LINE FOLLOWING TECHNIQUES

A line follower robot is a robot which follows a certain path controlled by a feed back mechanism. The control of the robot is the most important aspect of its working. Here the term control refers to the robot motion control, i.e. controlling the movement of the wheels. A basic line follower robot follows certain path and the motion of the robot along this path is controlled by controlling the rotation of wheels, which are placed on the shafts of the two motors. So, the basic control is achieved by controlling the motors. The control circuitry involves the use of sensors to sense the path and the microcontroller or any other device to control the motor operation through the motor drivers, based on the sensor output.

METHOD 1: **Using IR transmitter and receiver**

IR sensor consists of IR transmitter and IR receiver on a board. When the vehicle is moving on a black line, IR rays are continuously absorbed by the black surface and there is no reflected ray making output high. Whenever, the robot moves out to the white surface, it starts reflecting the IR rays and making the output low. Thus depending on the output of IR sensor microcontroller indicates the motors to change their direction.

Design of IR Sensors:

IR sensor circuit consists of mainly IR transmitter and IR receiver. IR transmitter is similar to an LED. Its operating voltage is around 1.4V. So to protect it, a 10k resistor is placed before IR and is connected in forward biased. IR receiver is connected in reverse bias and a 15K resistor is placed between VCC and the receiver. Output is taken between resistor and IR receiver.

Working of IR Sensors:

The IR transmitter continuously transmits the IR rays. When IR transmitter is on the black surface these rays were absorbed by the surface and when it is on white surface these rays were reflected. The IR receiver has maximum resistance when no IR rays are received and voltage from VCC flows through the resistor. At the output pin, voltage is approximately 5v.

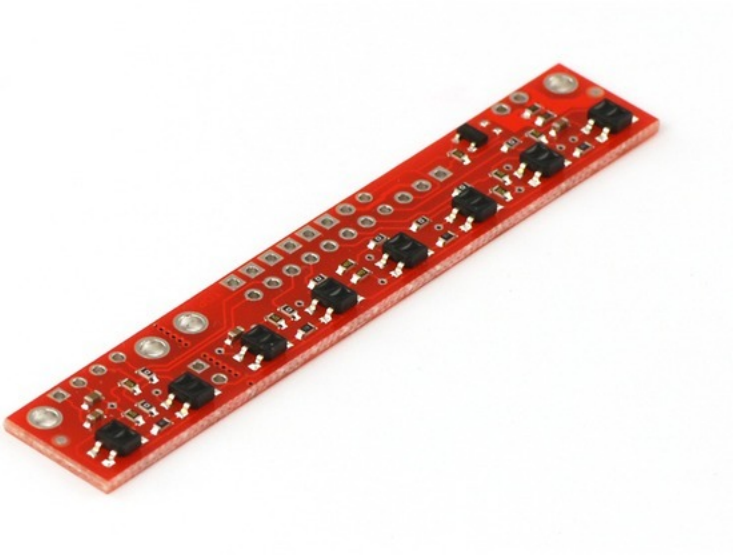
As the intensity IR rays received by the receiver increases, resistance value decreases and reverse break down occurs .Thus voltage through the resistor is grounded. So, at the output pin, it will produce 0V.

Limitations of Line Follower Robot:

1. Line follower robot requires 2-3 inches broad line.
2. It may not move properly if the black line drawn is of low intensity.
3. The IR sensors may sometimes absorb IR rays from surroundings also. As a result, robots may move in improper way.

METHOD 2: Using Line sensor

POLOLU QTR-8RC Reflectance Sensor Array: This sensor module has 8 IR LED/phototransistor pairs mounted on a 0.375" pitch, making it a great detector for a line-following robot. Pairs of LEDs are arranged in series to halve current consumption, and a MOSFET allows the LEDs to be turned off for additional sensing or power-savings options. Each sensor provides a separate **digital I/O-measurable output**.



