**Adaptive Thresholding**

The first extension I implemented for my extended coin detection pipeline was adaptive thresholding. A simple threshold with a value of 22 was the original threshold method; however, my pipeline was recognizing more than the intended number of coins when dilating and eroding under 5 times. To remedy this issue, I would have to dilate and erode more for my pipeline to recognise the correct number of coins. I thought that applying an adaptive threshold may help in reducing the number of times I would need to dilate and erode the image, while still recognizing the correct number of coins.

I followed the adaptive thresholding method presented in class. First, I created a computeHistogram function which I then used to find the px\_array histogram, using a for loop, I was able to find the intensity total and intensity histogram. I then calculated the initial threshold. A while loop was used to calculate the next iterations of the threshold, ending when the new calculated threshold was the same as the previously calculated threshold. Finally, with the adaptive threshold calculated, I looked through each pixel setting the pixel to 0 if it was under the threshold and to 255 if it was over the threshold.

Using adaptive thresholding resulted in bigger unwanted gaps. However adaptive thresholding also removed the unwanted spots that simple thresholding picked up, this is shown with hard\_case\_1. I would still need to dilate and erode a number of times to get good results.

|  |  |
| --- | --- |
| Simple Thresholding | Adaptive Thresholding |
| Threshold Value: 22 | Threshold Value: 31 |
| Threshold Value: 22 | Threshold Value: 35 |

**Median Filter and Gaussian Filter**

I next implemented a 5x5 borderIgnore median filter and a 5x5 borderIgnore gaussian filter to see how each would compare with the mean filter.

For the median filter, for each pixel I added all pixels within the 5x5 and added to a list which was then sorted, picking the 12th index which is the median pixel. For the Gaussian filter, I used the kernel shown in lectures for the 5x5 gaussian filter, then divided the values by 100.

The median filter produced bigger gaps, erasing the thinner connections. This was better for reducing irrelevant material, working the best for hard cases 1 and 3. The gaussian filter and mean filter were pretty similar in result.

|  |  |  |
| --- | --- | --- |
| X3 Mean Filter | X3 Median Filter | X3 Gaussian Filter |
|  |  |  |

**Laplacian Edge Filtering**

I also implemented the 3x3 Laplacian edge filtering to see how it would differ from the Scharr filter.

Using the kernel presented in the lecture/assignment handout I summed the calculated values.

The Laplacian filter produced a very dark image compared to the Scharr filter. It was very effective at reducing the noise and background information. With the Laplacian filter on hard case 1 and 3 I did not need to dilate/erode as the Laplacian edge filtering and thresholding was enough to register the correct number of coins. However, hard case 2 with Laplacian edge filtering and adaptive thresholding without dilate/erode pipeline, despite ignoring the background correctly, did not recognise the $2 coin. When increasing the amount of dilations/erosions, it would pick up parts of the background or recognise the coin incorrectly. The best result I found was to dilate it 3 times, then ignore the background that was detected as a component through size cutoffs in the connected component function.

|  |  |
| --- | --- |
| Scharr Edge Filtering | Laplacian Edge Filtering |
|  |  |

**Connected Component Size Cutoffs**

While the Laplacian edge filtering, median filter and adaptive thresholding produced close results, sometimes it would falsely trigger background as components. To fix this, I added a size checker to my Connected Component function that would help verify whether it was a coin or a false connected component. I chose the value 10000 as the min and 10000 as the max.

To implement this, I checked for each key in the connected components dictionary in the connected components function, checking that the value for each key was over 10000 and under 100000. I chose these values as a coin component is usually within these bounds. I also changed how I recorded my bounding box from a list to a dictionary of lists. For each key in the dictionary, I added an entry to the bounding box dictionary that would record the bounding box value. If the pixel in the connected components pixel array was in the bounding box dictionary, I would record the value if it was the min x, min y, max x or max y. I then returned the list of values to match the bounding box list format.

**Coin Type Detection**

I also implemented coin type detection by hard coding the range of values that the coin’s bounding box width \* bounding box height would be. I based my bounding boxes on the hard case images as the tutors suggested that coin type detection should be compared within the same folder because the coins have similar ranges in size. I printed this text using axs.text.

**Final Pipeline**

For my final pipeline I went with the Laplacian edge filtering, the median filter, adaptive thresholding, 4 dilations and connected component size cutoffs. The reduction in dilations and erosions also makes the pipeline run faster. Finally, I printed the total number of coins found in the image and the labelled coin types.