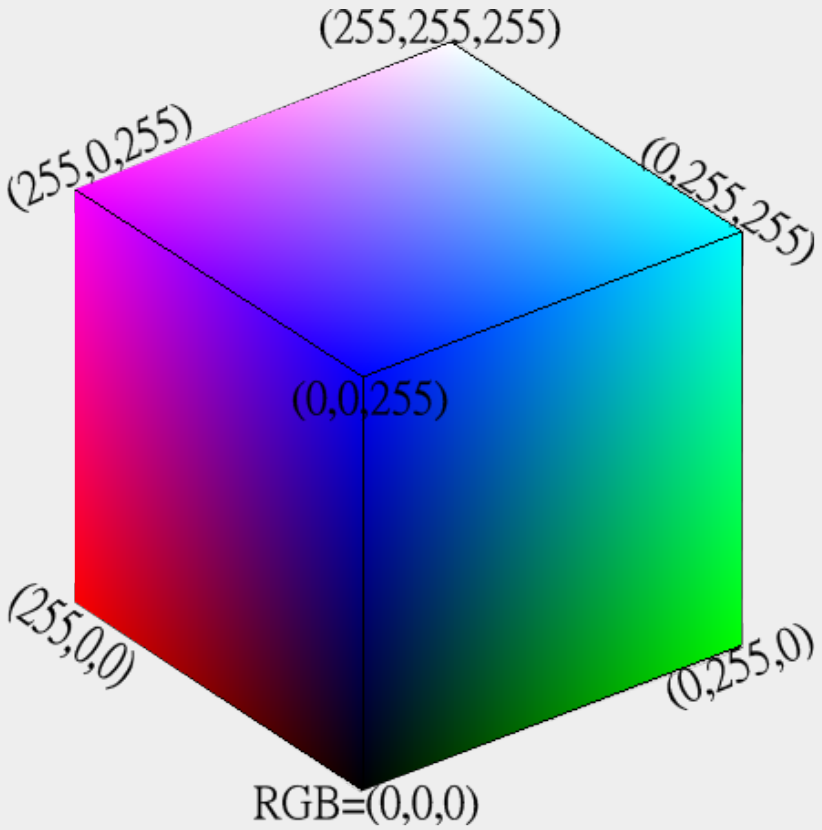
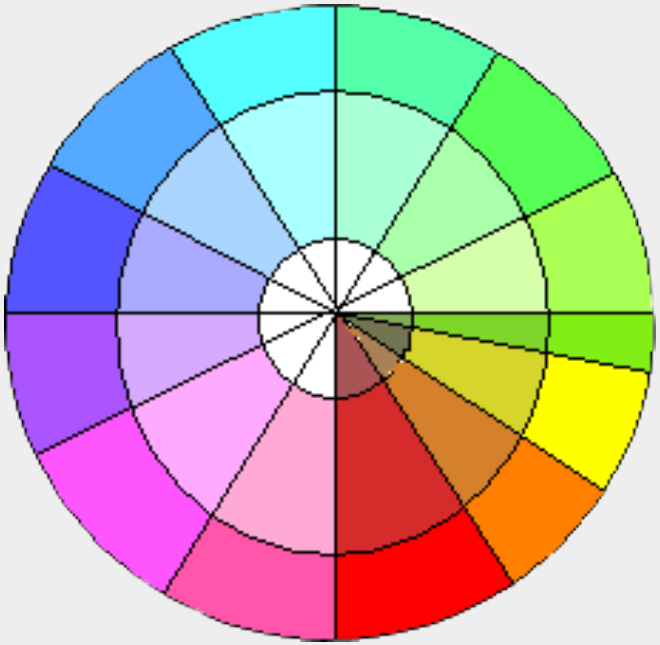
Monster Truck

