Requirements

- Gear Motors x2
- Wheels x2, chassis, caster wheel, breadboard
- Arduino UNO
- L293D Motor Driver
- array of 8 IR-sensors / 8 different IR-sensors.

What are the steps In maze solving?
There are basically 2 steps. The first is to drive through the maze and find the end of it. The second is to optimize that path so your robot can travel back through the maze, but do it perfectly with out going down any dead ends.

How does the robot find the end of the maze?
I use a technique called the left hand on the wall. Imagine you are in a maze and you keep your left hand on a the edge of the wall at all times. Doing this would eventually get you out of a non-looping maze. This instructable will only deal with mazes that do not loop back on themselves.

This left hand on wall algorithm can be simplified into these simple conditions:
- If you can turn left then go ahead and turn left,
- else if you can continue driving straight then drive straight,
- else if you can turn right then turn right.
- If you are at a dead end then turn around.

The robot has to make these decisions when at an intersection. An intersection is any point on the maze where you have the opportunity to turn. If the robot comes across an opportunity to turn and does not turn then this is consider going straight. Each move taken at an intersection or when turning around has to be stored.

L = left turn
R= right turn
S= going straight past a turn
B= turning around

So let us apply this method to a simple maze and see if you can follow it. View the photos to see this method in action.
The red circle will be the robot.

As you can see in the photos for this example, the final path is LBLLBSR.

Ok so now you have a path. In this case it is "LBLLBSR", but how does the robot change that into the correct path? Well lets take a look at what the correct path would be. Look at the photos for the correct path.

Final correct path = SRR

So we need our path to go from LBLLBSR to the right path that is SRR. To start off we look at where we went wrong. A "B" indicates the robot turned around meaning it went down the wrong path. To optimize the path we have to get rid of the "B" by using some substitution.

Lets look at the first 3 moves in the path "LBLLBSR". These moves are "LBL".
That move looks like the photo.

Instead of turning left then turning around and turning left again, the robot should have gone straight. So we can say that LBL = S.

This substitution is what the robot uses to optimize the path. That is one example but here is the whole list:

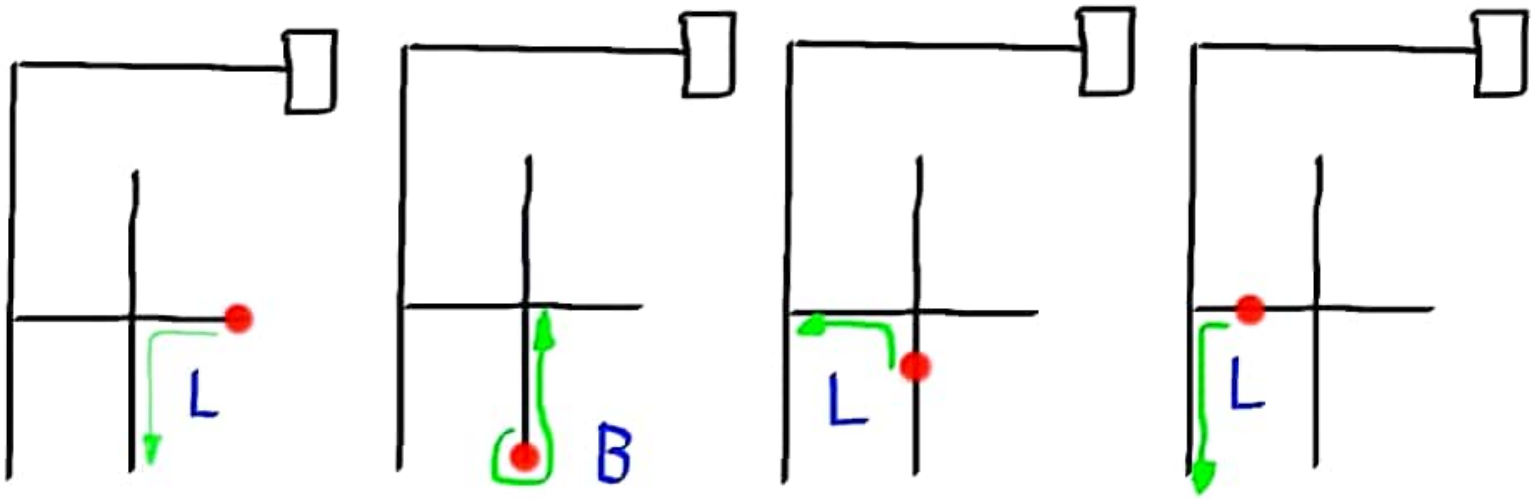
LBR = B
LBS = R
RBL = B
SBL = R
SBS = B
LBL = S

You may not come across all of these when maze solving, but they are required when

You may not come across all of these when maze solving, but they are required when optimizing the path. Some even put "B" back into the path. This is required to further optimize the path correctly. You can figure out why for yourself or just trust me.

