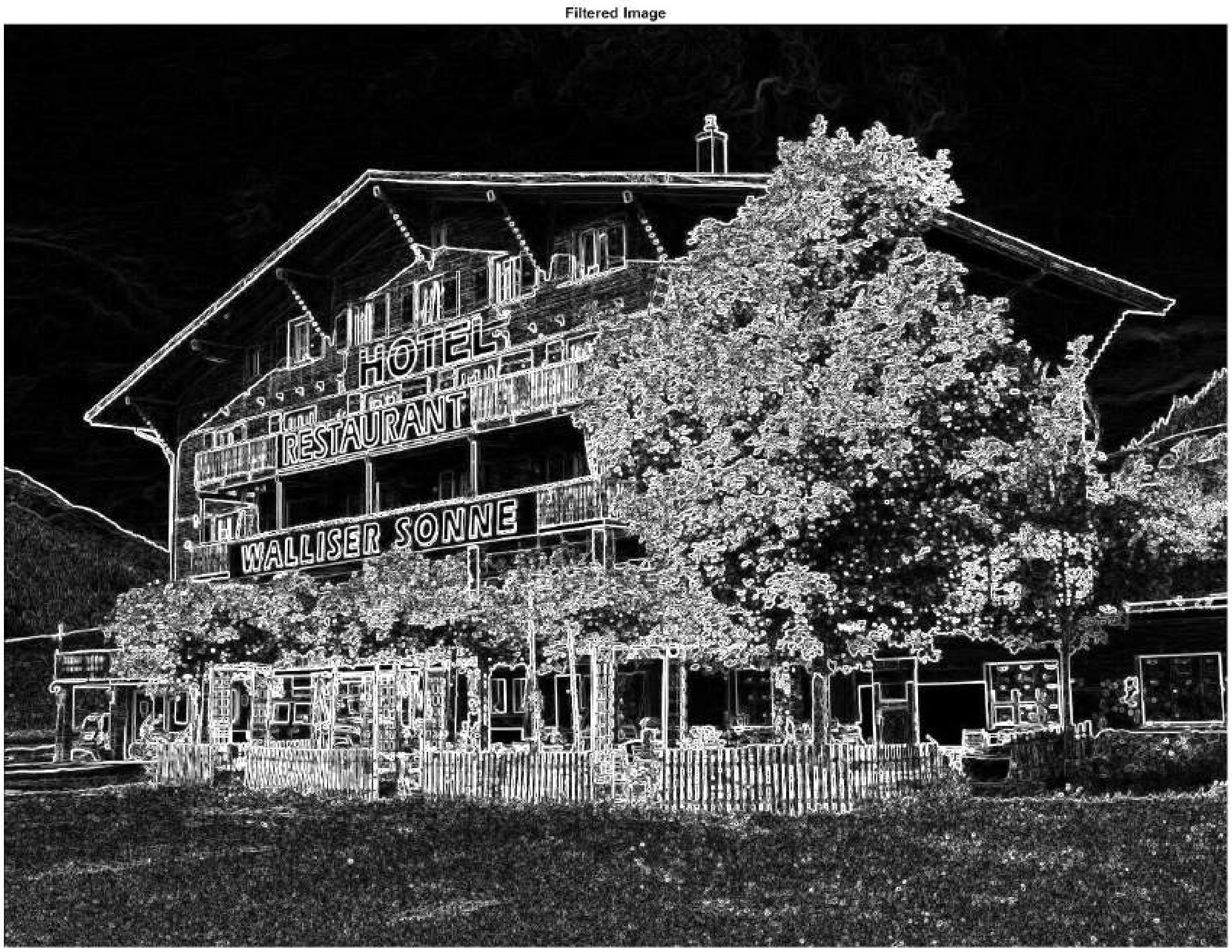
MCEN 5115 Graduate Project: Image Filtering for Mapping

Maxwell Patwardhan - Professor Derek Reamon

December, 2022



1 Background

The field of robotics is inherently a melting-pot of multiple disciplines within engineering. To be a successful roboticist, one must have an firm conceptual grasp on computer science, mechanical design, electronics, and applied mathematics. The challenges that exist within robotics are vast, and while I would like to discuss them all, I am a poor undergraduate with many deadlines and little time. Thus, I will choose to write my report on a small but necessary topic that I find quite interesting: Image processing!

