VACATION TASKS

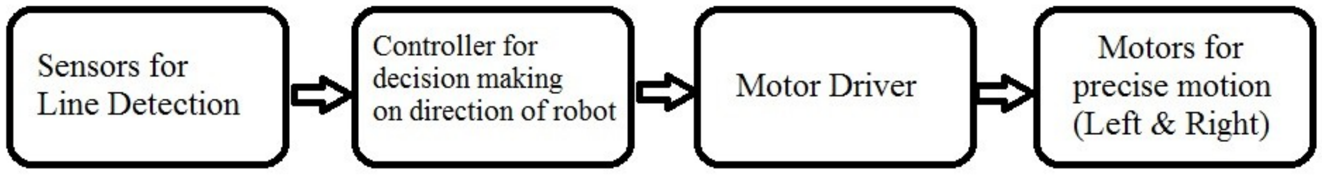
SET-5

Line Following Techniques

-

Tanush Biju

Using an IR sensor

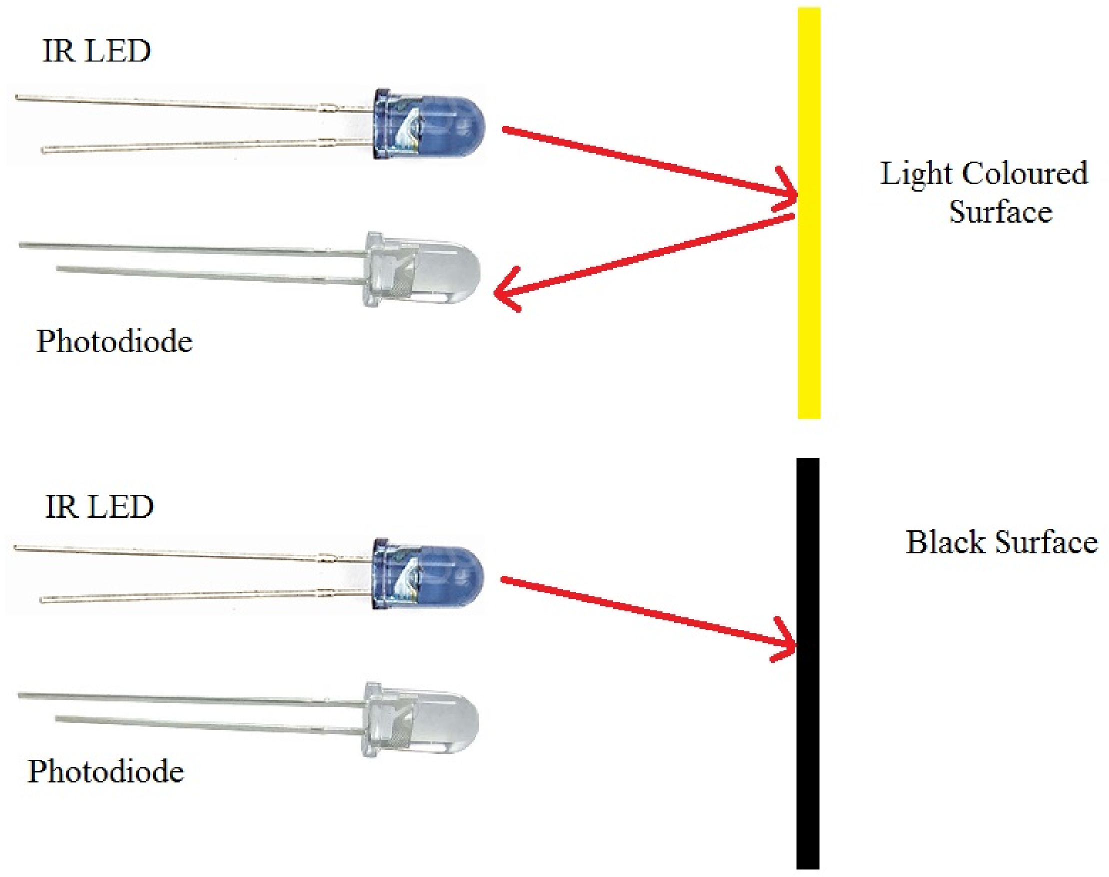


Sensors for Line Detection
Controller for decision making on direction of robot
Motor Driver
Motors for precise motion (Left & Right)

