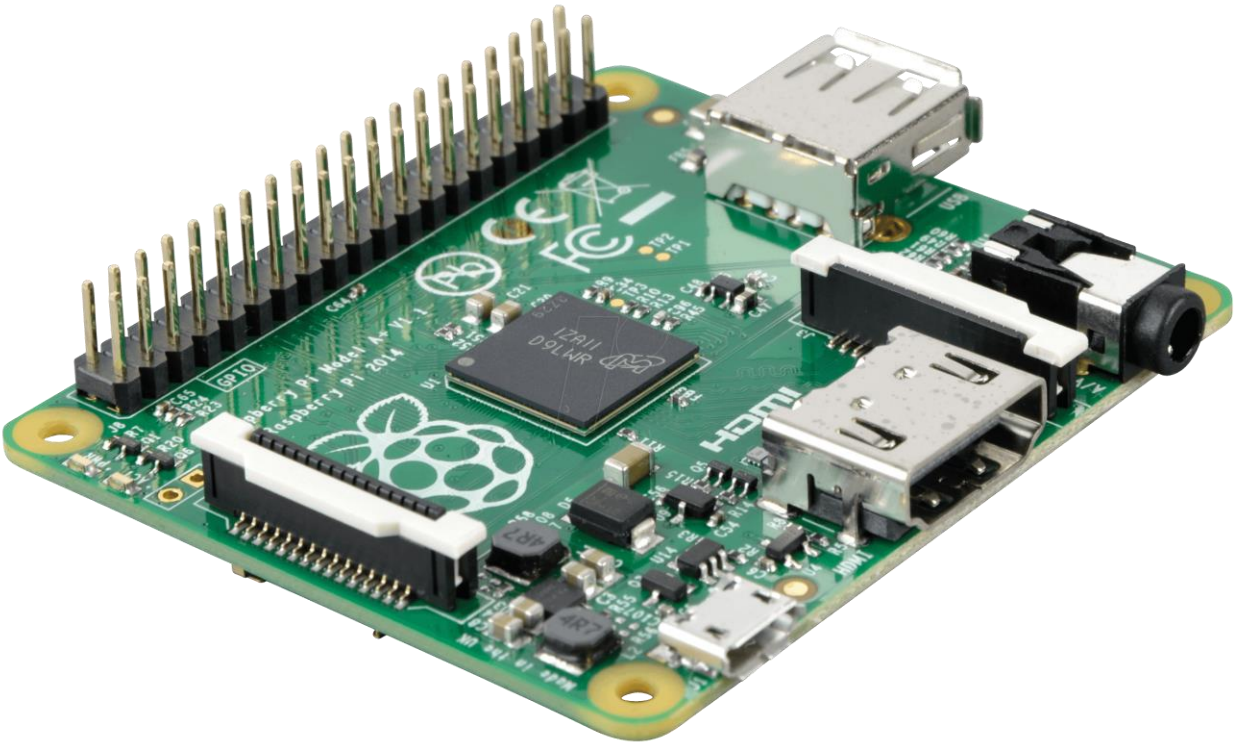
View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DigitalCommons@CalPoly

Cal Poly Computer Engineering Senior Project

Pet Food Monitor Using Raspberry Pi



Advisor:

Professor Andrew Danowitz

By:

Kimberly Agüero

June 10, 2016

Table of Contents

Introduction

Project Overview

For pet owners, monitoring their food intake is one of the most important aspects of ensuring a healthy and happy pet. Overeating has the potential of negatively affecting a pet's life, while a pet refusing to eat could be a sign of something more serious that may require a visit to the vet. Unfortunately, not all pet owners are able to constantly watch over their pet's food bowl. With the Pet Food Monitor, however, they can make sure that their beloved pets are getting the right amount of nutrition they need.

Currently, the only similar technology out there are automatic pet food dispensers. These machines may be convenient for establishing a food schedule, but they cannot accurately monitor a pet's eating habits. By using the Pet Food Monitor, one would be able to keep a record of how much a pet eats and when, conveniently through the internet. This kind of information can be vital in monitoring a pet's overall health and state of mind.

Project Outcome and Deliverables

The targeted outcome for this project is to build a system that can monitor how much food is inside a pet food bowl in real-time. The system will be interfaced with a camera that will be aimed at the bowl at all times, so that it can take a picture of it and use it for analysis. After the picture is taken, the system will run an image processing algorithm using OpenCV to determine

how much food, if any, is in the bowl at any given time. In the event that there is little to no food in the bowl, the system will notify the user and make a record of when the food ran out.

