LineBooth Printer

By:

Chris Kellendonk (4810800)

Jakub Subczynski (4706867)

Table of Contents

Summary 3

Detailed Description 3

Overview 4

How-to Guide 4

Remaining Problems/Quirks 5

Architecture 6

User Interface 6

Image Effects 7

Floyd-Steinberg Dither 7

Otsu Binarization 7

Winnemoller Binarization 8

Image Extraction 8

LineBooth-Printer Communication 9

Software 10

Architecture 10

Optimizing Printing 10

Hardware 11

Overview 11

Spring-augmented Pen 11

Base Support 11

Memory 12

Reduced Java API 12

Section 1: Problem Description

# Summary

The situation that we are addressing is one where an individual may want to be able to easily print an artistically rendered image of them that was taken via webcam.

# Detailed Description

The individual may want to apply different effects onto the image. As well, they should be given the option to extract themselves from the image and apply a fun background. If they like the output, they can then proceed to print it. All of this should be available in one software/hardware package.

Section 2: LineBooth Solution

# Overview

The project that we propose to meet this case is the LineBooth Printer. It consists of the LineBooth software and the LineBooth Printer. The software captures an image, adds a filter to it, and then lets the user add a background image to the extracted foreground of the original. This is then sent to the LineBooth Printer. The printer then receives the image and prints it out as a large grid of dots.

# How-to Guide

Setting everything up is fairly straightforward. The process to do so is as follows:

1. Turn the printer on and run the default program. Note: in case the default program is changed, the program to run is packaged as Printer.nxj. Now that the printer is on and waiting for print jobs, we turn to the software.
2. Run the LineBoth software. The main UI window will appear.
3. Position the camera in a static position to prepare it for calibration.
4. Now, step out of the image so that just the background is in the camera’s sight, and click to calibrate.
5. Step back into the camera’s sight.
6. Click on the Filter dropdown in order to change effect that is applied on your image.
7. Click on the Background dropdown to select an optional background that will be added. Selecting “None” will keep the original background (as seen in the camera’s preview) and selecting blank will set a completely white background.
8. If you would like to see the image that is generated for printing, you can click Preview.
9. Click Print in order to send the image to the printer.
10. Grab a cup of coffee.

# Remaining Problems/Quirks

The time it takes to extract the foreground from the background is large enough as to not make it feasible to present the extracted output in real-time. This is the case even with the optimizations that were made on the custom algorithm. For this reason, the image with a swapped background can only be seen when clicking the Preview button. In order for this application to be sufficiently large for two people, all of the image-processing algorithms were created from scratch – including the foreground extraction. Using openCV may have made the real-time preview more feasible.

Using a writing utensil in order to create the printed image leads to a couple of small issues. One such issue is that the marker tip dries up. If it is left without printing for a period of time, it may need to be activated again by wetting it. The second issue is that the tip of the marker we are using becomes slightly bent due to it constantly striking the paper. That said, it did not seem to produce any very noticeable issues or dot-inconsistencies.

There are problems related to the NXT that had to be worked-around. For these, please see *Section 5: NXT Limitations*.

Section 3: LineBooth Software

# Architecture

The LineBooth application is based around three major components. The webcam receiver to obtain images, the extraction algorithm to remove the person from the background for overlay, and image effects/binarization algorithms to convert the image into black and white for printing.

The binarized image is then packed into a byte array so that each bit of every byte represents a weather a pixel should be drawn or not. The width and height are sent along with the “Bit Packed” byte array to the NXT Printer. Using this information the printer can then unpack the image and print it accordingly. Using this technique the data being sent to the printer will be both maximized in terms of memory usage and speed of transfer (both of which are a concern on the NXT device).

# User Interface

The user interface is very simple. You can select three different sizes of images to print (25x25, 50x50, or 100x100). Images above those sizes are impractical to print due to the time needed to complete the job. There are also “Filter” and “Background” drop downs to augment the image obtained from the webcam. In order to use the background drop down a reference image must first be obtained from the webcam. This is an image without the subject in the frame. Once the user clicks the “Get Background” button the “Background” drop down becomes available. If the user selects a background the user’s face and body will be extracted from the foreground and merged with the selected background. Once the user has setup their preferences they can click the “Print” button to send it to the NXT Printer.