The QTR-8RC reflectance sensor array is intended as a line sensor, but it can be used as a general-purpose proximity or reflectance sensor. The module is a convenient carrier for eight IR emitter and receiver (phototransistor) pairs evenly spaced at intervals of 0.375" (9.525 mm). To use a sensor, you must first charge the output node by applying a voltage to its OUT pin. You can then read the reflectance by withdrawing the externally supplied voltage and timing how long it takes the output voltage to decay due to the integrated phototransistor. Shorter decay time is an indication of greater reflection. This measurement approach has several advantages, especially when coupled with the ability of the QTR-8RC module to turn off LED power:

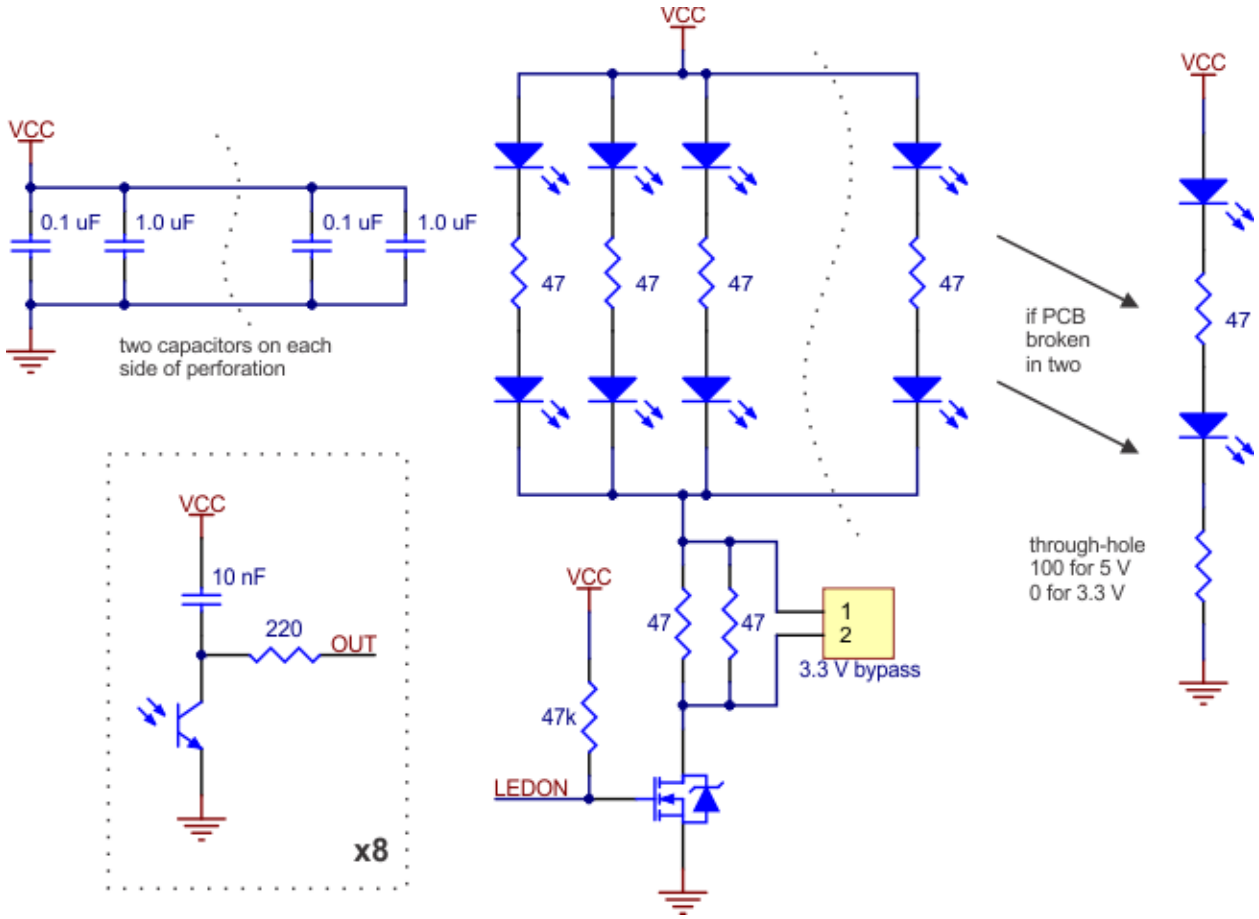
- No analog-to-digital converter (ADC) is required
- Improved sensitivity over voltage-divider analog output
- Parallel reading of multiple sensors is possible with most microcontrollers
- Parallel reading allows optimized use of LED power enable option

The outputs are all independent, but the LEDs are arranged in pairs to halve current consumption. The LEDs are controlled by a MOSFET with a gate normally pulled high, allowing the LEDs to be

turned off by setting the MOSFET gate to a low voltage. Turning the LEDs off might be advantageous for limiting power consumption when the sensors are not in use or for varying the effective brightness of the LEDs through PWM control.