Lets optimize our path now that we know how to:
Path = LBLLBSR
LBL = S so our new path would be: SLBSR
We also know LBS = R so our new path would be: SRR
As you can see we got the path that we were looking for.

My robot optimizes the path as it travels. The path is stored in an array and every time it goes to store a new move, it checks to see if the previous move was a "B", if it was then it optimizes the path. You need to know at least 3 moves to optimize the path: The move before and after the turn around (and the turn around itself).

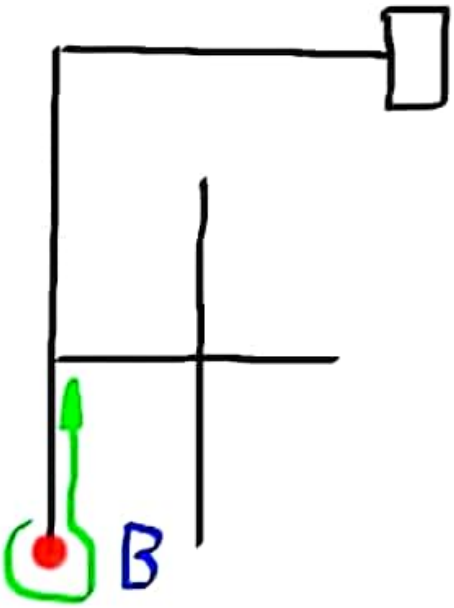


Path = L

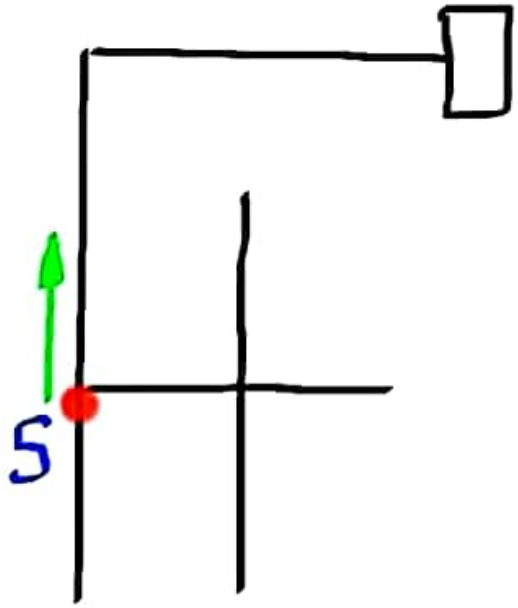
Path = LB

Path = LBL

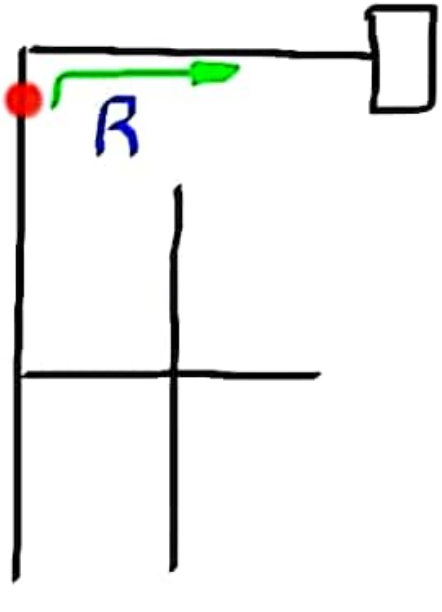
Path= LBLL



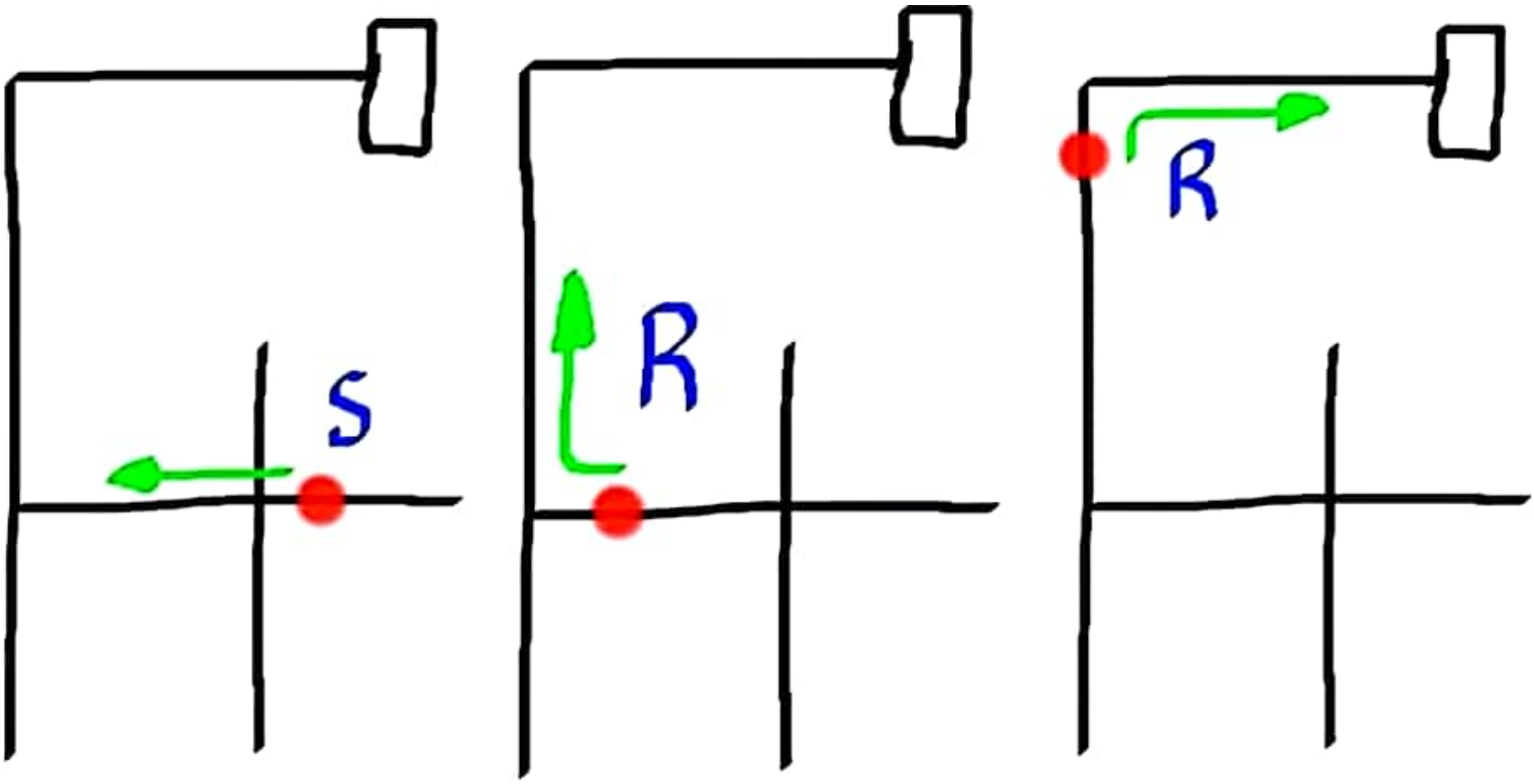
Path = LBLLB



Path = LBLLBS



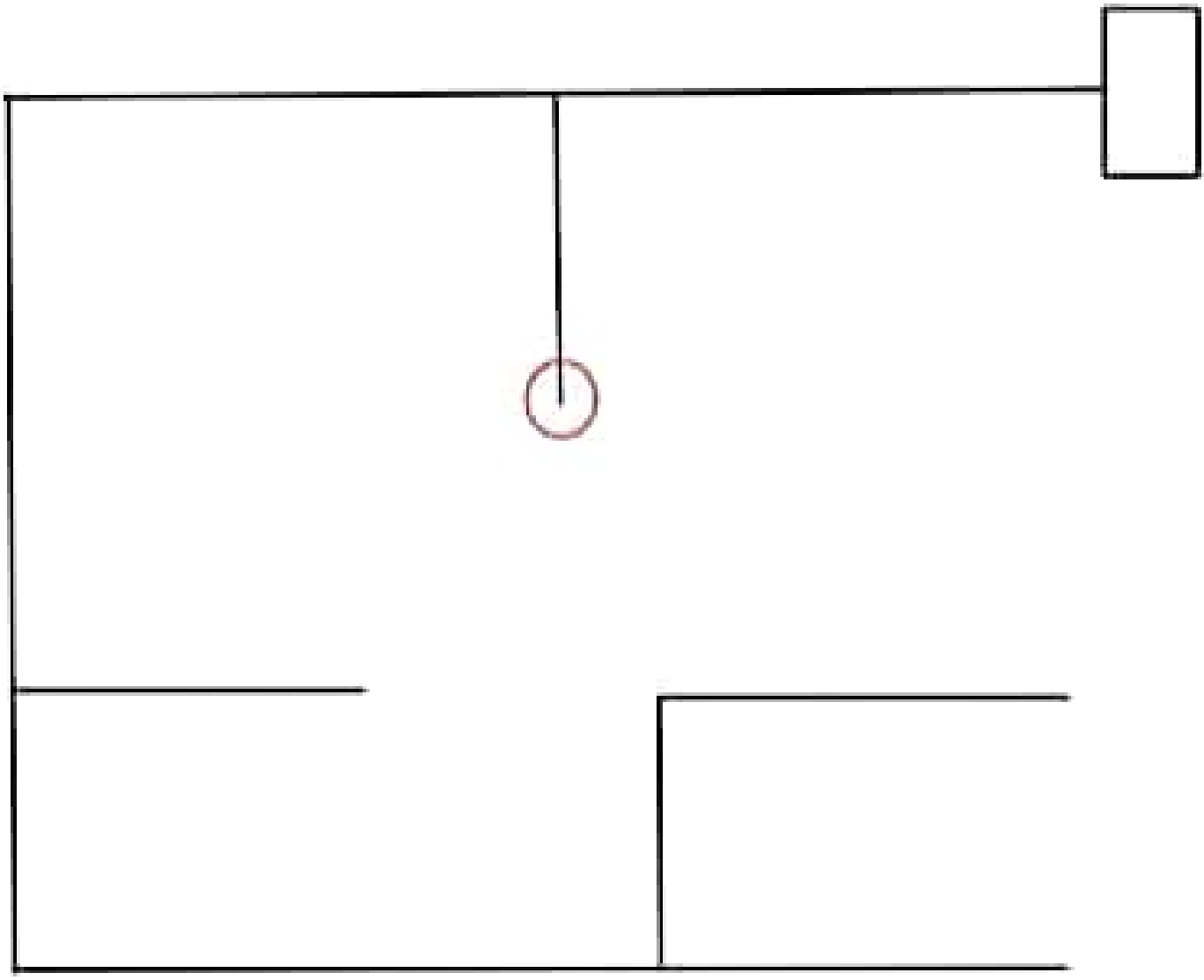
Final Path = LBLLBSR



Path = S

Path = SR

Final Path = SRS



Here is another example.

Using the left hand on the wall algorithm, here is the path the robot would take:
LLLBLLLRBLLBSRSRS

Now here is the process of shortening that path:
LL(LBL = S)LL(RBL = B)(LBS = R)RSRS

The new path would be:
LLSLLBRRSRS

Continue shortening it until all the "B"s are gone:
LLSL(LBR = B)RSRS

The new path would be:
LLSLBRSRS

Continue shortening it:

LLS(LBR = B)SRS

The new path would be:

LLSBSRS

Continue shortening it:

LL(SBS = B)RS

The new path would be:

LLBRS

Continue shortening it:

L(LBR = B)S

The new path would be:

LBS

Continue shortening it:

LL(SBS = B)RS

The new path would be:

LLBRS

Continue shortening it:

