By: Rahul Jajodia
Date: 02 July 2023

What did you learn after looking at our dataset?

The dataset has a lot of variation in terms of 1- naming convention, 2- image resolution, 3- lighting conditions, 4- field of view, and, 5- objects present in scene

1- For example, c20 uses two different naming conventions to timestamp the images. This can make sorting images according to date/time stamps a little challenging. One could ideally read the image metadata using PIL packages to curtail this, but at the cost of more computation power. In any case, the metadata seems to be missing the ‘Date taken’ entry so this would be impossible to achieve in this particular dataset.

2- c10 has two different resolutions for images, namely 640x480 px and 2688x1520 px, which is again very different from the resolutions used for the other three cameras at 1920x1080 px. C21 also did have a few outliers in this category, which I ignored and deleted from the dataset before running the program. Namely, c21_2021_03_27__12_53_37.png which has a resolution of 10x6 px, c21_2021_04_27__12_44_38.png which has a resolution of 1200x675 px, and c21_2021_04_27__12_04_38.png which has a resolution of 1100x619 px. C21_2021_03_27__12_53_37.png has a very small resolution to get any meaningful results, whereas the other two resolutions of 1200x675 px and 1100x619 px were unique to the dataset. I could have worked around it with image resizing and padding but tuning the parameters to these specific images seemed excessive for this task. There was also a corrupt image c21_2021_03_27__10_36_36.png which was purged.

3- Many images are overexposed due to sunlight (c10-1623913758221.png) and other light sources such as the exit sign (c23-1616704118786.png) or ceiling lights (c20_2021_03_26__05_01_37.png) which make the deduplication problem a challenging task. For the images overexposed due to sunlight, the problem is especially challenging because the light source is non-static. This leads to the shadows and glare being far apart from each other in sequential frames, especially if the images are temporally far apart. For the static light sources, image subtraction should take care of this even if the frames are temporally far apart. The only exceptions are the transition images where the lights are off in one image and on in the other, and vice-versa.

4- The field of view is also very different between camera views. For example, c10 has a much narrower field of view than c20 or c23. This means that objects that occupy a certain image area in c10 could occupy a much smaller area in the images of c20 or c23 due to being much further from the camera. Consequently, the min_contour_area parameter would thus have to be much smaller for c20 and c23 than c10. Having depth information coupled with the camera parameters could help curtail this to some extent.