Block Diagram for Line Follower Robot

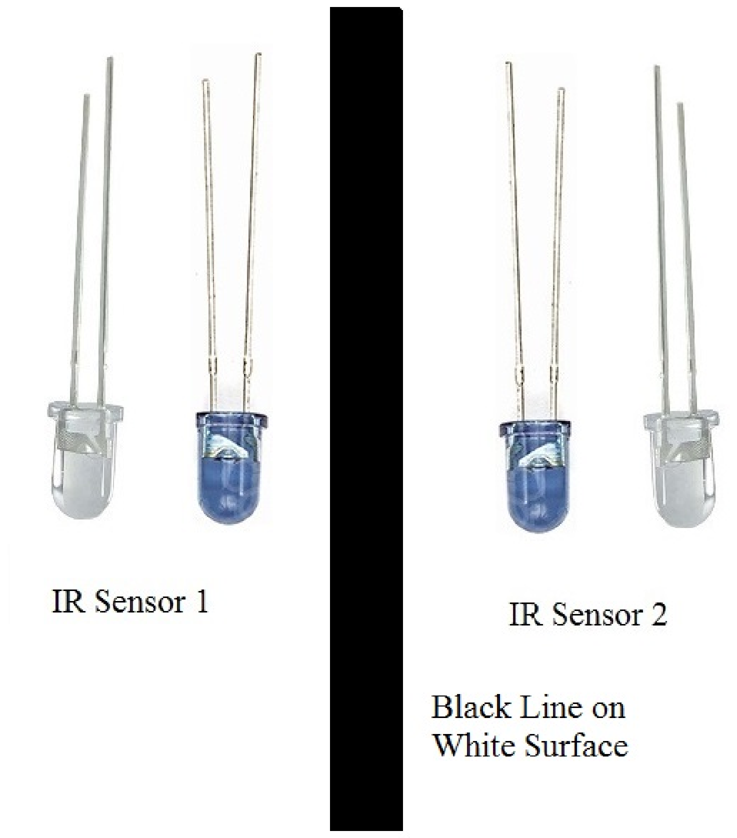
As mentioned in the block diagram, we need sensors to detect the line. For line detection logic, we used two IR Sensors, which consists of IR LED and Photo-diode. They are placed in a reflective way i.e. side – by – side so that whenever they come in to proximity of a reflective surface, the light emitted by IR LED will be detected by Photo diode.

The following image shows the working of a typical IR Sensor (IR LED – Photo-diode pair) in front of a light colored surface and a black surface. As the reflectance of the light colored surface is high, the infrared light emitted by IR LED will be maximum reflected and will be detected by the Photo-diode.