Engineering Specifications

Bill of Materials

Item Description	Qty	Cost
Raspberry Pi Model A+	1	$20.00
Wi-Pi Wireless Adapter	1	$9.69
Raspberry Pi Camera Module	1	$25.00
MicroSD Memory Card 32GB	1	$11.00
Subtotal:		$65.69
Total with Tax:		$69.80

Table 1: *Bill of Materials*

I decided to use a Raspberry Pi as the center computer of this system because of its low cost, compactness, and low power usage. Even with its minimalist design it still has the capacity to carry out image processing algorithms and somewhat heavy computations. I needed to purchase a separate memory card that would hold the operating system image that would boot the system. Since this model of the Raspberry Pi did not come with networking capabilities, I also had to purchase a Wi-Fi adaptor, sold under the name “Wi-Pi.” Another convenience of using a Raspberry Pi was being able to use the camera module that the company also provides. Integration of the camera and Wi-Fi adaptor to the system was quick and simple.

Target User

The main persona that is the target of this project is the everyday hard worker who loves her pet but is unable to track his eating habits because of a demanding work schedule. Before the Pet Food Monitor, the only time she could check the food bowl was in the morning before leaving work and later that night when she finally returned. Now the monitor will automatically notify her if the bowl is every empty and using that, she can keep track of when her pet eats.

Background

This system uses the camera to systematically take a picture of the food bowl and use image processing to determine how much food is present inside. The image processing was all done in C++ using the extensive library of OpenCV.

The image processing could be broken down into a few steps; image segmentation, thresholding, and analysis. In image segmentation, the image is broken down and grouped together by certain specified features. Segmentation can reduce any noise and blurriness that might be present in the image and also simplify the image and make it easier to analyze. During thresholding, the image is broken down even further to isolate the most important object in the image. This allows the program to isolate the food bowl from the background. After successfully isolating the food bowl, the program can then analyze it and determine what level of food is present or if it is empty.

Final Design Process

Configuring the Raspberry Pi

First, the Raspberry Pi was configured to run on the Raspbian OS. The next step was to install the OpenCV library. This step proved to be very difficult because of the low processing power of the A+ model. The installation steps provided by the OpenCV source were simple, but when it came to building the library the Pi would slow down to a standstill and eventually crash. After several failed attempts, even trying to install later versions of OpenCV, I searched elsewhere for installation instructions specific to the Raspberry Pi.

The solution came when I discovered a later version of OpenCV (Version 2.3) is available for the Raspberry Pi for direct installation through the console.

```
$ sudo apt-get update

$ sudo apt-get install libopencv-dev

$ sudo apt-get install python-opencv
```

After that, I was able to use the OpenCV library in both Python and C++. I decided to go with C++ because I had more experience with that language. It is also important to note that CMake was necessary to compile the code.

The Wi-Fi dongle and camera module were simple to install. I only needed to plug them in and enable them on the Pi.

Software Development

The image processing program was developed with three stages in mind; segmentation, thresholding, and analysis. The image was segmented using the K-Means algorithm, the threshold was done using the Hough Circle Transform algorithm, and analysis was done using a histogram.

The K-Means algorithm is an iterative process that is used to cluster data that share a certain feature into groups. The version provided by OpenCV has many specific parameters that allow the user to customize the clustering to best fit their purpose. A programmer can specify how many iterations to go through and how many groups to separate the data points into. I decided to use the RGB values as the criteria for clustering. Doing so reduces number of colors in the image and make the edges of the objects stand out more. Below in **Figure 1** are outputs of K-Means Clustering using 4 and 7 clusters, respectively.

(a)

(b)