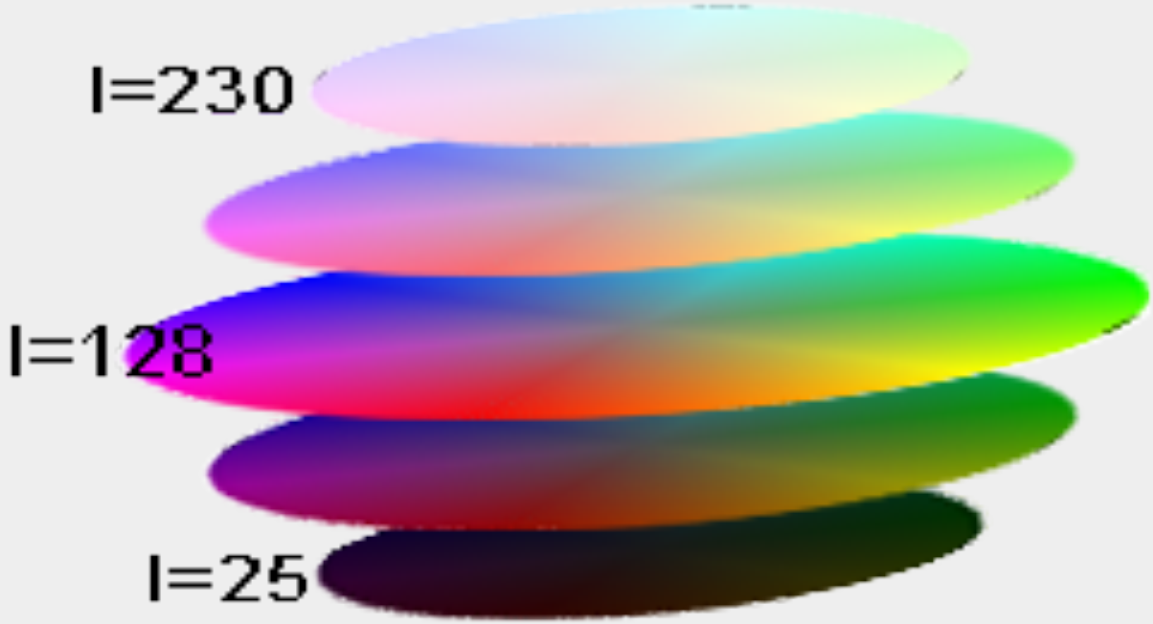
Dr. D.J. Lee, Professor
Director of Robotic Vision Laboratory
Electrical and Computer Engineering
Brigham Young University
Provo, Utah 84003
http://www.ee.byu.edu/faculty/djlee

RGB ColorSpace



HSI Colorspace





- Imgproc.cvtColor(mRgba, mHsv, Imgproc.COLOR_BGR2HSV, 3);

- Core.extractChannel(mRgba, mIntermediateMat, n);
  - Red: n=0, Green: n=1, Blue: n=2
  - Hue: n=0, Saturation: n=1, Intensity: n=2

- Imgproc.threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

- **THRESH_TRUNC**

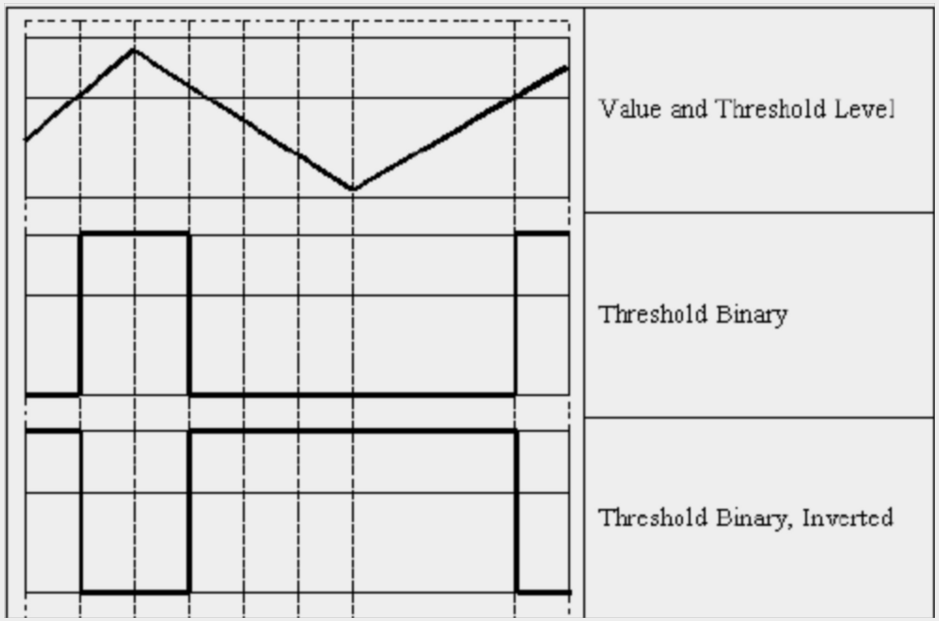
$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