This sensor was designed to be used with the board parallel to the surface being sensed.

The LED current-limiting resistors for 5 V operations are arranged in two stages; this allows a simple bypass of one stage to enable operation at 3.3 V. The LED current is approximately 20–25 mA, making the total board consumption just less than 100 mA.



METHOD 3: Using Image Processing

The system uses:

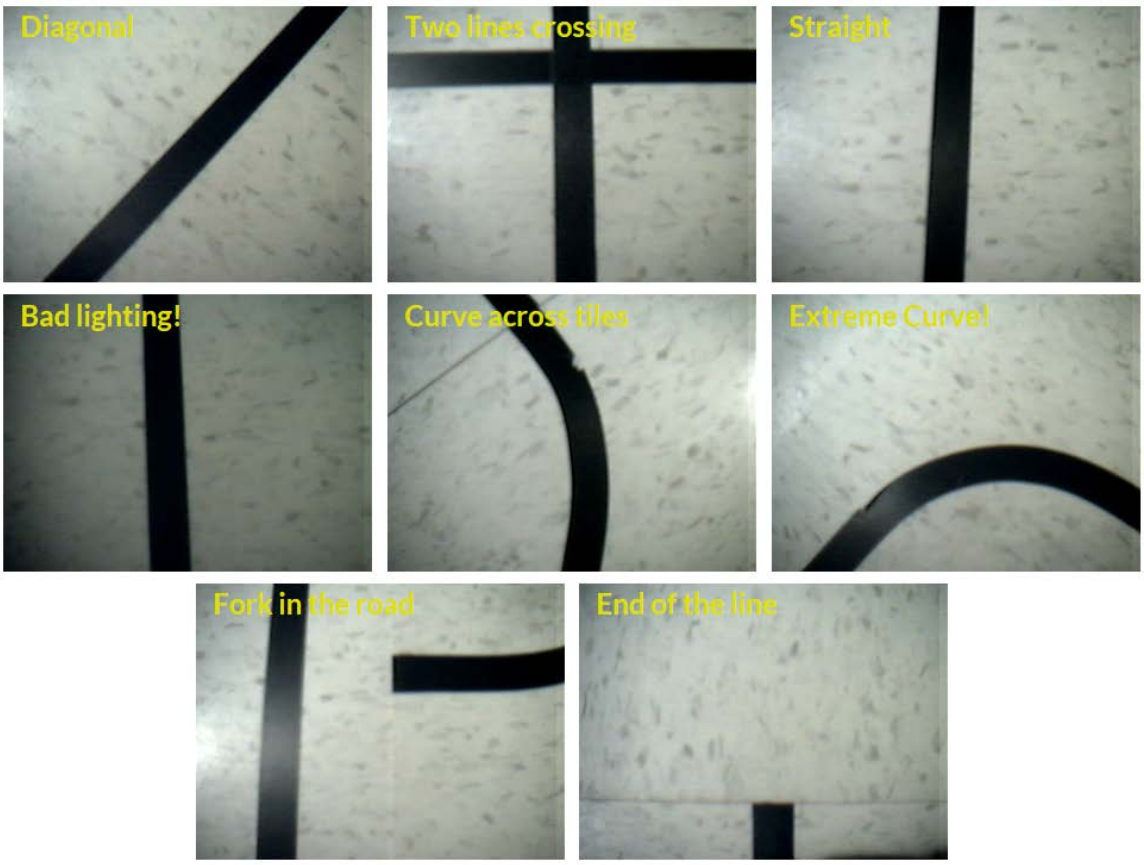
- single CCD camera
- USB Digitizer
- Pentium CPU on board

- Left, Right differential motors

To get a better idea of what we're trying to accomplish lets first look at some sample pictures that the BucketBot took. The images are from the CCD camera mounted to the front of the BucketBot angled towards the ground.

Note that we have not done any calibration of lighting adjustments. The images are straight from the camera.

If it worth mentioning that the lines are created by black electrical tape stuck on moveable floor tiles. This allows us to move the tiles around and experiment with shapes quite easily.