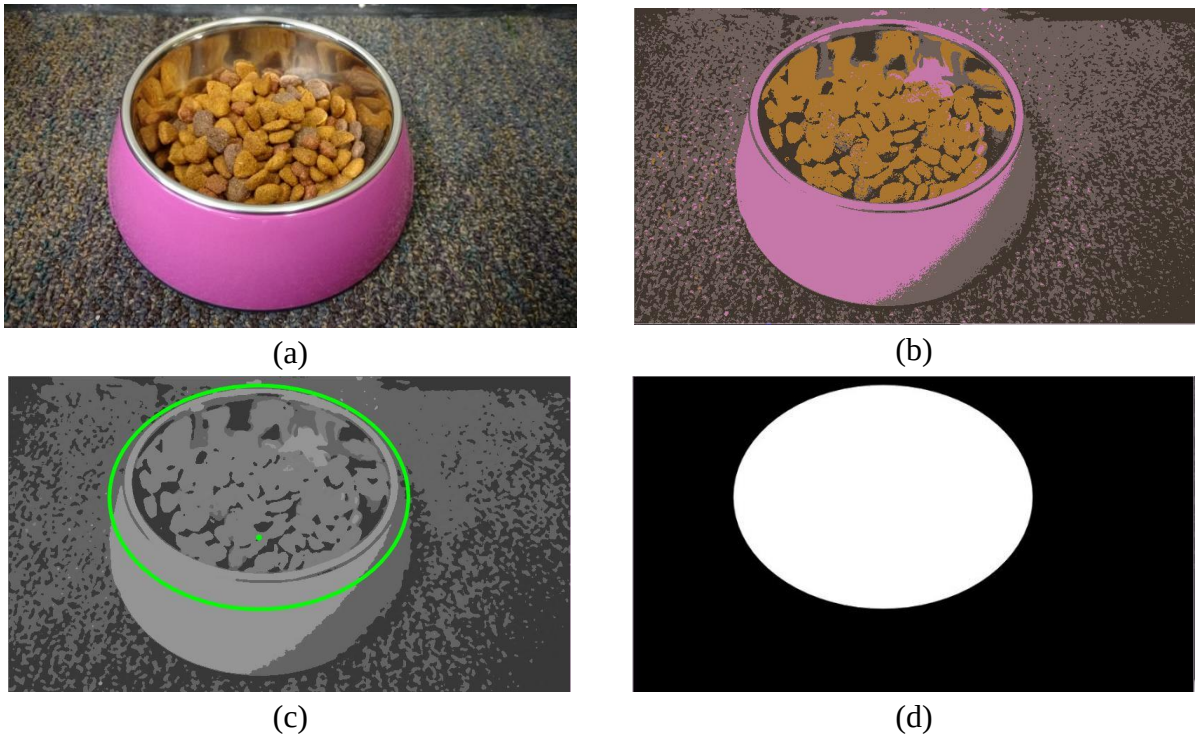
(c)

(d)

Figure 1: *K-Means Clustering with two Images. One set with 7 clusters, (a) & (c), and one set with 4 clusters, (b) & (d).*

the number of clusters reduces the feature variation in the image. This enables a more accurate threshold with the Hough Circle Transform algorithm.

The Hough Circle Transform algorithm is used specifically to find circles in an image. I specified the parameters to so that the algorithm finds only one circle and that circle is used to create a mask. This mask is designed to isolate the food bowl from the rest of the image so the image analysis is more accurate. **Figure 2** shows the resulting image from each step. The algorithm will determine if there is food in the bowl by measuring the intensity values of the bowl. A bowl with food in it will have lower intensity compared to a bowl with no food.



(a) (b)

(c) (d)

Figure 2: *Image segmentation and thresholding process. It shows (a) the original image, (b) the segmented image, (c) the threshold image, and (d) the mask.*

The software then uses the resulting mask to analyze the inside of the food bowl and determine how much food is in it.

User Interface

The system will only interact with the user if there is no food in the bowl. A bash script is used to systematically take a picture of the bowl, pass it through the algorithm to determine if there is food, and notify the user if the bowl is empty. It then will sleep for 1 hour and then repeat the cycle. The notification is sent through email with the message, "NOTICE: Food bowl is empty." Another option is to be notified through a text message. Cell service providers have an email domain that can convert any email sent to it into a text message. The text message will have the same notification as an email would.

System Testing and Analysis

Image analysis and testing was done with a collection of images that had varying amounts of food in the bowl. The amounts of food could be characterized as high, low, or no amount. The software used the mask to create a histogram that labeled and collected each pixel by intensity. This data was then averaged together to generate the average intensity of the entire food bowl.

Amount of Food	Mean Intensity

High Amount	4.887
Low Amount	6.413
No Amount	7.951

Table 2: *Mean intensities based on amount of food in bowl.*

As shown in **Table 2,** the mean intensity of the food bowl increases as the amount of food decreases. Based on this, the software can guess the level of food in the bowl after determining the current mean intensity.

Conclusion

With the Pet Food monitor, a pet owner can carefully monitor their pet's food intake even when they are not around. This system can be applied in many different households because of the small size of size of the Raspberry Pi and Camera module. This system keeps track of the amount of food in the bowl by taking a picture of it and using image processing and analysis. There are many different ways to process this image and approach I chose to take involved K-Means Clustering and Hough Circle Transform algorithms.

Some problems I encountered while building this project came from the low processing power of the Raspberry Pi. The program would take some time to go process the image, however this problem could be solved by simply adapting a more recent and powerful version of the Raspberry Pi.