Modern autonomous robots rely heavily on imaging in order to navigate their environments. Imaging technology comes in a variety of styles ranging from Lidar to standard cameras that operate within the visible spectrum. Cameras provide the robot with necessary information for path planning and localization. Yet, there is an inherent problem with imaging technology: it is too information-dense. What does this mean? A typical camera provides a surfeit of data every second. It is not possible for modern day computers to handle such a volume of information at a rate that is useful for real-time autonomous decision making. Thus, the data must be filtered. Processing and filtering image data allows the controller of the robot to parse necessary information in order to make decisions. The speed, accuracy, and efficiency of this filtering is key to the performance of an autonomous agent. For this project I have built a filter that takes an image input and returns the edges of the objects within it, commonly known as a Gaussian Filter with the Sobel Operator.

2 The Importance of Filtering and Edge Detection

Suppose there is a standard webcam on your robot. The camera has a resolution of 1920x1080 pixels, each which measure a red, green, and blue value. The camera operates at 30 frames per second, which means it is taking 30 images every second. It can be calculated that the robot is absorbing:

$$1920 \times 1080 \times 3 \times 30 = 186624000 = 1.87 \times 10^8 \text{ bytes per second}$$

This is an inordinate amount of data for the robot to inspect every second. Thus, the content of the image must be filtered before it reaches the controller. These filters can drastically reduce the amount of data that reaches the processor, all while maintaining pertinent information. The type of filters that I have built are convolution based filters. When two functions are *convolved*, one function is continually shifted and multiplied across the other. Mathematically, this is represented as:

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \tag{1}$$

It is hard to gain intuition by just looking at this integral. It is important to note that $g(x)$ is regarded as the ”filter” of this operation. It is the function g that is slid along the function f. Thus g can be be constructed to eliminate and keep certain data within f. Also, images aren't infinite, and thus a discretized version of the convolution function must be used in place of the continuous model. Bounding and discretizing any continuum model will inherently introduce error, which will be discussed in later sections. The discrete representation of the convolution function looks like:

$$f(x) * g(x) = \sum_{i=-n}^{n} f[i]g[x-i], \text{ where } n \text{ is the size of } f(x) \tag{2}$$

Now, as previously mentioned, images are also two dimensional, thus the sum can be easily expanded to support another variable. Not only this, due to the fact that convolution is a linear operator, the role of f and g within the convolution function can be swapped. This allows the filter g to be preserved, and only f to be augmented within the sum. Implementing these new facts, we arrive at:

$$\boxed{f[x,y] * g[x,y] = \sum_{i=-n}^{n}\sum_{j=-m}^{m} f[x-i,y-j]g[i,j], \text{ where } n,\, m \text{ is the size of } f[x,y]} \tag{3}$$

3 Convolution Based Filtering

Now that the baseline function for filtering has been derived, we can construct our filters themselves. The image will be passed through two standard filters: a Gaussian Denoising filter, and Sobel Kernel Filter. The Gaussian Filter will essentially perform a weighted average of a pixels values relative to it's neighbors. The Gaussian distribution takes many forms, but it is most useful to us in matrix representation:

$$g[x,y] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4}$$

The average weighting (or σ) value can be augmented in order to determine how much the values of the neighboring pixels influence the augmented pixel. This is a commonly used filter with a σ value of 2. This operation will drastically reduce noise associate with an image, as pictured below:

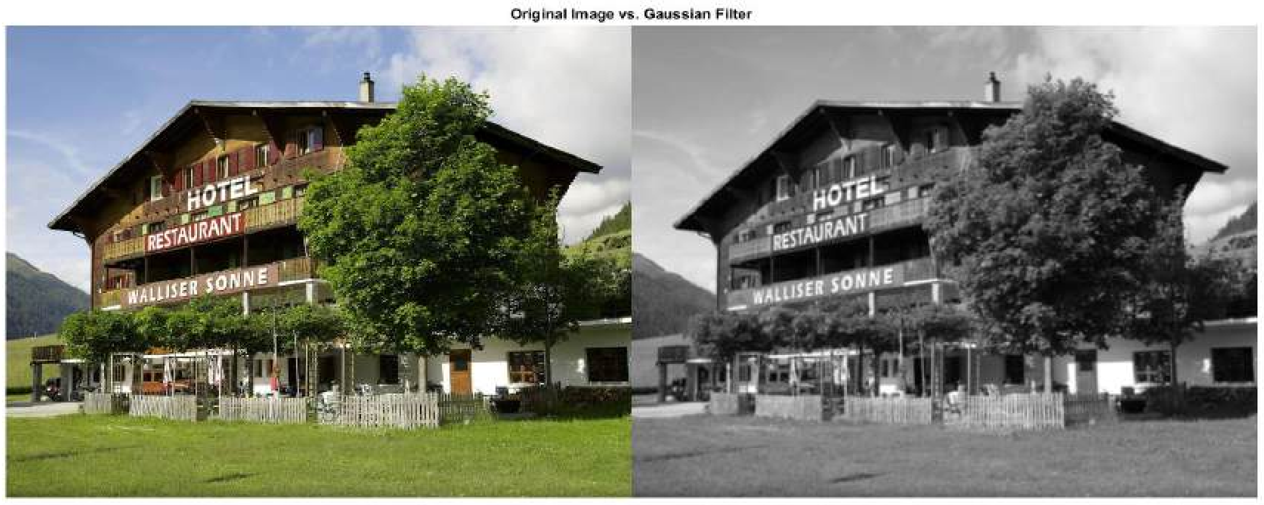


Figure 1: Gaussian Denoising Filter

Each of these images will be put in full resolution in the appendix. The image in Figure 1 was converted to black and white using an average of each RGB pixel value to determine the overall magnitude of brightness, and then was convoluted with function g across each pixel to create the denoised image.

4 The Sobel Operator on Images

While the averaging performed in the previous section has not drastically reduced the size of our image, we can now exploit another filter to discard unimportant information. Take the Gaussian Filtered image