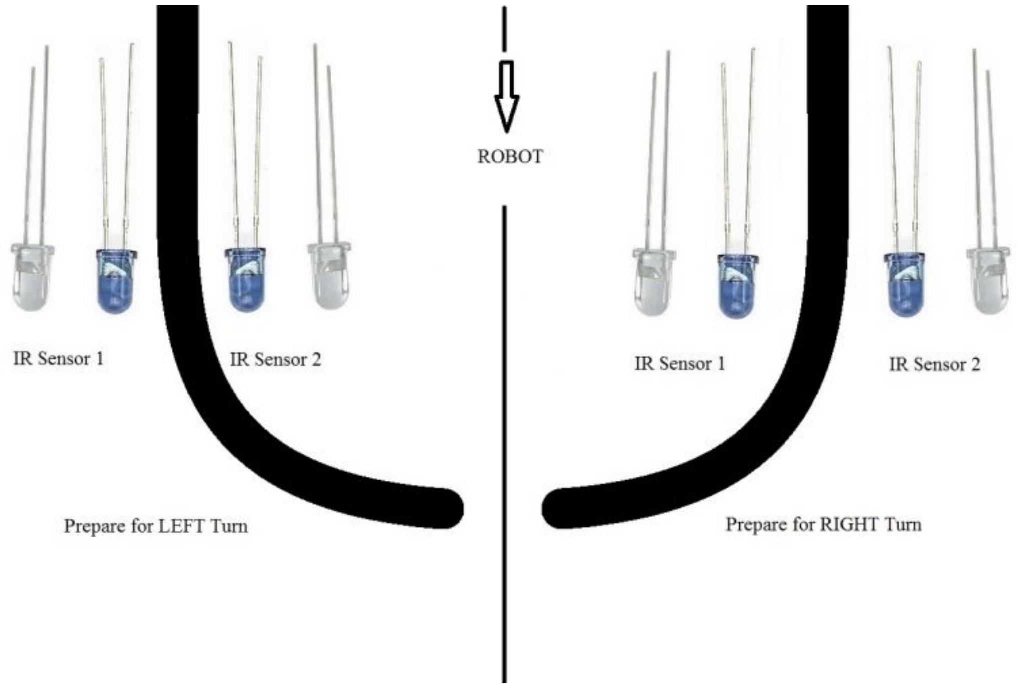
In case of black surface, which has a low reflectance, the light gets completely absorbed by the black surface and doesn’t reach the photodiode.

Using the same principle, we will setup the IR Sensors on the Line Follower Robot such that the two IR Sensors are on the either side of the black line on the floor. The setup is shown below.



When the robot moves forward, both the sensors wait for the line to be detected. For example, if the IR Sensor 1 in the above image detects the black line, it means that there is a right curve (or turn) ahead.

Arduino UNO detects this change and sends signal to motor driver accordingly. In order to turn right, the motor on the right side of the robot is slowed down using PWM, while the motor on the left side is run at normal speed.



Similarly, when the IR Sensor 2 detects the black line first, it means that there is a left curve ahead and the robot has to turn left. For the robot to turn left, the motor on the left side of the robot is slowed down (or can be stopped completely or can be rotated in opposite direction) and the motor on the right side is run at normal speed.

Arduino UNO continuously monitors the data from both the sensors and turns the robot as per the line detected by them.

CODE:-

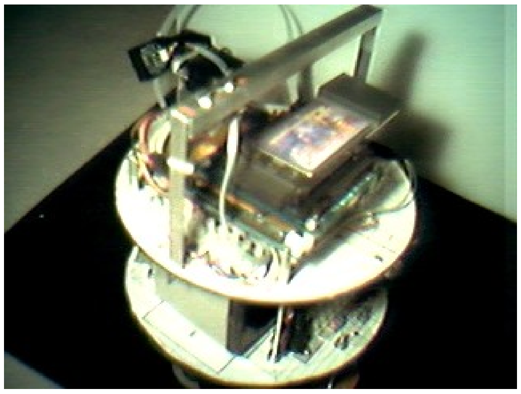
```
int mot1=9;
int mot2=6;
int mot3=5;
int mot4=3;

int left=13;
int right=12;

int Left=0;
int Right=0;

void LEFT (void);
void RIGHT (void);
void STOP (void);

void setup()
{
  pinMode(mot1,OUTPUT);
  pinMode(mot2,OUTPUT);
  pinMode(mot3,OUTPUT);
  pinMode(mot4,OUTPUT);

  pinMode(left,INPUT);
  pinMode(right,INPUT);

  digitalWrite(left,HIGH);
  digitalWrite(right,HIGH);


}

void loop()
{

analogWrite(mot1,255);
analogWrite(mot2,0);
analogWrite(mot3,255);
analogWrite(mot4,0);

while(1)
{
  Left=digitalRead(left);
  Right=digitalRead(right);

  if((Left==0 && Right==1)==1)
  LEFT();
  else if((Right==0 && Left==1)==1)
  RIGHT();
}
}

void LEFT (void)
{
   analogWrite(mot3,0);
   analogWrite(mot4,30);


   while(Left==0)
   {
    Left=digitalRead(left);
    Right=digitalRead(right);
    if(Right==0)
    {
      int lprev=Left;
      int rprev=Right;
      STOP();
      while(((lprev==Left)&&(rprev==Right))==1)
      {
         Left=digitalRead(left);
         Right=digitalRead(right);
      }
    }
    analogWrite(mot1,255);
    analogWrite(mot2,0);
   }
   analogWrite(mot3,255);
   analogWrite(mot4,0);
}

void RIGHT (void)
{
   analogWrite(mot1,0);
   analogWrite(mot2,30);

   while(Right==0)
   {
    Left=digitalRead(left);
    Right=digitalRead(right);
    if(Left==0)
    {
      int lprev=Left;
      int rprev=Right;
     STOP();
      while(((lprev==Left)&&(rprev==Right))==1)
      {
         Left=digitalRead(left);
         Right=digitalRead(right);
      }
    }
    analogWrite(mot3,255);
    analogWrite(mot4,0);
    }
   analogWrite(mot1,255);
   analogWrite(mot2,0);
}
void STOP (void)
{
analogWrite(mot1,0);
analogWrite(mot2,0);
analogWrite(mot3,0);
analogWrite(mot4,0);

}
```