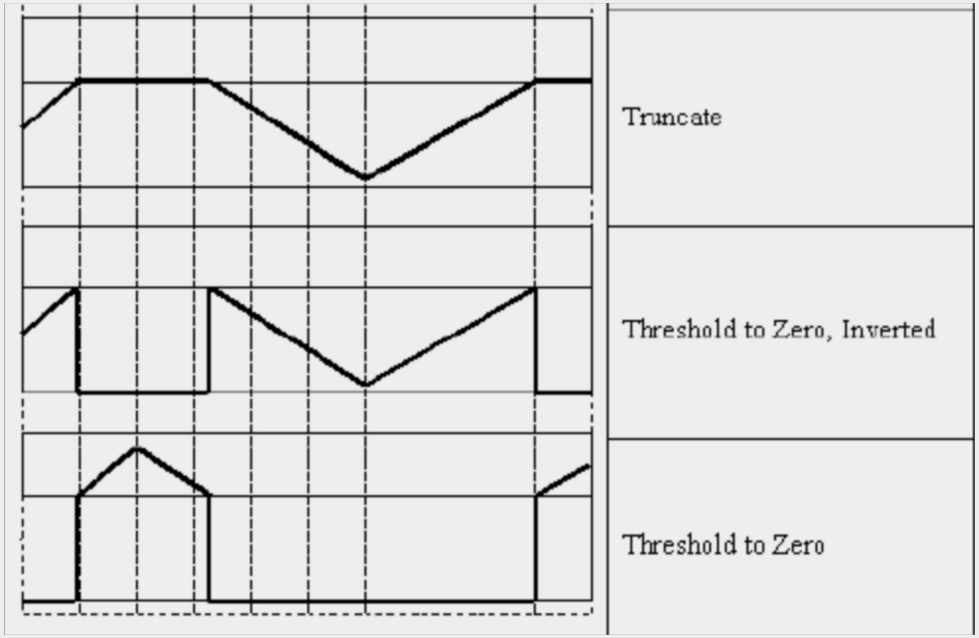
- **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$