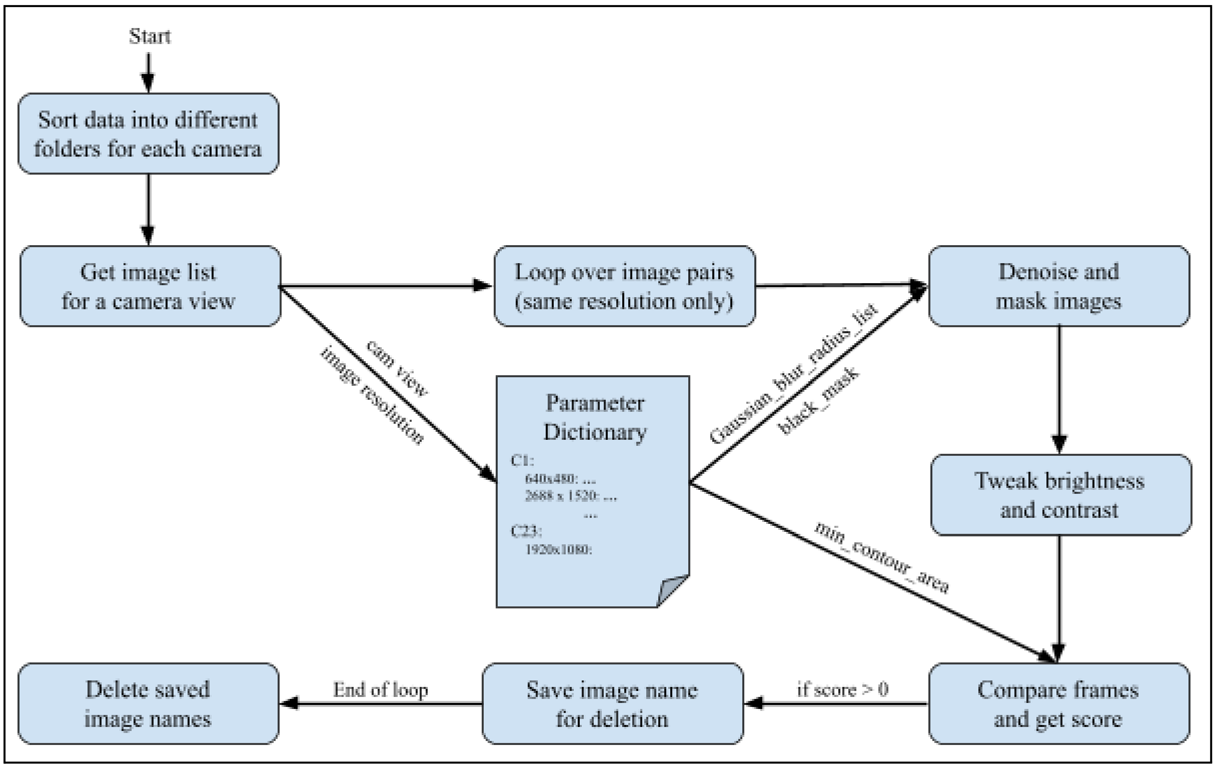
5- The scenes also provide a good reflection of real world scenarios. There are often pedestrians (c20_2021_04_27__12_50_19.png) and objects (c20_2021_04_27__11_18_28.png) present in the scene. Some are closer to the camera, like in c20_2021_04_27__12_33_10.png while some are much further back in the view, such as in c20_2021_04_27__12_21_45.png. There are also cars frequently entering and exiting the parking lot. With one of Kopernikus automotive problem statements being autonomous parking, I believe this dataset provides good insight into the daily facets of a parking lot. It thoroughly familiarises me, as a candidate, to the challenges faced in solving such a complex problem.

6- Lastly, there is prominent distortion in the images, particularly barrel distortion. This can be fixed with camera parameters if one could encode it to the metadata during dataset creation.

Overall, the dataset provides a good sample of a real world scenario with varied scenes, unknown camera parameters, distorted and noisy images and diverse lighting conditions.

How does your program work?

The program follows the pipeline as illustrated in figure 1.



Start
Sort data into different folders for each camera
Get image list for a camera view
Loop over image pairs (same resolution only)
Denoise and mask images
cam view
image resolution
Parameter Dictionary
C1:
640x480: ...
2688 x 1520: ...
...
C23:
1920x1080:
Gaussian_blur_radius_list
black_mask
Tweak brightness and contrast
min_contour_area
Delete saved image names
End of loop
Save image name for deletion
if score > 0
Compare frames and get score

Figure 1: Pipeline for deduplication workflow

1) At start, the data is sorted into different folders for each camera view. This is done purely for organisation and serves no other purpose.
2) Next, we get an image list for each of the camera views. For camera views with multiple resolution images, such as c10, the image list is split into two according to their resolutions. The lists are then sorted by name to arrange them temporally.
3) Image list is then parsed over, and pair of subsequent images are selected
4) These images are converted to grayscale, denoised, and masked as per chosen parameters using the pre-given algorithm
5) The images are further tweaked to lower variance in pixel values such as those in the case of overexposure
6) The images are finally compared using the pre-given algorithm and the score is obtained
7) If the score surpasses a threshold, 0 in our case, the name of the latter image in the pair is saved
8) Once all pairs have been compared, the saved names are deleted, thus achieving deduplication
9) We return to step 2 until all camera views have been parsed

The Parameter Dictionary:

This dictionary serves as a lookup table for the hand-picked parameters, namely *gaussian_blur_radius_list*, *black_mask* and *min_contour_area* used for each resolution in each camera view. The motivation for using different blur radius is due to the different image resolutions across and within views. Different black mask percentages are used across views to block out ceiling fixtures. This is different for each view, thus requiring a different value respectively. Different camera views also have different fields of view as mentioned earlier, and thus require different *min_contour_area* parameters. For example, a non-occluded person in c10 will occupy a much larger contour area than one in c20 if they are at a good distance from the camera. Therefore c20 will use a much lower *min_contour_area* than c10 as ignoring these contours can be disastrous to the use-case of autonomous parking. Another motivation is that an image with an original resolution of 640x480 px will surely need a smaller *min_contour_area* than one with an original resolution of, say, 1920x1080 px for the same view.

What values did you decide to use for input parameters and how did you find these values?

Different parameter values were used for different resolution images and different views. A quick run-down is as follows:

Camera ID	Resolution	*blur_radius*	*black_mask*	*min_contour_area*
c10	640x480 px	[5,5]	(0,13,0,0)	500 px^2
	2688x1520 px	[11,5,5,5]	(0,13,0,0)	1500 px^2
c20	1920x1080 px	[11,5,5,5]	(0,28.5,0,0)	1250 px^2
c21	1920x1080 px	[11,5,5,5]	(0,30.5,0,0)	2500 px^2
c23	1920x1080 px	[11,5,5,5]	(0,35,0,0)	600 px^2

Table 1: Hand-picked parameters for different resolution images in different camera views. *blur_radius* is *gaussian_blur_radius_list*

All the parameters were handpicked after extensive testing for the use-case of autonomous parking. As an extension to the given task, one would want to segment out objects from deduplicated images to allow for object segmentation and recognition. Large blurring kernels, or masking the sides can impede that. The parameters were selected with that in mind.

Motivation:
The motivations for the *gaussian_blur_radius_list* and *black_mask* are almost the same across all scenes.

gaussian_blur_radius_list:
Blurring is generally done to denoise the images. The chosen blurring kernels achieve a nice balance of denoising the image while not blurring the image too much so as to lose valuable segmentation details for thresholding. Ideally, larger images allow for bigger blurring kernels without sacrificing segmentation details.. This is why c10 has a much smaller effective kernel size for Gaussian blurring due to its much smaller resolution. All other resolutions use the same effective kernel size for Gaussian blurring.

black_mask:
The main goal with the masks was to hide the ceiling beams and lights while not hiding the sides. The sides are not masked because it can provide valuable information

such as open parking spots in c20, approaching vehicles/pedestrians in c10 and c21, and a mix of both in c23. This information is important as having a precursor to an approaching object can help avoid collisions. Knowing which parking spots are available are also vital for autonomous driving. As such, the provided scenes did now allow for masks on the sides or at the bottom. A plausible situation for not masking the ceiling would be if it was undergoing maintenance, but we would still be able to see at least a ladder or other equipment in the rest of the view.

min_contour_area:
This was the trickiest parameter to hand-pick. It is very scene-dependent, and thus, is assigned a different value for all scenes and resolutions.
c10 640x480px:

> *min_contour_area* of 500 px^2 allows to filter out most small thresholding artefacts after two iterations of dilation while preserving objects large enough to obstruct autonomous parking. A smaller resolution means we must be able to detect objects that appear small but are actually big enough to obstruct autonomous driving.

c10 2688x1520 px:

> *min_contour_area* of 1500 px^2 is much larger than 640x480px to account for the much larger resolution. I experimented with values of 1500 px^2,1750 px^2 and 2000 px^2 and 1500 px^2 seemed like the better fit. For this particular dataset, a much bigger value could be used since only one image has a car and a person, occupying a contour area much higher than 5000 px^2 while the others just have an empty scene. However, for consistency, 1500 px^2 was ultimately chosen.

c20 1920x1080 px:

> This scene has a much deeper view of a parking lot. Close-by objects appear huge but a car could occupy a contour area even smaller than 2000 px^2 in the distance. To not miss these important cues, a *min_contour_area* of 1250 px^2 is chosen.