L(LBR = B)S

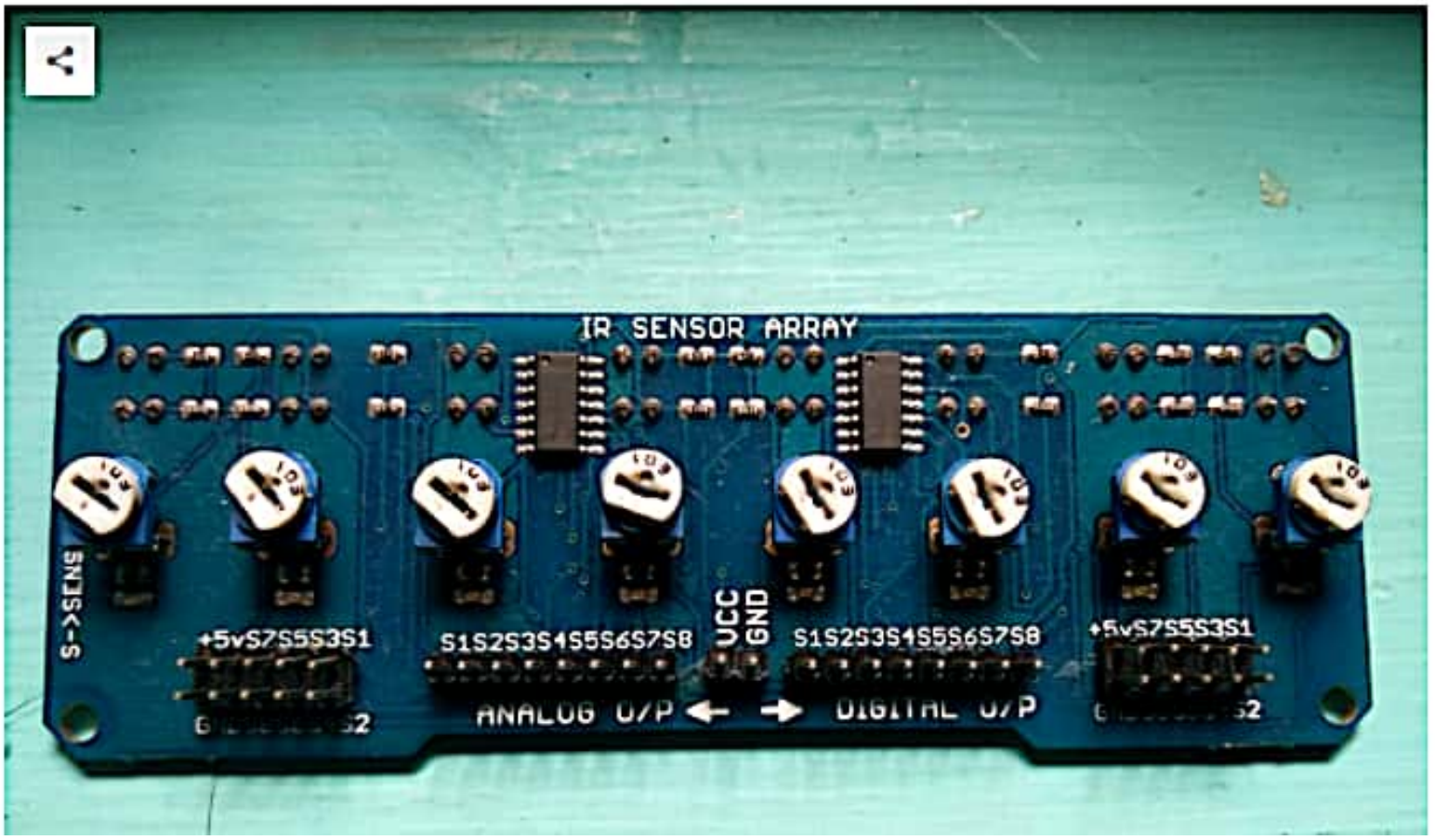
The new path would be:

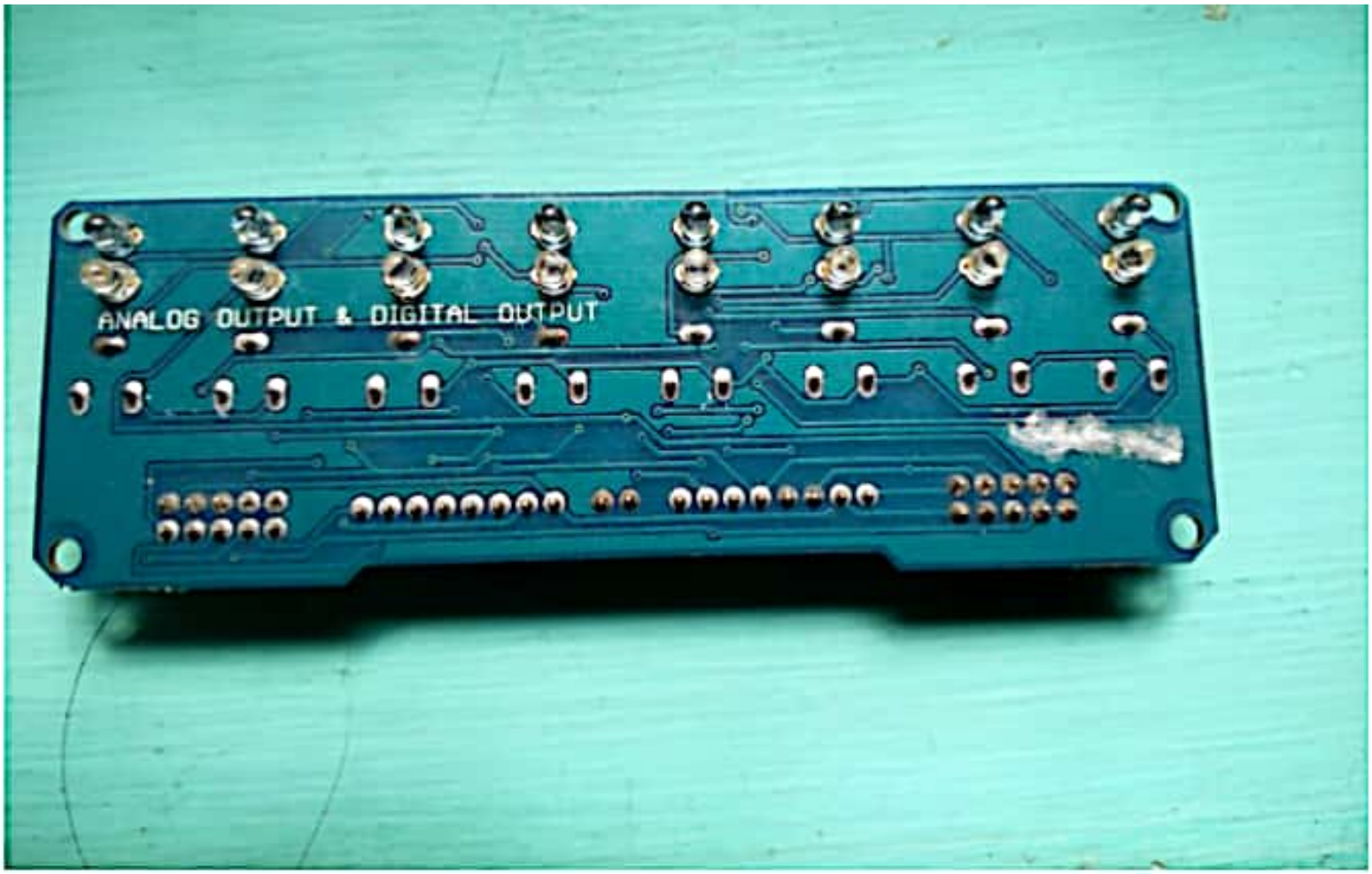
LBS

The final path is:

LBS = R

if your line follower gets some extra eyes. You could either add the number of individual IR Sensors, which is quite messy and power consuming, or you could buy or build an array of sensors in a single board. This can be called as an **IR Sensor Array**.





An IR Array board consist of 4-9 pairs of IR Transmitters and Receivers arranged in a line. Most of the array boards will be having a potentiometer for each sensor to control the sensitivity of the sensor. Also in most cases, you will be having provision to extract digital value as well as analog values from the sensors. These analog values can be used to calculate error levels while using the PID Algorithm.

It will be having 2 pins – One for VCC (5V), GND (0V) and Vout (Output Voltage) from 8 sensors. Since we are using Arduino Nano, we can only using 6 of them. Connect them as follows.

- IR Sensor ——> Arduino
- 5V ——> 5Vout of Arduino
- GND ——> GND of Arduino
- Vout of IR Sensor 1 ——> A0 of Arduino
- Vout of IR Sensor 2 ——> A1 of Arduino
- Vout of IR Sensor 3 ——> A2 of Arduino
- Vout of IR Sensor 4 ——> A3 of Arduino
- Vout of IR Sensor 5 ——> A4 of Arduino
- Vout of IR Sensor 6 ——> A5 of Arduino

Upload the below code and see if it is working.