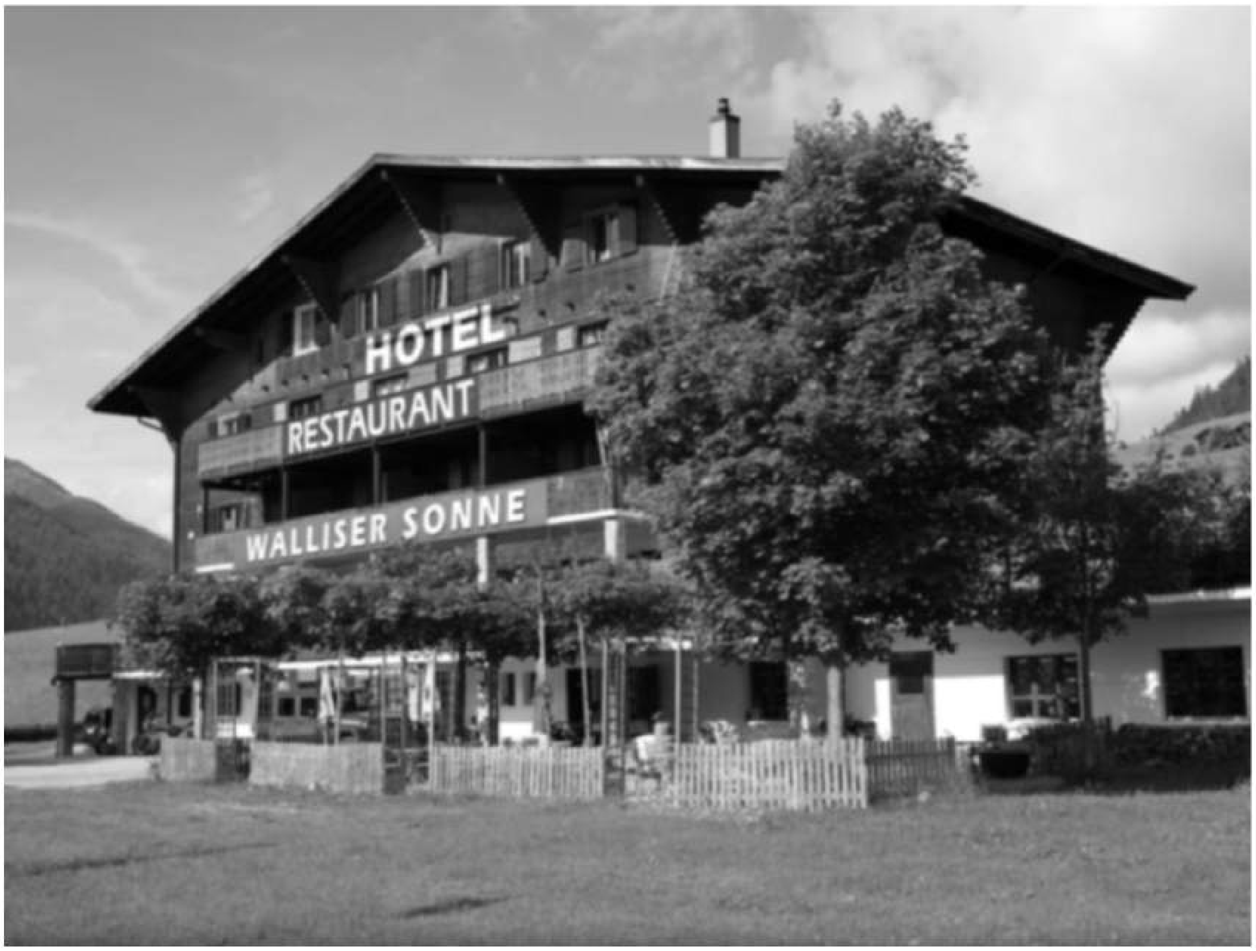


Figure 2: Gaussian Denoised Image

Next, we will convolute this image with the Sobel Filter, in order to reduce the image to it's edges and pertinent information. The Sobel filter consists of two basis kernels s_x and s_y, each of which are convolved with our image, f, and then summed in order to generate the filtered image. The kernels are essential taking the gradient at each point of the matrix, and thus each kernel operates in only one direction. To retrieve the final construction, the magnitude of the gradient at each pixel must be taken. The kernels take the form:

$$s_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad s_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{5}$$

Now, we can convolve our image f with the kernels and compute the magnitude of the directional components of the gradient.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * f[x,y], \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * f[x,y] \tag{6}$$

Lastly, with the uni-directional gradient values calculated, the magnitude can be calculated

$$G[a,b] = \sqrt{G_x[a,b]^2 + G_y[a,b]^2} \quad \forall \left|a\right|, \left|b\right| \leq m,n \tag{7}$$

Enough derivation! Let us look at the effect this has on images. Recall Figure 2, after applying a pass of the Sobel filter, we arrive at this image:

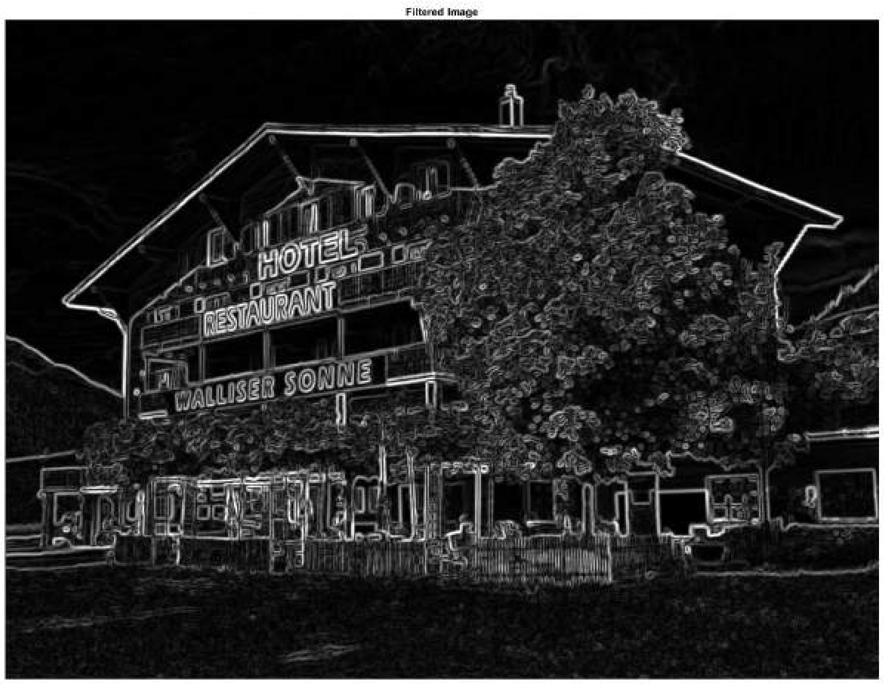


Figure 3: Gaussian, Sobel Filtered Image

Lastly, a tolerance value can be defined in order to remove some of the less-important edges within the image, thus reducing the data even more:

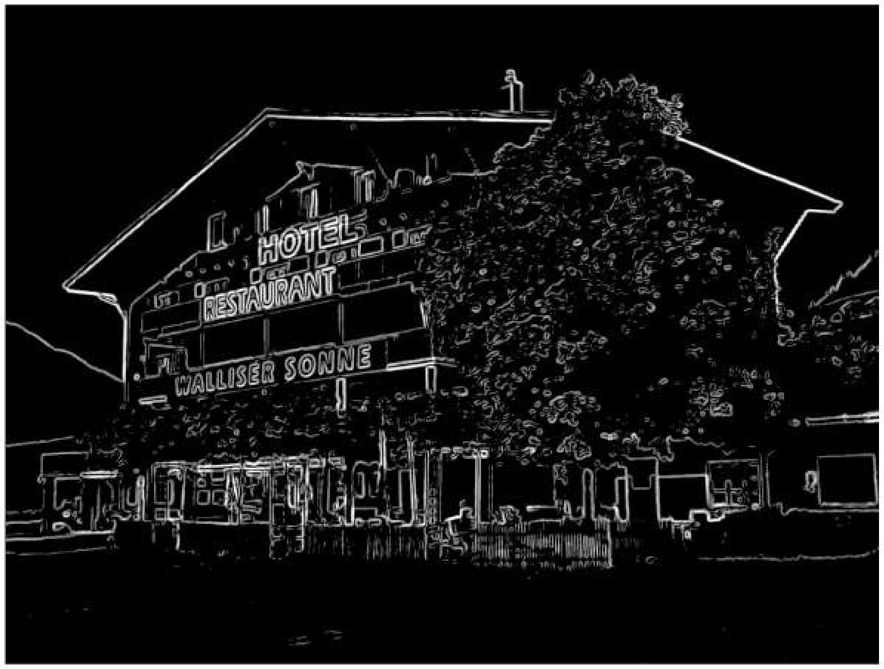


Figure 4: Final Toleranced Edge Image

This version of the same image is 4 orders of magnitude smaller than the original (already compressed) image. With a few simple linear algebra calculations, we have drastically reduced the size of the image, while retaining information relevant to path planning.

It is important to note the importance of the Gaussian filter in this process. Without it, the filtered image would lack much of the necessary noise reduction. This effect can be seen below:

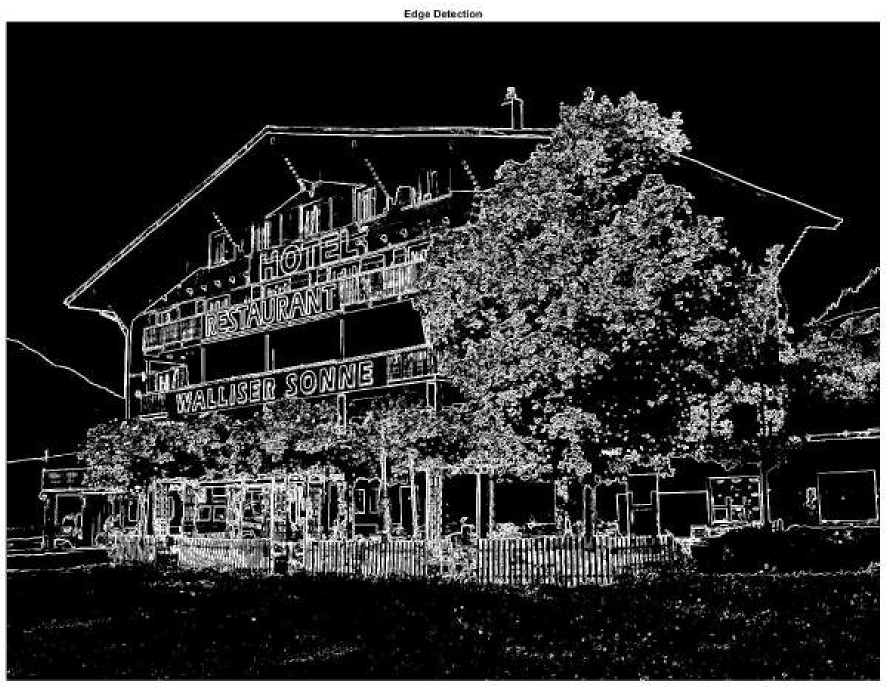


Figure 5: Toleranced Edge, without Gaussian Filtering

The image retains much more useless data without the application of the Gaussian filter, as can be seen when compared to Figure 4.

5 Analysis and Conclusion

In a short set of steps, we have managed to perform a series of calculations that actually make localization for a robot feasible. The figure below compares the frequency and intensity of pixel values ranging from 0 to 255 on both the original image and edge-detected image:

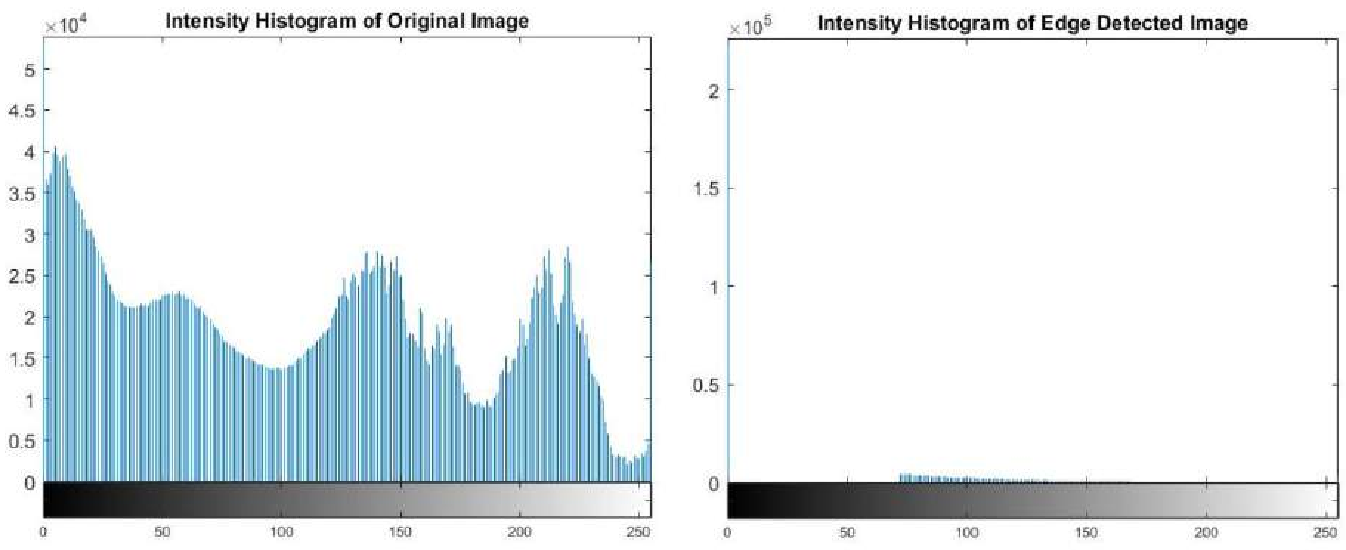


Figure 6: Comparison of Intensity Histograms

The processed image is a handful of kilobytes, compared to the megabytes of the original image. Its pixel-matrix representation is full of zeros, which makes computational operations necessary in localization far less expensive than they otherwise would be. Though, these data filtering and image processing techniques are not at the current forefront of the field, but they do a good job in representing how powerful and necessary image processing is. Without quick access to good sensor information, a robot cannot make good decisions, and a robot that cannot make good decisions is not a very good robot in the first place.

6 Appendix