Task 2 : Car Detection

1. Implement a function that convolves a greyscale image I with an arbitrary two-dimensional kernel H. Make sure your code handles image boundaries in some reasonable way.

Created a mean filter with "**mean_filter[i][j] = 1/9.0;** " this is used to convolve the image.
Used the custom function " **convolve_general_no_border(input_image, mean_filter);** " to convolve images. The **resultant** image is saved as " **convolved_image_no_border.png** ".

Border handling :

3 different bordering techniques are applied.

1. No-border (as discussed above) - **convolved_image_no_border.png**

2. Reflective Bordering – reflections are considered for filling the border missing values - "**convolved_image_reflective.png**"

3. Replication - "**convolved_image_replication.png**"

2. Implement a function that convolves a greyscale image I with a separable kernel H.

The kernel function is being split into horizontal and vertical separable row and column vectors.

```
mean_filter_horizontal[0][i]=1/3.0;
mean_filter_vertical[i][0]=1/1.0;
```

Custom function : " **convolve_separable_reflective(input_image, mean_filter_horizontal, mean_filter_vertical);** " convolves and returns the image.

The resultant image is saved to : **"convolved_image_separable.png"**
The Border case used was the **"Reflection technique"**

3. Use your convolution code to implement an edge detector, using for instance the Sobel operator followed by a threshold on the gradient magnitude.

The sobel operator was applied as follows :

1. calculated the gradient over X direction and then over the Y direction. Did the convolution over the image.

```
 sobel_image_result_x = sobel_gradient_filter(input_image,true); // horizontal = true;
  SDoublePlane sobel_image_result_y = sobel_gradient_filter(input_image,false);
```

```
SImageIO::write_png_file("sobel_x_gradient.png",sobel_image_result_x,sobel_image_result_x,sobel_image_result_x);

SimageIO::write_png_file("sobel_y_gradient.png",sobel_image_result_y,sobel_image_result_y,sobel
```

_image_result_y);
The resultant images are saved as above shown.
2. After above process, calculated the gradient resultant magnitude and the orientation.
3. Did **thinning** using the non-maximum suppression using the gradient resultant orientation calculated above – angle of the resultant gradient.

find_edges(input_image,canny_threshold);

SimageIO::write_png_file("**edge_input_image.png**",edge_input_image,edge_input_image,edge_input_image);

gradient_non_max_suppression.png - image after applying the non- max suppression.

4. **Canny Edge detection :**
applied the canny edge detection with the max and min values for the hysteresis.
Used a recursive approach to find if the pixels above the min threshold are connected to a max value in its recursion.

sobel_and_canny_edges(input_image,canny_threshold,canny_min_threshold);

SImageIO::write_png_file("**canny_edge_input_image.png**",canny_edge_input_image,canny_edge_input_image,canny_edge_input_image);

SDoublePlane canny_image = **canny_edges_detection**(canny_edge_input_image,canny_threshold,canny_min_threshold);

SimageIO::write_png_file("**canny_image.png**",canny_image,canny_image,canny_image);

5. **Template Matching:**
Extracted the 4 templates from the images given. **These 4 templates are to be in the same folder** of the detect.cpp and other files. These images are used for the template matching.

Implemented the Chamfer distance for the template matching.

SDoublePlane c**hamfer_match_score_1** = chamfer_match(input, templates, min, second_min);
SDoublePlane **chamfer_match_score** = non_max_suppression_match_score(chamfer_match_score_1);

Since the chamfer may detect the image multiple times and draws the border of detection more than once, non-max suppression was used to improve the efficiency.

To eliminate multiple windows being detected for the same object, we have detected nearby windows based on Euclidean distance and those within given threshold value are ignored. But, this did not eliminate all the multiple detections for the same object. This can be improved by utilizing non maximum suppression to select best window with the local maximum.

Detection :
the code uses the approaches discussed above to draw a rectangle around the car detected.
Assumptions :
1. Assumed all the cars are aligned in the normal way with no orientation. So the above approach is used. Our goal was to directly detect the cars by their shape using a template match with out relying only on presence of an object in between the two straight lines (parking slots) obtained from a Hough transform.
2. A Hough transform for the detection of car slots and checking for presence of an object would be a different approach that can be explored. Hough transform would help to draw the straight lines for the angled / oriented parking slots too – which can be further checked for presence of an object for car detection.