Value and Threshold Level
Threshold Binary
Threshold Binary, Inverted


Truncate
Threshold to Zero, Inverted
Threshold to Zero

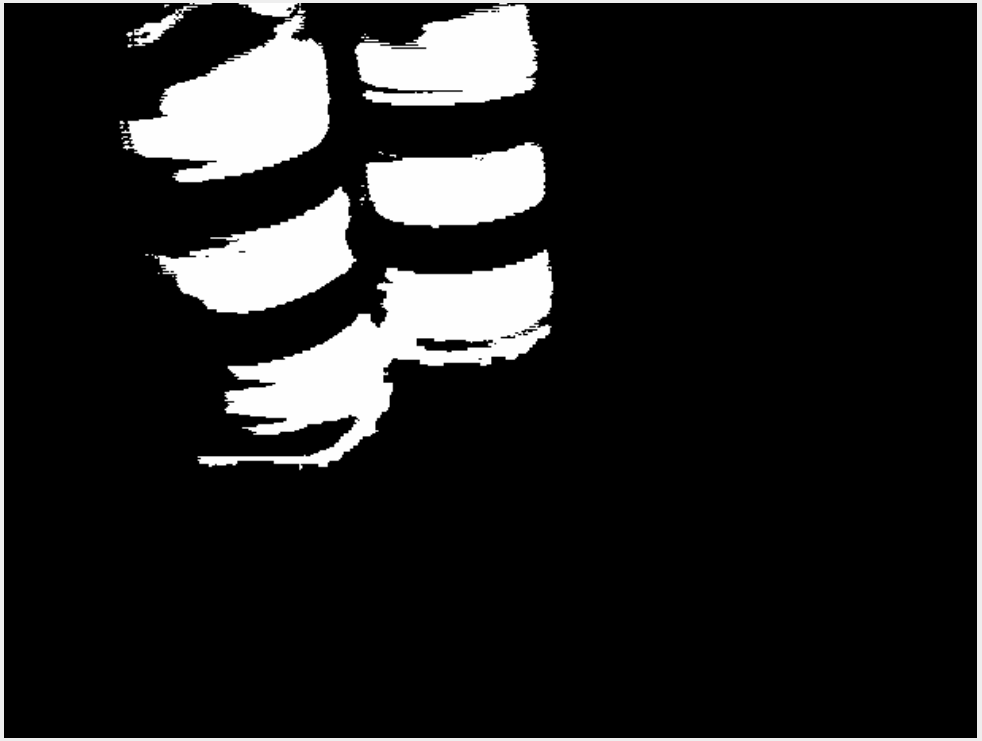
Original and Hue

RGB ➔ HSV ➔ Extract H ➔ Binarize H This is used as a mask.

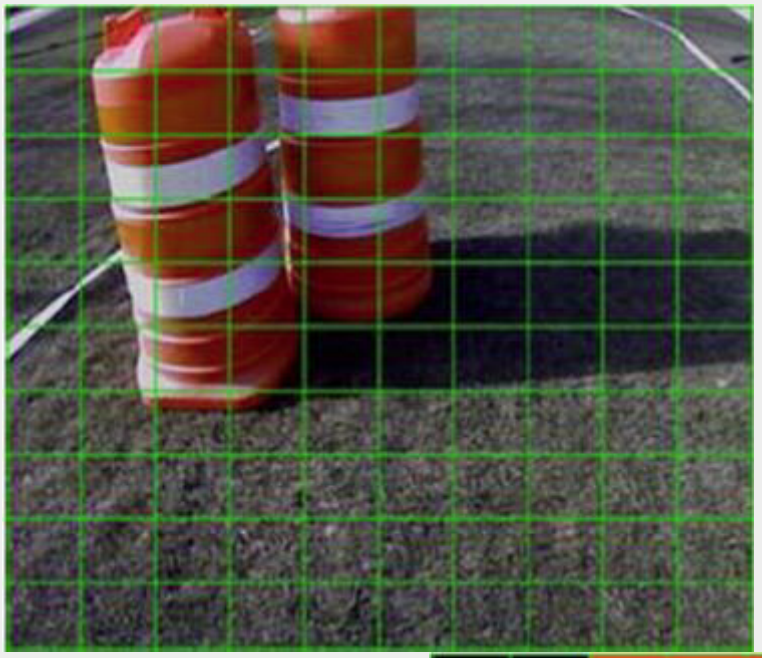
White Lines

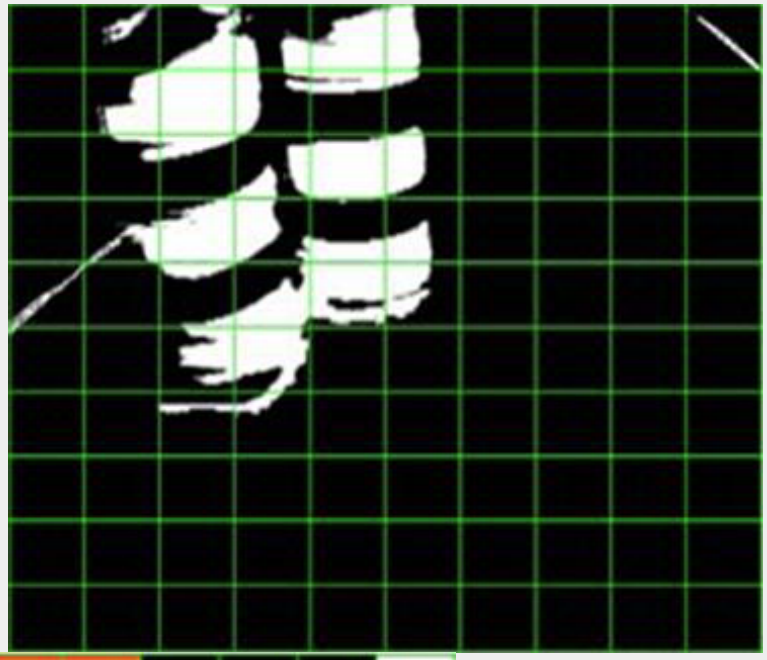
AND the mask with the hue channel or a selected channel (for example Blue) to isolated the lines using Core.bitwise_and(Mat src1, Mat src2, Mat dst)