The interface also contains two preview images: the left preview image is the current webcam image with the effects added to the image, and the right preview is the bit packed image exactly as it will be printed by the printer after it has been binarized and packed.

# Image Effects

There are a variety of image effects that can be applied to the image before final output. Each of these effects produces an augmented image that can then be binarized for output to the printer.

### Floyd-Steinberg Dither

The Floyd-Steinberg dithering algorithm produces a very accurate resulting binarized image that retains that shadow detail present in the original grayscale image. It works very well with the printer to produce an image on paper that retains the original quality and shadow detail. It is also a very quick algorithm for our purposes and can easily process an image in real time.

The algorithm works by quantizing each pixel to either black or white and then spreading out the error obtained during this process across the neighboring pixels (see the matrix below). This is known as Error Diffusion. This ensures that the overall detail of an image is not lost during binarization. Because the error is spread out over neighboring pixels it keeps the dark and light areas of the image intact.

|  |  |  |
| --- | --- | --- |
| … | \* | 7/16 |
| 3/16 | 5/16 | 1/16 |

### *The \* represents the current pixel that is being quantized.*

### Otsu Binarization

Otsu binarization is a fast thresholding algorithm that works on a grayscale image to separate the foreground from the background. It first develops a histogram of all the gray pixels in the image to determine their frequency of use. The algorithm assumes that there is a distinction between the foreground and background threshold levels in the image. Thus it finds the midpoint between the two areas of high frequency gray pixel usage. This midpoint determines the threshold; anything above the threshold value will be turned into one pixel color and everything below it will be turned into another. The result is a binarized image with a prominent foreground.

### Winnemoller Binarization

Winnemoller binarization is a complex algorithm that creates a stylized binary image from a grayscale image. It does this using a difference of Gaussians to make the edges more prominent and smooth. The resulting image is then sharpened and thresholded on a tan curve to create a stylized binary image that appears smoother and more artistic then a regular difference of Gaussians would produce on its own.

This filter takes significantly more computational power than the previous filters but produces a nice looking black-and-white image that is appealing to look at. It can process small images in real time but larger images will stutter when trying to process at over 30 frames per second.

# Image Extraction

This is the process by which the background of the original image is removed from the foreground. In order to do this, there are three main algorithms that act upon the original and calibration images.

The first algorithm does the main foreground-background separation. It compares pixels in like-position in the background and foreground images based on their hue, saturation and brightness. If they are the same within a hardcoded threshold, then it treats it as the background and makes the pixel transparent in the resultant image.

The purpose of the second and third algorithms is to clean the up the result of the first algorithm. This is done by first removing any clusters of opaque pixels that are smaller than a certain threshold and then adding back clusters of transparent pixels smaller than a certain threshold. For example, if a cluster of 2 opaque pixels is found, then they will be made transparent, as they are not part of the “main” foreground object. Inversely, if a small cluster of 5 pixels on a persons face is transparent, they will be added back as it is assumed that a small group of transparent pixels surrounded by opaque pixels are part of the foreground image. Using a hash table to store pixels that had been already considered optimized both of these algorithms. This way, large clusters would not be computed each time a different pixel in the cluster was started with.

# LineBooth-Printer Communication

Communication between the PC and NXT is set up solely to transfer the image. There are several steps involved with sending the image to the NXT due to the limited memory and Java API that it has. Both of these limitations are documented in *Section 5: NXT Limitations*.

Both ends of the communication use a PrintJob object in order to store the image and associated printing information. Communication is done purely through reading and writing bytes.

The image is stored in a BitPackedImage object that stores the image in byte array as well the height and width of the image. Each bit in the byte array represents one dot in the output. This led to a very compact/efficient way to store the image. This byte-array-image is sent to the printer by writing the byte array to the output stream. In order to be able to recreate the image from the byte array on the other the NXT, we also first pass the aforementioned height and width of the image.

Section 4: The Printer

# Software

### Architecture

The software for the printer consists of two main components – the PrinterController thread and the Printer thread. The printer controller is the component that maintains the connection to the PC software. It accepts bytes (i.e. images) from the PC and then adds it to a PrintJob queue. This queue is shared with the Printer thread. The Printer thread accesses this queue to get any images to be printed. This is the component directly controlling the hardware to service print requests.