- In order to increase the efficiency of black line detection, number of sensors can be increased. An array of sensors will be more accurate than just two sensors.
- In this project (where two sensors are used), the positioning of the sensors is very important. The width of the black line plays a major role in the placement of the sensors.
- The sensor to detect the line can also be constructed using an LED and LDR pair.

Visual Line Following

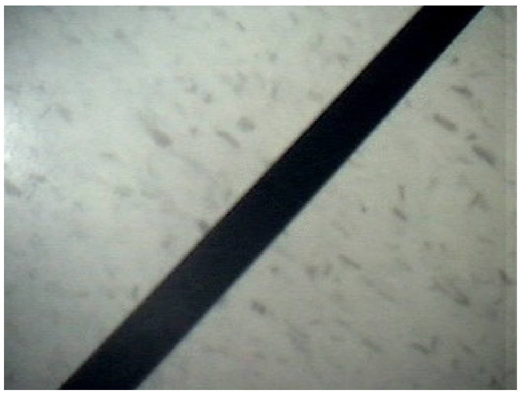
The system uses:

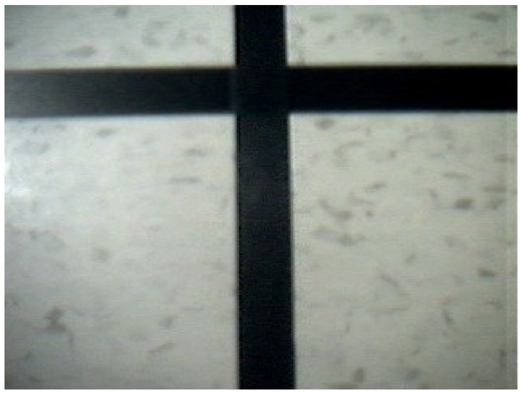
- single CCD camera
- USB Digitizer
- Pentium CPU on board
- Left, Right differential motors



Example Images

The images are from the CCD camera mounted to the front of the bot angled towards the ground.



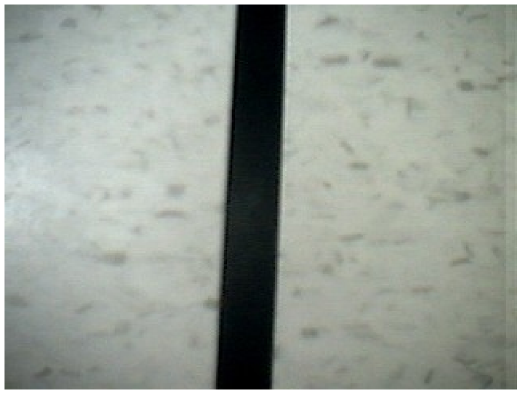


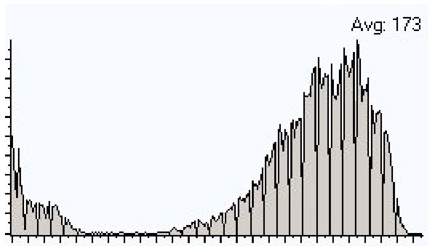
Lighting Issues

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!