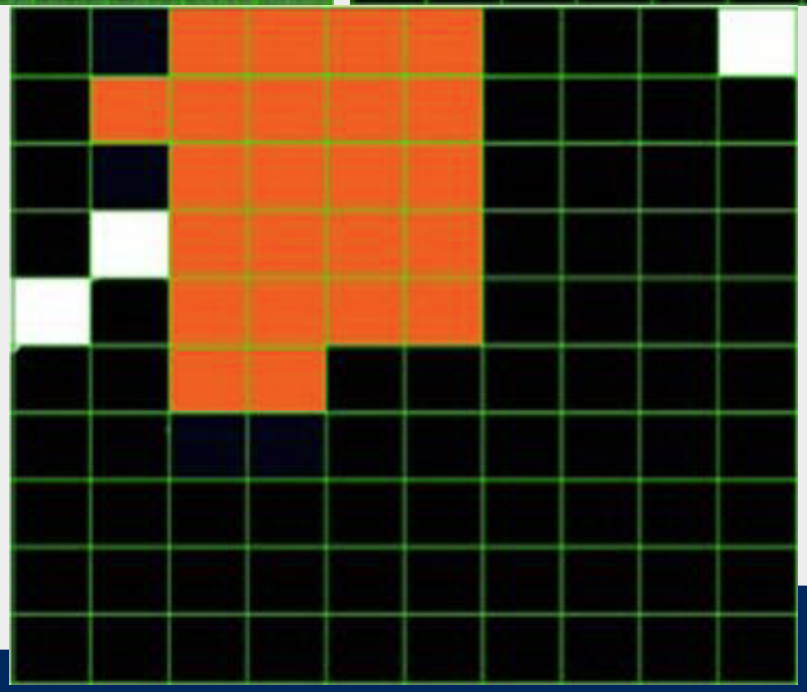
Obstacle



AND the mask with the hue channel or a selected channel (for example Red) to isolated the obstacle using Core.bitwise_and(Mat src1, Mat src2, Mat dst). Same thing can be done for the ramp in the green channel.