Code

```
void setup() {
 Serial.begin(9600);
}
void loop() {
 int s1 = digitalRead(A0);
 int s2 = digitalRead(A1);
 int s3 = digitalRead(A2);
 int s4 = digitalRead(A3);
 int s5 = digitalRead(A4);
 int s6 = digitalRead(A5);

Serial.print(s1);
 Serial.print(" -- ");
 Serial.print(s2);
 Serial.print(" -- ");
 Serial.print(s3);
 Serial.print(" -- ");
 Serial.print(s4);
 Serial.print(" -- ");
```

```
 Serial.print(s5);
 Serial.print(" -- ");
 Serial.print(s6);
 Serial.println("");
 Serial.println("");
 delay(100);
}
```

Once the code is uploaded, open the serial monitor and you will see output like this

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

1 -- 1 -- 1 -- 1 -- 1 -- 0

1 -- 1 -- 1 -- 1 -- 0 -- 0

1 -- 1 -- 1 -- 1 -- 0 -- 0

1 -- 1 -- 1 -- 0 -- 0 -- 0

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

```
0 -- 0 -- 0 -- 0 -- 0 -- 0

0 -- 0 -- 0 -- 0 -- 0 -- 0

0 -- 0 -- 0 -- 0 -- 0 -- 1

0 -- 0 -- 0 -- 1 -- 1 -- 1

0 -- 0 -- 1 -- 1 -- 1 -- 1

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

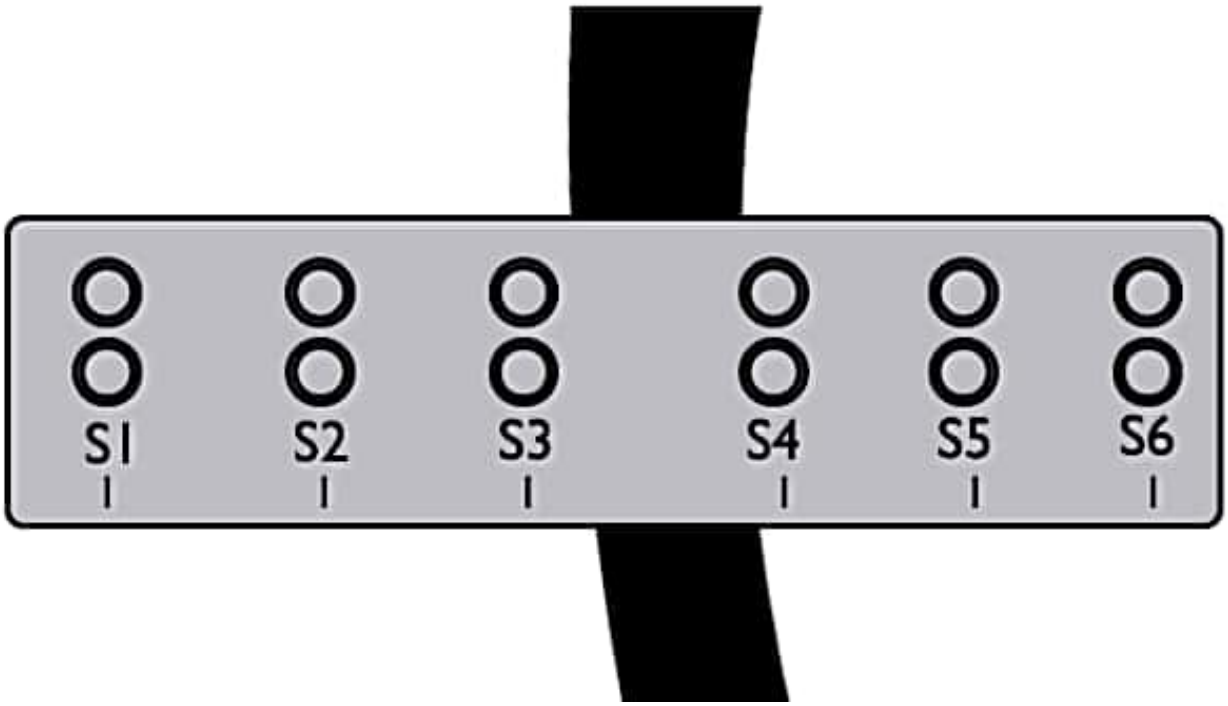
These values will be changing depending upon the surface where you are pointing this sensors.