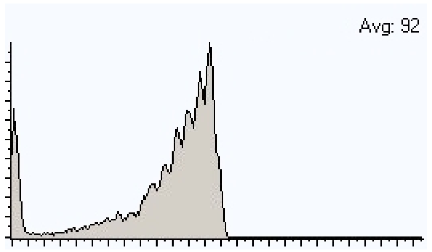
The best vision techniques try to be more robust to lighting changes.
To understand some of those issues lets look at two histograms from a straight line image from the previous slide. Next to each of these images are the images histogram. The histogram of an image is a graphical representation of the 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.





Good Image



Bad Image

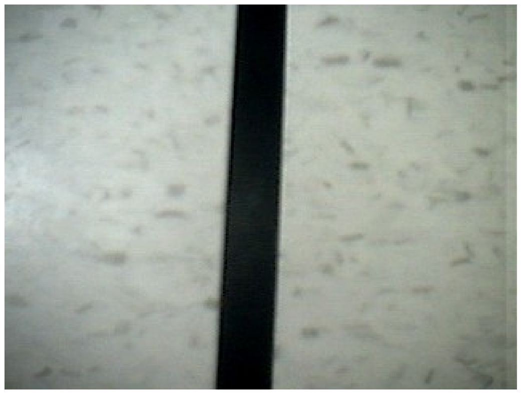
The next step is to see if we can correct these two images so that they look closer to one another. We do that by normalizing the images.

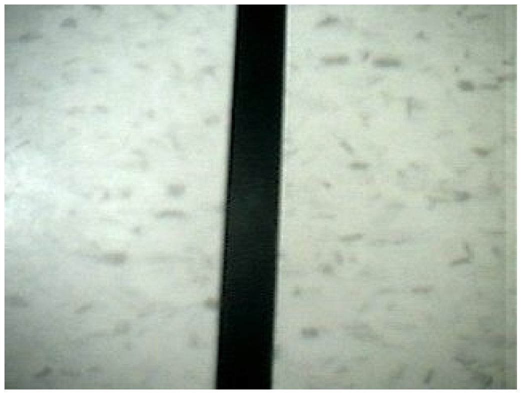
Normalize Intensities

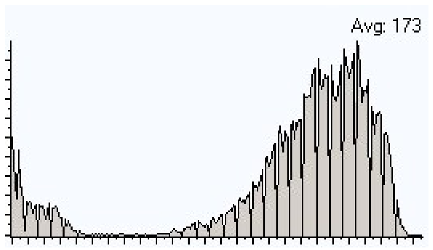
To counter the effects of bad lighting we have to normalize the image. Image normalization attempts to spread the pixel intensities over the entire range of intensities. Thus if you have a very dark image the resulting normalization process will replace many of the dark pixels with lighter pixels while 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.

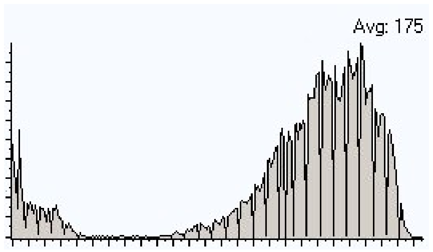
By evenly distributing the image intensities other image processing functions like thresholding which are based on a single cutoff pixel intensity become less sensitive to lighting.

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.