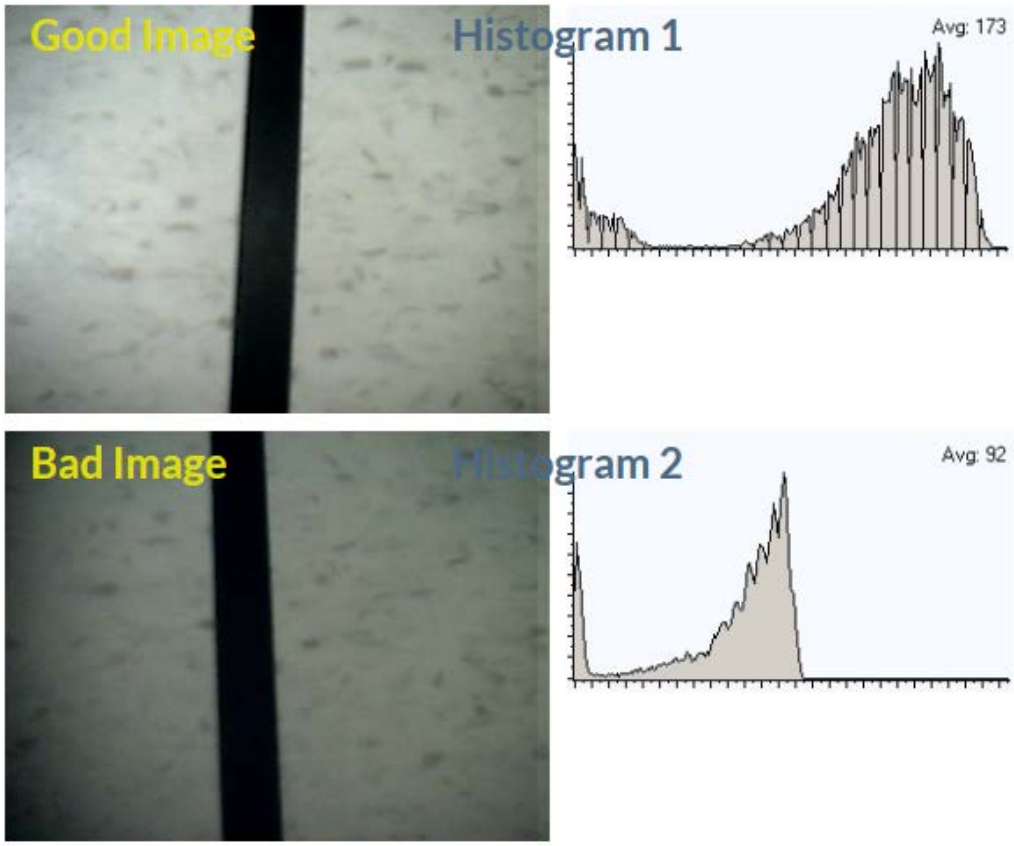


Lighting Issues...

Bad lighting can really cause problems with most image analysis techniques (esp. when thresholding). Imagine if a robot were to suddenly move under a shadow and lose all control! The best vision techniques try to be more robust to lighting changes.

To understand some of those issues lets look at two histograms from a straight line image from the previous slide. Next to each of these images are the images histogram. The histogram of an image is a graphical representation of the count of different pixel intensities in the image.

As you can see, histograms for lighter images slump towards the right (towards the 255 or highest value) whereas darker images have histograms with most pixels closer to zero. This is obvious but using the histogram representation we can better understand how transformations to the image change the underlying pixels.



The next step is to see if we can correct these two images so that they look closer to one another. We do that by normalizing the images.

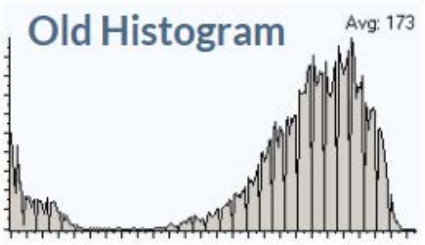
Normalize Intensities

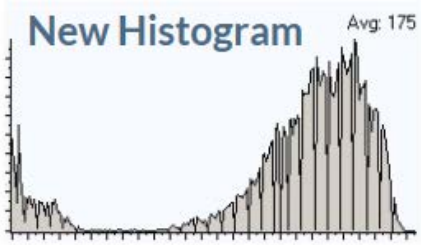
To counter the effects of bad lighting we have to normalize the image. Image normalization attempts to spread the pixel intensities over the entire range of intensities. Thus if you have a very dark image the resulting normalization process will replace many of the dark pixels with lighter pixels while keeping the relative positions the same, i.e. two pixels may be made lighter but the darker of the two will still be darker relative to the second pixel.