Note: If you are using sensors with analog output, and use analogRead instead of digitalRead you will be getting values from 0 to 1023.

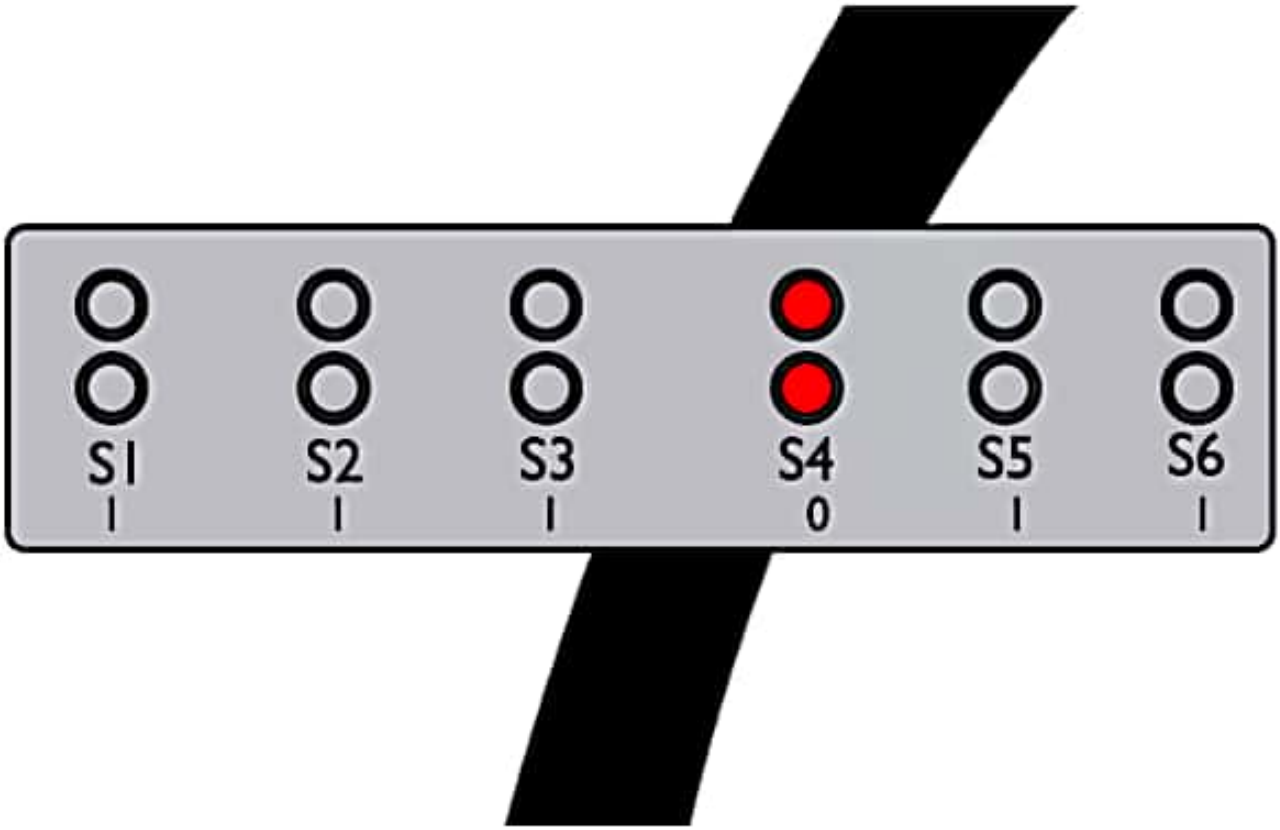
What do these outputs means?

Sensor output '1' means that most of the IR rays are reflected back to the sensor and the surface which is pointed by the sensor is white. And sensor output '0' means that most of the rays are absorbed and the surface is black.

When the arduino line follower is on track, the line will be in between the two sensors and will be pointing to the white surface. Which means the output of all the sensors will be 1. In that condition, the line following robot should move forward.



When there is a right turn on the track, the first sensor that crosses the line will be S4 and all other sensors will be pointing white surface. Thus the output of S4 will be 0 and all others will be 1. In this case out Arduino line follower should turn right.



When there is a Left turn on the track, the first sensor that crosses the line will be S3 and all other sensors will be pointing white surface. Thus the output of S3 will be 0 and all others will be 1. In this case out Arduino line follower should turn left.