        filtered_partition = imfilter(im_partitions{i}, kernel);
        pixel_sums_per_partition(i) =
sum(filtered_partition(:));
    end

    % getting index of greatest pixel sum
    [~, max_par_index] = max(pixel_sums_per_partition); % ~ ->
place holder...we dont care about the actual value just the
index



    % return the movement command thats mapped to this partition
    movement_command = commands{max_par_index};

end
```

Output

commands = 1×5 cell
'turn hard right''turn slight right''go straight''turn slight left''turn hard left'

Line in center:

move = 'go straight'

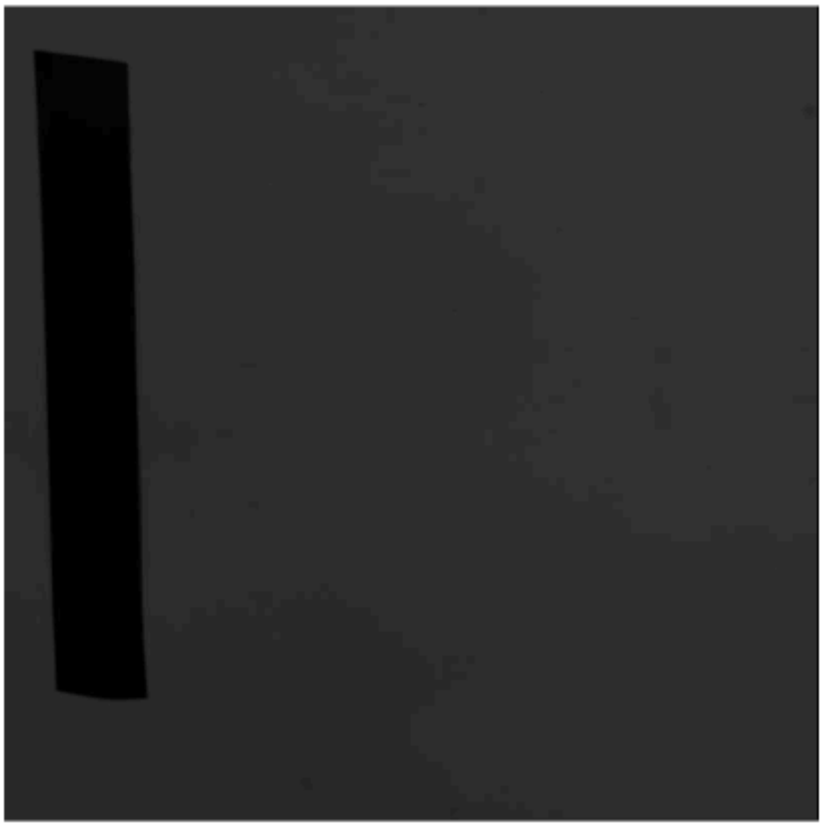
Line in Right Side:

move = 'turn slight left'

Line in Left Side:

move = 'turn slight right'

Line in hard Left Side:



move = 'turn hard right'

Line in hard Right Side:

move = 'turn hard left'

Results

Overall, I feel that the combination of filtering and partitioning does a good job of discerning the position of a vertical line on the screen. However, the fact that we need to manually program a kernel matrix to detect each line type is pretty inefficient so more complicated tasks with may different kinds of lines would require something a bit more versatile like hough transforms.