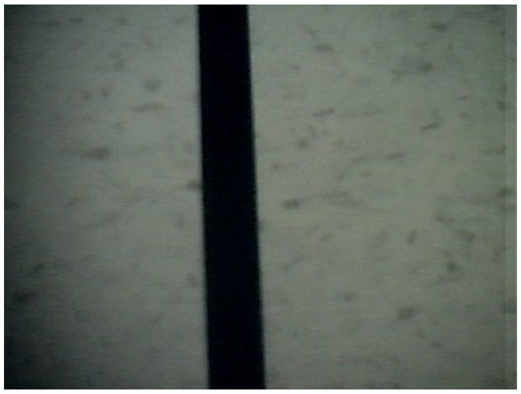


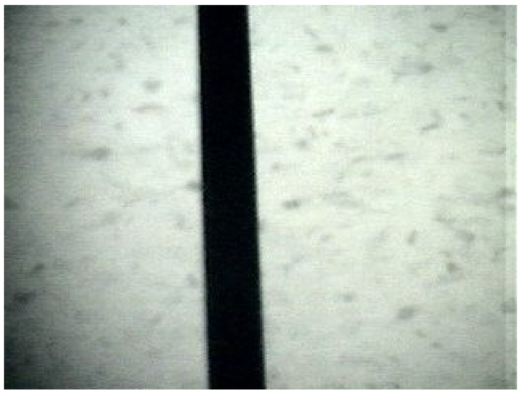


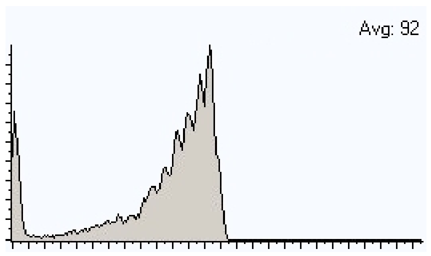


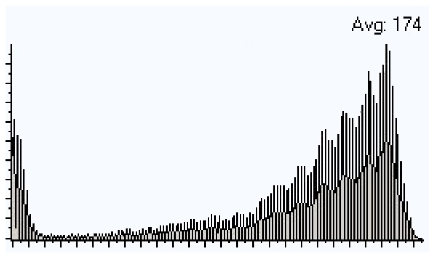


Normalization did not create much change because image was lighted brightly to begin with.









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.

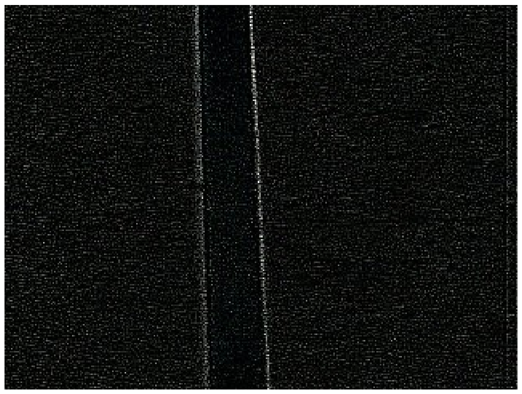
The following convolution matrix is geared to detect edges:

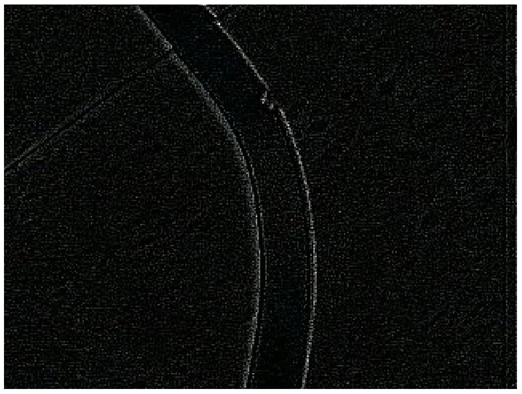
-1	-1	-1
-1	8	-1
-1	-1	-1

The next step is to run the images through this edge detector.

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.

