Original	Translation
The algorithm we will look at in this tutorial is an edge detection algorithm, specifically an edge detection algorithm based on the Sobel operator. This algorithm works by calculating the gradient of the intensity of the image at each point, finding the direction of the change from light to dark and the magnitude of the change. This magnitude corresponds to how sharp the edge is.	Алгоритм, который мы рассмотрим в этом уроке - это алгоритм обнаружения края, в частности, алгоритм обнаружения края на основе оператора Собеля. Этот алгоритм работает при помощи вычисления градиента интенсивности изображения в каждой точке, путём нахождения направления перехода от света к темноте и измерения величины изменения. Эта величина соответствует степени четкости грани.
The Sobel Operator	Оператор Собеля
To calculate the gradient of each point in the image, the image is convolved with the Sobel Kernel. Convolution is done by moving the kernel across the image, one pixel at a time. At each pixel, the pixel and its neighbours are weighted by the corresponding value in the kernel, and summed to produce a new value. This operation is shown in the following diagram.	Для того чтобы вычислить градиент каждой точки изображения, изображение свернуто с помощью ядра Собеля. Свертывание осуществляется путем перемещения ядра через изображение, один пиксель за один раз. На каждом пикселе, пиксель и соседние пиксели измеряются по соответствующему значению в ядре и суммируются для получения нового значения. Эта операция показана на следующей схеме.
We treat it as though the kernel has been overlaid onto the image, with the centre pixel of the kernel aligning we the first pixel in the image. Then we multiple each entry in the kernel by the value beneath it, and sum them to produce a single single output value from that pixel.	Мы обрабатываем его путем накладывания ядра на изображение, при этом центральный пиксель ядра совмещается с первым пикселем в изображении. Затем мы умножаем каждый вход в ядре на величину под ним и суммируем их для того, чтобы получить единое выходное значение из этого пикселя.
For the pixels on the boundary, we just ignore any entries in	Для пикселей на границе мы просто пренебрегаем

the kernel that fall outside.
Edge detection using the Sobel Operator applies two separate kernels to calculate the x and y gradients in the image. The length of this gradient is then calculated and normalised to produce a single intensity approximately equal to the sharpness of the edge at that position.
The kernels used for Sobel Edge Detection are shown below.

любыми входами в ядре, которые выходят за его рамки.
В обнаружении края с помощью оператора Собеля используется два отдельных ядра для расчета x и y градиентов в изображении. Длина этого градиента затем вычисляется и упорядочивается для получения единой интенсивности, приблизительно равной четкости края в этой позиции.
Ядра, используемые для определения краев Собеля, показаны ниже.

The Algorithm

Алгоритм

Now the algorithm can be broken down into its constituent steps
1 Iterate over every pixel in the image
2 Apply the x gradient kernel
3 Apply the y gradient kernel
4 Find the length of the gradient using pythagoras' theorem
5 Normalise the gradient length to the range 0-255
6 Set the pixels to the new values
This illustrates the main steps, though we miss out some specifics such as what we do when we meet the boundary of the image.

Теперь алгоритм можно разбить на составляющие его шаги
1 Перебрать каждый пиксель в изображении
2 Применить x градиент ядра
3 Применить y градиент ядра
4 Найти длину градиента, используя теорему Пифагора
5 Упорядочить длину градиента в диапазоне 0-255
6 Установить новые значения пикселей
Этот алгоритм иллюстрирует основные этапы, хотя мы упускаем некоторые специфические особенности, например действия при встрече границы изображения.

We need to do one tweak so that this will work for our edge detection. When we are iterating over the image, we will be changing value however when we move on to the next pixel, the just changed value will still be in the area of influence of our kernel. For this reason we need to make sure the image

Здесь мы применим одну хитрость, которая будет работать для обнаружения нашего края. Когда мы перебираем изображение, мы будем менять значение, однако, когда мы переходим к следующему пикселю, только что измененное значение будет по-прежнему

we read pixels from and the image we write pixels to are different, and we can do this by just created an empty image, which we will populate with the new values.

находиться в зоне влияния нашего ядра. По этой причине мы должны убедиться, что изображение, из которого мы считываем пиксели, и изображение, в которое мы вписываем пиксели, отличаются, и мы можем сделать это с помощью только что созданного пустого изображения, которое мы наполним новыми значениями.

Now we will read pixels from the original image, but write the edge values to img_edge

Before we start fiddling with the pixel values, we need to decide how we will handle the image boundaries. There are a few different ways to do this, one way is to check each pixel to see if it is on the boundary of the image, and ignore the non-existant pixels on that side. An alternative way would be to wrap the pixels, so that a non-existant pixel on the left edge actually maps to the pixel on the same row on the right edge. For simplicities sake I am simply going to ignore the boundary pixels, instead choosing to start one pixel in from the starting point, and 1 pixel away from the finishing edge. To do this, change the iterators to start at 1, and finish at img.width - 1 and img.height - 1 respectively. Now we can safely index adjacent pixels without falling off the edge of the image.