### Optimizing Printing

Taking the time to optimize the physical printer by adjusting the commands that were sent to it were vital when trying to print large images. This is so as printing an image that is 160 dots across and 144 dots down could take 2+ hours to complete! That is, after all, over 23,000 dots to print! Also, since the hardware is where the bottleneck is when printing, it was especially important to focus on it.

The most valuable optimization was to skip groups of whitespace all at once. For example, if there are two black dots, separated by three spaces, the printer makes a dot, moves 3x the normal distance, and then makes another dot instead of making a dot, moving right, right, right and then making another dot. Joining these separate movements saves a great deal of time. An additional check is also made to see if a line being printed ends with one or more blank spaces. If so, it does not bother moving to the end of the line (since it won’t print anything there anyway) and returns to start the next line.

Another such optimization was to have the pen hover above the paper at all times so that it did not need to travel a far distance to make a dot on the page and then to retract. This was perfected simple using trial and error.

# Hardware

### Overview

It is quite remarkable that a printer can be made using the Lego Mindstorm kit. That said, it also leads to inherent issues that need to be corrected in some way.

### Spring-augmented Pen

When the paper is being fed through the printer, it never lays completely flat on the printing surface. The amount it bows out depends on how far the paper has been fed, how straight it is and more. This leads to the pen making lines instead of only a dot while it prints.

In order to make sure that the paper is flat against the printing surface when the dot is being made, a spring is attached around the pen tip that just sticks past the end of the tip. This way, the spring will push the paper flat against the surface and when more pressure is applied, the tip of the marker will then make a perfect dot (and not a line). The spring that is used from this is a shortened spring from a standard, clickable pen.

### Base Support

There are two wood pieces fixed to the base of the printer. Note that the Minstorm pieces have **not** been harmed in any way in order to fix them to the structure. The reason for these pieces is to reinforce the printer to make it less flexible.

When the pen motor pushes the pen into the paper, the guides the printer head moves along bend upward. Depending on the location that is being printed, the guides bend more or less. This leads to inconsistencies with dot spacing, size and visibility. The way to dampen the amount of bend was to finish the rectangular printer structure along the bottom. Without these pieces, the bottom legs would swing out to allow the guides to bend more. Now, the printer legs hold their position which translates to less bending and more dot-consistency when printing.

Section 5: NXT Limitations

# Memory

The small amount of memory that the NXT has is certainly a limitation when trying to print large images. Due to this, the image that is generated is converted into a byte array – where each bit in the array represents one pixel. “1” represents a dot and a “0” represents a space. Doing so allows us to store images 8x as large. When we had initially tried to store the image in a 2D byte array, we were, of course, receiving OutOfMemory exceptions. When we are printing the image, the NXT gets each pixel one bit at a time in order to keep the memory low.

# Reduced Java API

The Java API which is built into the legos API is a severely stripped down version of Oracle’s API. Due to this, many features/classes could not be used. These include: serialization and ObjectOutputStream and LinkedBlockingQueue.

Serialization and the ObjectOutputStream class were initially setup in order to easily send PrintJob objects from the PC to the NXT. This created a very clean form of communication. Unfortunately, the legos API only includes a very basic OutputStream that reads and writes bytes. This meant that there was no easy way to convert the PrintJob objects into byte streams.

The Java LinkedBlockingQueue class was initially chosen to act as the printer queue to store all of the PrintJobs on the NXT. This class automatically deals with concurrency as well. This was valued so that the network listener can add a new PrintJob while the printer thread current read the next PrintJob in the queue. As the legos API does not have this queue implementation, we had to make sure that there were no concurrency issues manually.

Section 6: Resources

Klette, R. (2014). *Concise Computer Vision: An Introduction into Theory and Algorithms.*. England: Springer London Ltd.

Webcam Capture in Java - SarXos. (n.d.). . Retrieved May, 2014, from http://webcam-capture.sarxos.pl/

* API used to access the MacBook iSight cameras using Java

leJOS NXT API. (n.d.). *leJOS NXT API documentation*. Retrieved May, 2014, from http://www.lejos.org/p\_technologies/nxt/nxj/api/

Floyd-Steinberg Dithering. (n.d.). *Floyd-Steinberg Dithering*. Retrieved May, 2014, from http://www.visgraf.impa.br/Courses/ip00/proj/Dithering1/floyd\_steinberg\_dithering.html