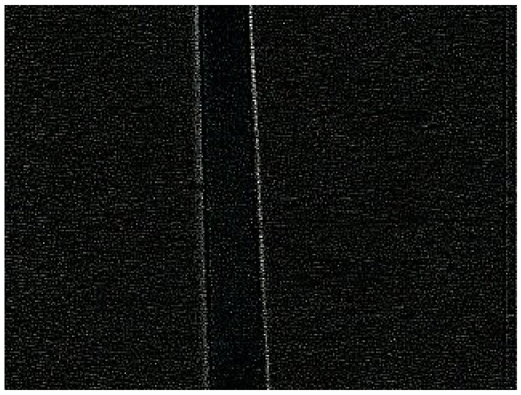
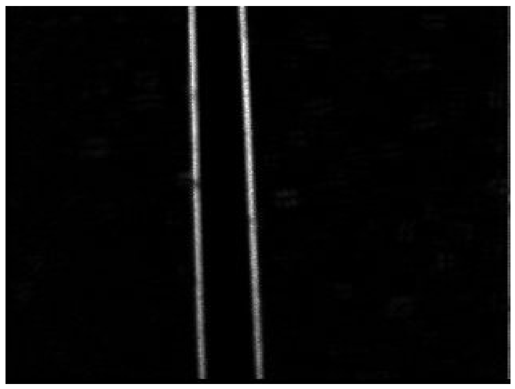
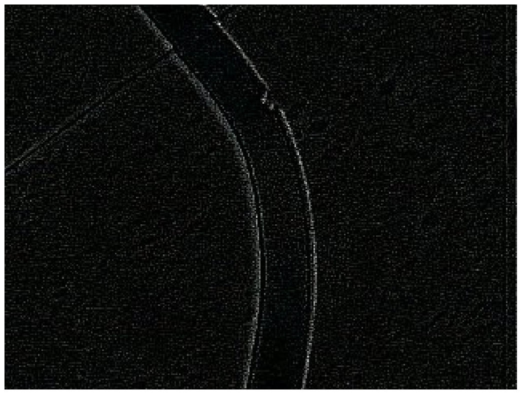
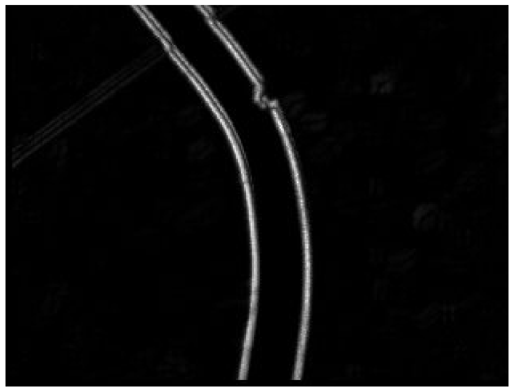
Modified Line Detection

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.

The results of this modified technique:

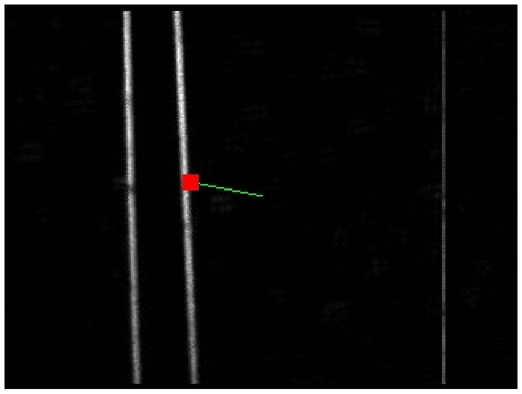
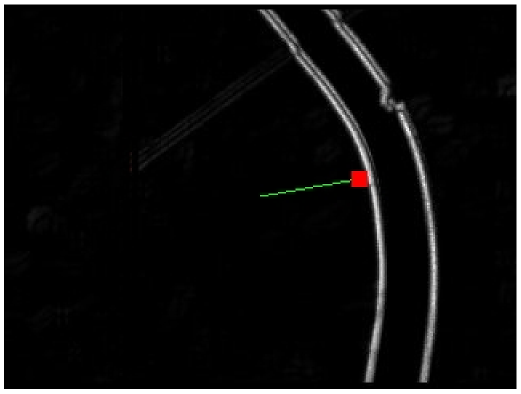
Center of Gravity

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.