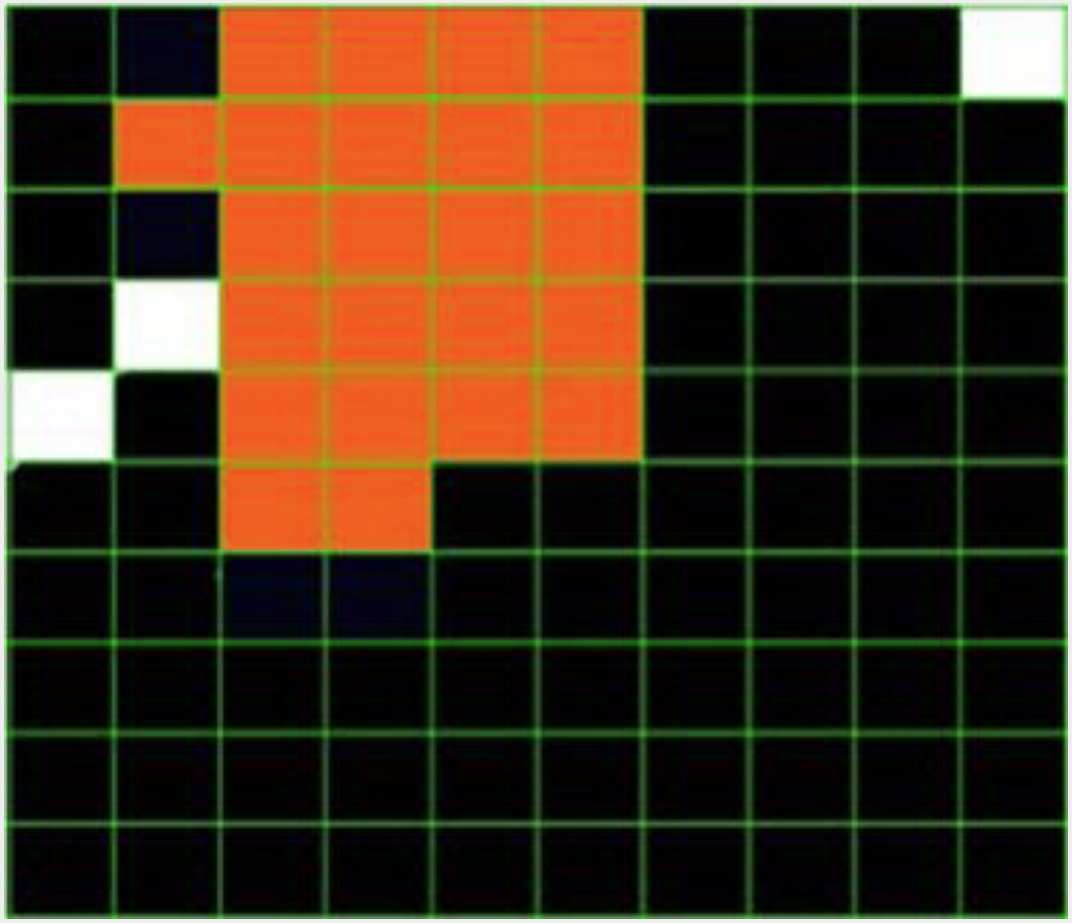




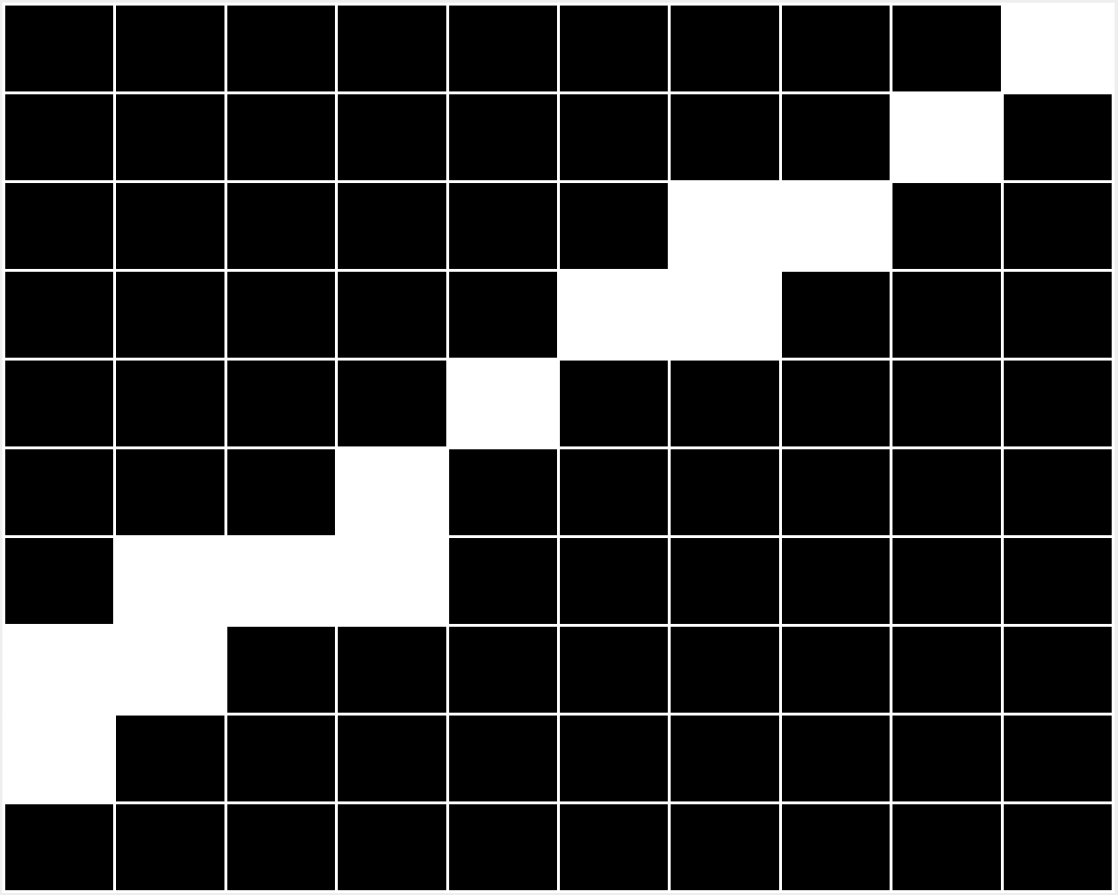