References

1. "How to Send Email to SMS (Text)". http://www.digitaltrends.com/mobile/how-to-send-e-mail-to-sms-text/#:tIKcj6rn7dsIxA
2. OpenCV Library – Open Source Computer Vision. www.opencv.org
3. Raspberry Pi Source Page. www.raspberrypi.org
4. "Step 3 – Install softwares (for webcam and computer vision)". https://thinkrpi.wordpress.com/2013/04/05/step-3-install-softwares-for-webcam-and-computer-vision/
5. Title Page Image. http://www.modmypi.com/raspberry-pi/model-a-plus/raspberry-pi-model-a-plus-new
6. "Understanding K-Means Clustering". http://docs.opencv.org/master/de/d4d/tutorial_py_kmeans_understanding.html#gsc.tab=0

Appendix A

Image Processing Program – Written in C++ using OpenCV

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;

int main(int argc, char** argv)
{
	Mat src = imread(argv[1], 1);
	Mat samples(src.rows * src.cols, 3, CV_32F);
	for (int y = 0; y < src.rows; y++)
		for (int x = 0; x < src.cols; x++)
			for (int z = 0; z < 3; z++)
				samples.at<float>(y + x*src.rows, z) = src.at<Vec3b>(y, x)[z];


	int clusterCount = 4;
	Mat labels;
	int attempts = 5;
	Mat centers;
	kmeans(samples, clusterCount, labels, TermCriteria(CV_TERMCRIT_ITER |
CV_TERMCRIT_EPS, 10000, 0.0001), attempts, KMEANS_PP_CENTERS, centers);


	Mat new_image(src.size(), src.type());
	for (int y = 0; y < src.rows; y++)
		for (int x = 0; x < src.cols; x++)
		{
			int cluster_idx = labels.at<int>(y + x*src.rows, 0);
			new_image.at<Vec3b>(y, x)[0] = centers.at<float>(cluster_idx, 0);
			new_image.at<Vec3b>(y, x)[1] = centers.at<float>(cluster_idx, 1);
			new_image.at<Vec3b>(y, x)[2] = centers.at<float>(cluster_idx, 2);
		}
	imshow("clustered image", new_image);
	imwrite("clustered_image.jpg", new_image);
	Mat img = imread("clustered_image.jpg", 0);
	Mat cimg;
	Mat thresh = Mat::zeros(img.size(), img.type());

	medianBlur(img, img, 5);
	cvtColor(img, cimg, COLOR_GRAY2BGR);
	std::vector<Vec3f> circles;
	HoughCircles(img, circles, HOUGH_GRADIENT, 1, 500,
		100, 30, 200, 450 // change the last two parameters
				  // (min_radius & max_radius) to detect larger circles
		);
```

```
	for (size_t i = 0; i < circles.size(); i++)
	{
		Vec3i c = circles[i];
		ellipse(cimg, Point(c[0], c[1] * 3 / 4), Size(c[2], c[2] * 3 / 4), 0, 0,
360, Scalar(0, 255, 0), 3, LINE_AA);
		ellipse(thresh, Point(c[0], c[1] * 3 / 4), Size(c[2], c[2] * 3 / 4), 0, 0,
360, Scalar(255, 255, 255), -1, LINE_AA);
		circle(cimg, Point(c[0], c[1]), 2, Scalar(0, 255, 0), 3, LINE_AA);
	}
	imshow("detected circles", cimg);
	imshow("threshold img", thresh);

	Mat hist;
	int histSize = 256;
	float range[] = { 0, 256 };
	const float* histRange = { range };
	calcHist(&img, 1, 0, thresh, hist, 1, &histSize, &histRange, true, false);


	// Draw hist
	int hist_w = 512; int hist_h = 400;
	int bin_w = cvRound((double)hist_w / histSize);

	Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));

	/// Normalize the result to [ 0, histImage.rows ]
	normalize(hist, hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());

	/// Draw for each channel
	for (int i = 1; i < histSize; i++)
	{
		line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(hist.at<float>(i -
1))),
			Point(bin_w*(i), hist_h - cvRound(hist.at<float>(i))),
			Scalar(255, 0, 0), 2, 8, 0);
	}

	/// Display
	namedWindow("calcHist Demo", CV_WINDOW_AUTOSIZE);
	imshow("calcHist Demo", histImage);

	std::cout << "Mean intensity is: " << mean(hist) << std::endl;

	waitKey();
	return 0;
}
```

Appendix B

Bash Script Used to Run System

```
waitKey();

#!bin/bash

a=10;

# create infinite loop

Until [$a –le 10]

do

	# take picture and store

	Raspistill –awb incandescent ––width 800 ––height 450 –o bowl_img.jpg


	# pass image through algorithm

	./foodbowldetect bowl_img.jpg


	#wait

	Sleep 60m

done
```