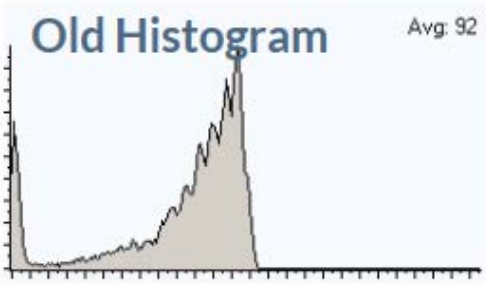
By evenly distributing the image intensities other image processing functions like thresholding which are based on a single cutoff pixel intensity become less sensitive to lighting.

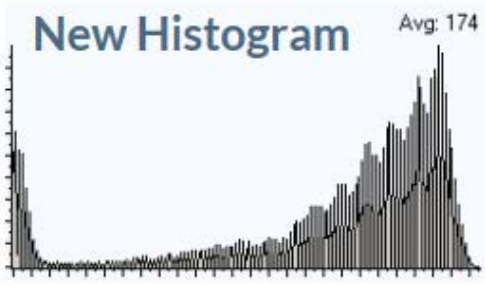
For example, the following images from the previous page show what happens during normalization. It is important to note that any image transformation that is meant to improve bad images must also

preserve already good ones. In testing image processing functions be sure to always understand the negative side effects that any function can have.









The bad image experienced a large amount of change as the image intensities did not cover the entire intensity range due to bad lighting. You can see from the histogram that the image intensities are now more evenly distributed.

Also you can note how the new histogram appears to be not as solid as the original. This is due to how the intensity values are stretched. Since the new image has exactly the same number of pixels as the old image the new image still has many pixels intensity values that do not exist and therefore show up as gaps in the histogram. Adding another filter like a mean blur would cause the histogram to become more solid again as the gaps would be filled due to smoothing of the image.

Next we need to start focusing on extracting the actual lines in the images.

Edge Detection

In order to follow the line we need to extract properties from the image that we can use to steer the robot in the right direction. The next step is to identify or highlight the line with respect to the rest of the image. We can do this by detecting the transition from background tile to the line and then from the line to the background tile. This detection routine is known as edge detection.

The way we perform edge detection is to run the image through a convolution filter that is 'focused' on detecting line edges. A convolution filter is a matrix of numbers that specify values that are to be multiplied, added and then divided from the image pixels to create the resulting pixel value.

To learn more about convolution filters have a look at our help section.

We will start with the following convolution matrix that is geared to detect edges:

-1	-1	-1
-1	8	-1
-1	-1	-1

The next step is to run the images through this edge detector and see what we get ...