c21 1920x1080 px:

> This scene has a very small field of view, both horizontally and vertically. Almost any object that could potentially obstruct autonomous driving would appear significantly larger in this view than in other given cameras. As such, a *min_contour_area* of 2000 px^2 seemed like a fair choice.

c23 1920x1080 px:

> This scene uses the same motivation as c20 and the deeper view of the parking lot motivates a smaller *min_contour_area* of 600 px^2. This is much smaller than c20 for the simple reason that while this view is deep, it is less deeper than c20. Blurring kernels will make it very difficult to threshold smaller objects in c20 that are far away. However this does not apply to c23 as even the furthest

objects would be comparatively closer in c23 than in c20, and thus should not be filtered out.

What would you suggest to implement to improve data collection of unique cases in future?

Using a single camera to capture deep scenes such as c20 and c23 can be detrimental. It would be much better to use more cameras to capture smaller parts of the scene along the length. Also, accounting for camera failures is important. Cameras should be set up such that adjacent cameras can be used when one breaks down, so as to not have blind spots. An illustration is shown in figure 2.

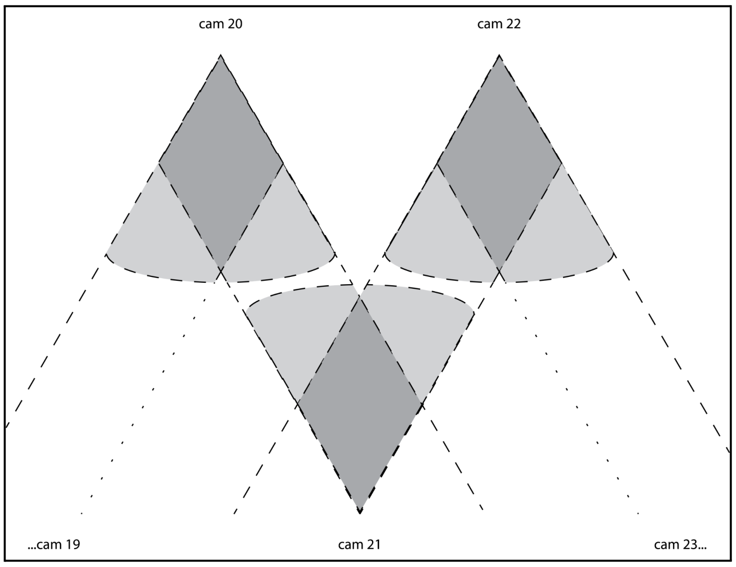


Figure 2: Illustration showing cameras covering each other’s views to compensate if a camera breaks down

One of the main augmentations to improve data collection of unique cases in future would be to include depth information and camera matrix, along with images for background subtraction. While some views did have images with empty scenes which could be treated as background, some did not. Further, undistorting images can also help improve images, although this requires distortion coefficients. A robust way to calculate distortion coefficients could be to stick ArUco or chessboards in the background.

Any other comments about your solution?

I experimented with many algorithms to get a better solution when subtracting one image from another, particularly for the images with overexposed pixels. However the problem seemed too difficult to solve in the given time constraint. I tried adaptive thresholding, otsu's binarization, different combinations of erosion and dilation. Ultimately, adjusting the brightness and contrast of the image to get a more distributed image histogram combined with the pre-given thresholding seemed to give the best results for now. I think having background images for all view cases to background subtract would definitely help, particularly in the scenes with heavy or very low lighting. I would need to test to be sure. I would also like to experiment with gamma correction by using a low gamma value for bright images and high gamma value for darker images. Ideally one should be able to distinguish between such scenes by looking at the image histograms. It should help in achieving a more robust background subtraction and thresholding. Having camera parameters or depth information would also make contour filtering more robust.

Please note that I have not included error checking code in many places and it is also missing a lot of comments. This is purely due to time constraints.

Thank you.