Теперь мы будем считывать пиксели исходного изображения, а значения края будем вписывать в img_edge (изоб_край).

Прежде чем мы начнем разбираться со значениями пикселей, мы должны решить, как мы будем обрабатывать границы изображения. Есть несколько различных способов сделать это, один из них – это проверить каждый пиксель, чтобы увидеть, находится ли он на границе изображения, и пренебречь несуществующими пикселями на той стороне. Альтернативным способом было бы свёртывание пикселей, так что несуществующий пиксель с левого края отображался бы на пикселе на той же строке, но с правого края. Для простоты я просто собираюсь пренебрегать граничными пикселями, просто выбирая один пиксель от начальной точки, и один пиксель от края. Для того, чтобы сделать это, измените итераторы, чтобы начать с 1, и закончить на img.width - 1 и img.height - 1 соответственно. Теперь мы можем смело указывать соседние пиксели, не попадая за край изображения.

Right, lets move on to applying the gradients. For the x

Теперь давайте перейдем к применению градиентов. Для

gradient, we need the column of pixels to the left and right, for each of the 6 pixels we index we will add up the red, green and blue intensities, multiply it by the kernel, and accumulated it into Gx. We will also do the same for the y gradient. As Gx and Gy both use the same pixels on occasions, it is best to apply the two kernels simultaneously.
Now that the kernels are applied, Gx and Gy contain un-normalised values, which equal the relative length of the gradient in their respective axis. We wish to calculate the length of the gradient, and normalise it to a range suitable for displaying.
Lets calculate the relative length of each gradient. We can do this with pythagoras' theorem:

x градиента, нам нужен столбец пикселей слева и справа, до каждого из 6 пикселей мы добавляем красную, зеленую и синюю яркость, умножаем его на ядро, и собираем его в Gx. Мы также будем делать то же самое для y градиента. Так, как Gx и Gy время от времени используют одни и те же пиксели, то лучше всего применять два ядра одновременно.
Теперь, когда ядра применяются, Gx и Gy содержат неупорядоченные значения, которые равны относительной длине градиента в их соответствующей оси. Мы хотим вычислить длину градиента и упорядочить ее в диапазоне, который подходит для отображения.
Давайте рассчитаем относительную длину каждого градиента. Мы можем сделать это с помощью теоремы Пифагора:

This gives us an un-normalised length, to get the length normalised, we need to know the range of length

Это дает нам неупорядоченную длину, и чтобы получить длину упорядоченную, мы должны знать диапазон длины.

We know that each pixel's intensity value has a range of 0 to 765, due to it being the addition of 3 variables with range 0 to 255. Also, from looking at the kernels, we can see the maximum amount you can accumulate is +/- 4 intensities. Therefore the total range of Gx and Gy is between 0 and 3060. These values are then squared, added and square rooted, giving us a final range of 0 to 4328 (sqrt(2 * sqr(3060))). We can renormalise by dividing by the upper bounds and multiplying by 255.

Мы знаем, что значение интенсивности каждого пикселя имеет диапазон от 0 до 765, из-за того, что он является сложением 3 переменных с диапазоном от 0 до 255. Кроме того, глядя на ядра, мы можем увидеть максимальную сумму, которую можно накопить, и она составляет + / - 4 яркости. Поэтому полный спектр Gx и Gy составляет от 0 до 3060. Эти значения затем возводятся в квадрат, суммируются, и затем извлекаются квадратные корни, что дает нам окончательный диапазон от 0 до 4328

Right at the end of the code we need to change displayImage to display our new image rather than the original. Other than that, the script is finished, and we can test it to see the results. The script should produce a new grayscale image, highlighting edges from the original image.

This concludes this tutorial. Below are the code listings for this tutorial. The C code will actually continually capture images from the webcam, and you will find that on the Raspberry Pi it is capable of displaying the edge detected image in realtime.

(SQRT (2 * SQR (3060))). Мы можем переупорядочить их путем деления на верхние грани и умножения на 255.

В самом конце кода мы должны изменить displayImage, чтобы отобразить наше новое изображение, а не оригинал. Таким образом, скрипт закончен, и мы можем проверить его, чтобы увидеть результаты. Скрипт должен производить новое полутоновое изображение, выделяя края из исходного изображения.

На этом заканчивается этот урок. Ниже приведен исходный код программы для этого урока. Код на C будет постоянно захватывать изображения с веб-камеры, и вы увидите, что на Raspberry Pi возможно отображать края найденного изображения в режиме реального времени.