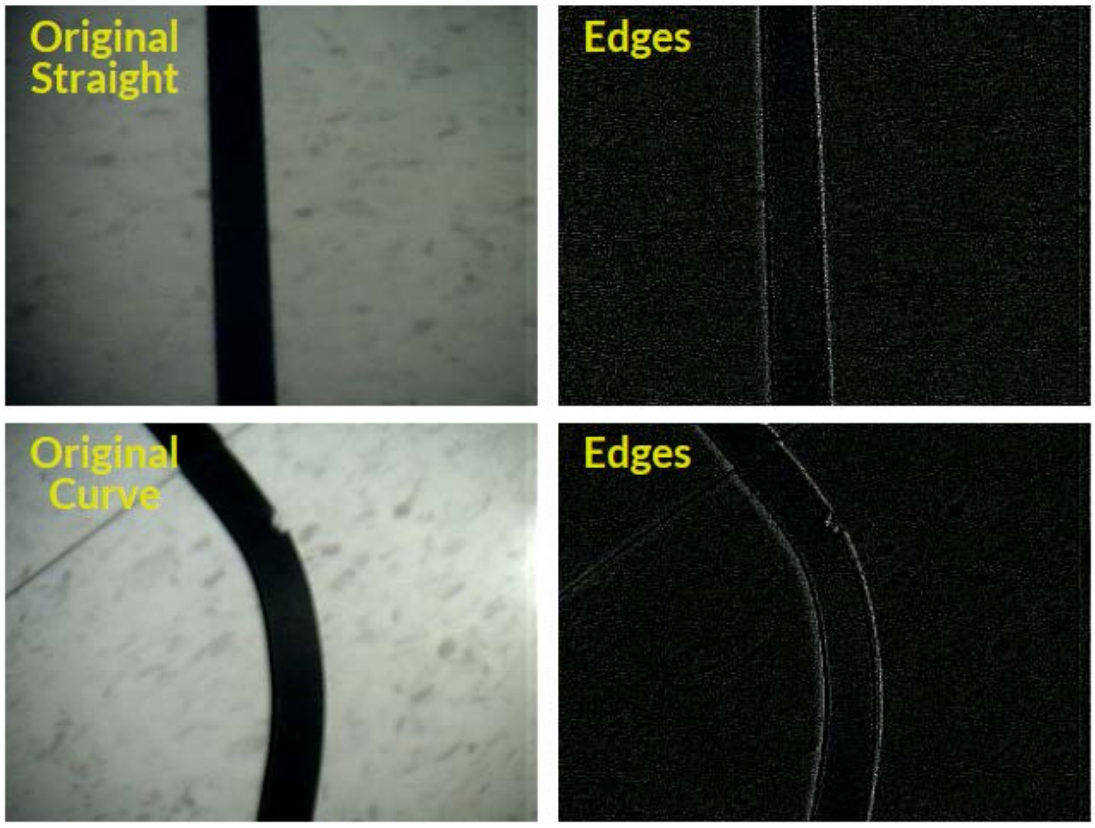
Edge Detection

Here are two sample images with the convolution edge detection matrix run after normalization:

It is interesting to note that while the convolution filter does identify the line it also picks up on a lot of speckles that are not really desired as it causes noise in the resulting image.

Also note that the detected lines are quite faint and sometimes even broken. This is largely due to the small (3x3) neighborhood that the convolution filter looks at. It is easy to see a very large difference

between the line and the tile from a global point of view (as how you and I look at the images) but from the image pixel point of view it is not as easy. To get a better result we need to perform some modifications to our current operations.



Modified Line Detection

To better highlight the line we are going to:

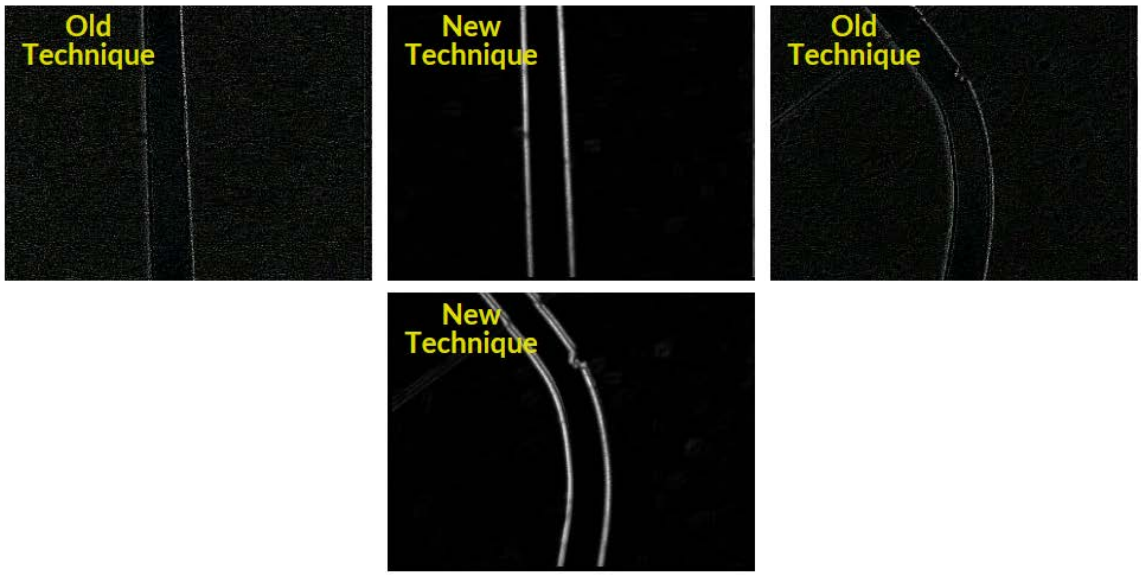
1. Use a larger filter; instead of a 3x3 neighborhood we will use a 5x5. This will result in larger values for edges that are "thicker". We will use the following matrix:

```
-1-1-1-1-1
-1 0 0 0-1
-1 016 0-1
-1 0 0 0-1
-1-1-1-1-1
```

2. Further reduce the speckling issue we will square the resulting pixel value. This causes larger values to become larger but smaller values to remain small. The result is then normalized to fit into the 0-255 pixel value range.

3. Threshold the final image by removing any pixels lower than a 40 intensity value.

We now nicely see the line edges and most of the noise speckles are gone. We can now continue to the next step which is how to understand these images in order to map the results to left and right motor pulses.



Center of Gravity

There are many ways we could translate the resulting image intensities into right and left motor movements. A simple way would be to add up all the pixel values of the left side of the image and compare the result to the right side. Based on which is more the robot would move towards that side. At this point, however, we will use the COG or Center of Gravity of the image to help guide our robot.

The COG of an object is the location where one could balance the object using just one finger. In image terms it is where one would balance all the white pixels at a single spot. The COG is quick and easy to calculate and will change based on the object's shape or position in an image.

To calculate the COG of an image add all the x,y locations of non-black pixels and divide by the number of pixels counted. The resulting two numbers (one for x and the other for y) is the COG location.

The COG location is the red square with a green line from the center of the image to the COG location.

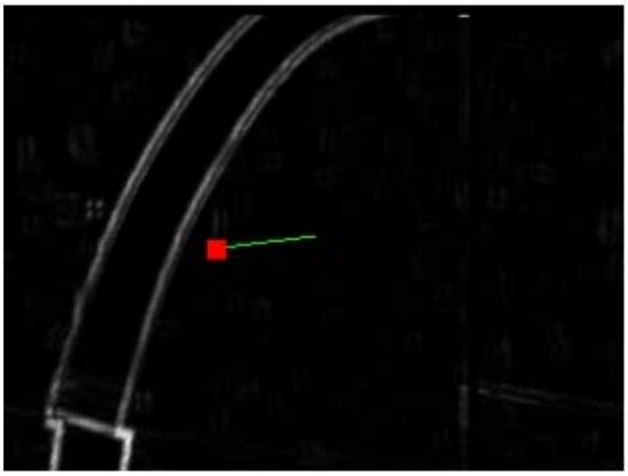
Based on the location of the COG we can now apply the following conditions to the robot motors:

- When the COG is to the right of the center of screen, turn on the left motor for a bit.
- When the COG is on the left, turn on the right motor.
- When the COG is below center, apply reverse power to opposite motor to pivot the robot.

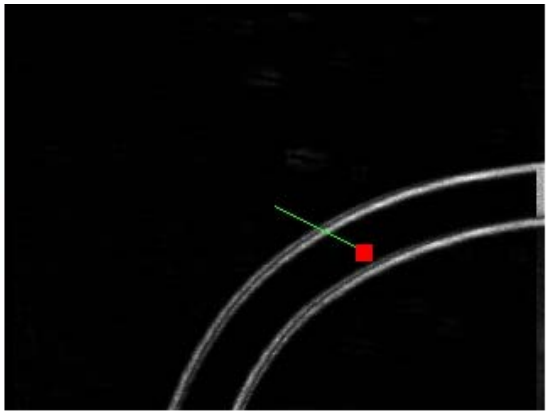
You can also try other conditions such as when the distance of the COG to the center of screen is large really turn up the motor values to try to catch up to the line.

This technique works regardless of the color of the line ... as long as the line can be well differentiated from the background we can detect it!

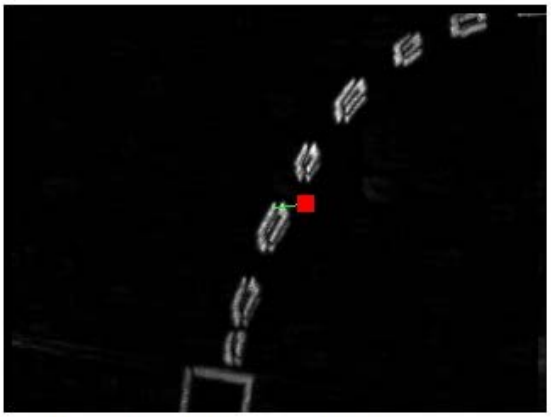
This technique works regardless of the color of the line ... as long as the line can be well differentiated from the background we can detect it!



It even works if we reverse the original black line on white tile assumption.



In fact, we don't really even need a connected line!



Here is a case where the end of the line is reached. Notice that the COG has fallen below the center of the screen. Perhaps we can use this condition to stop the robot!