Occupancy Grid

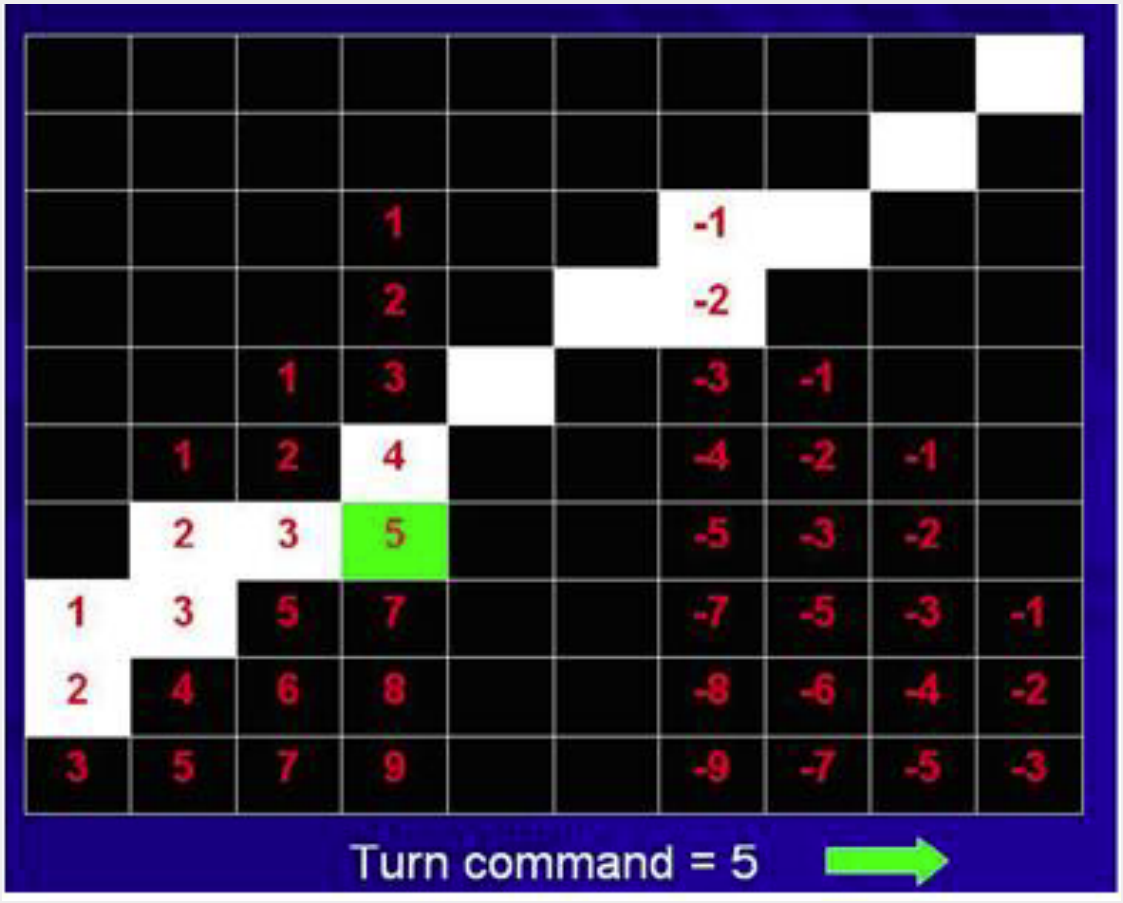
Occupancy Grid & Reactive Control



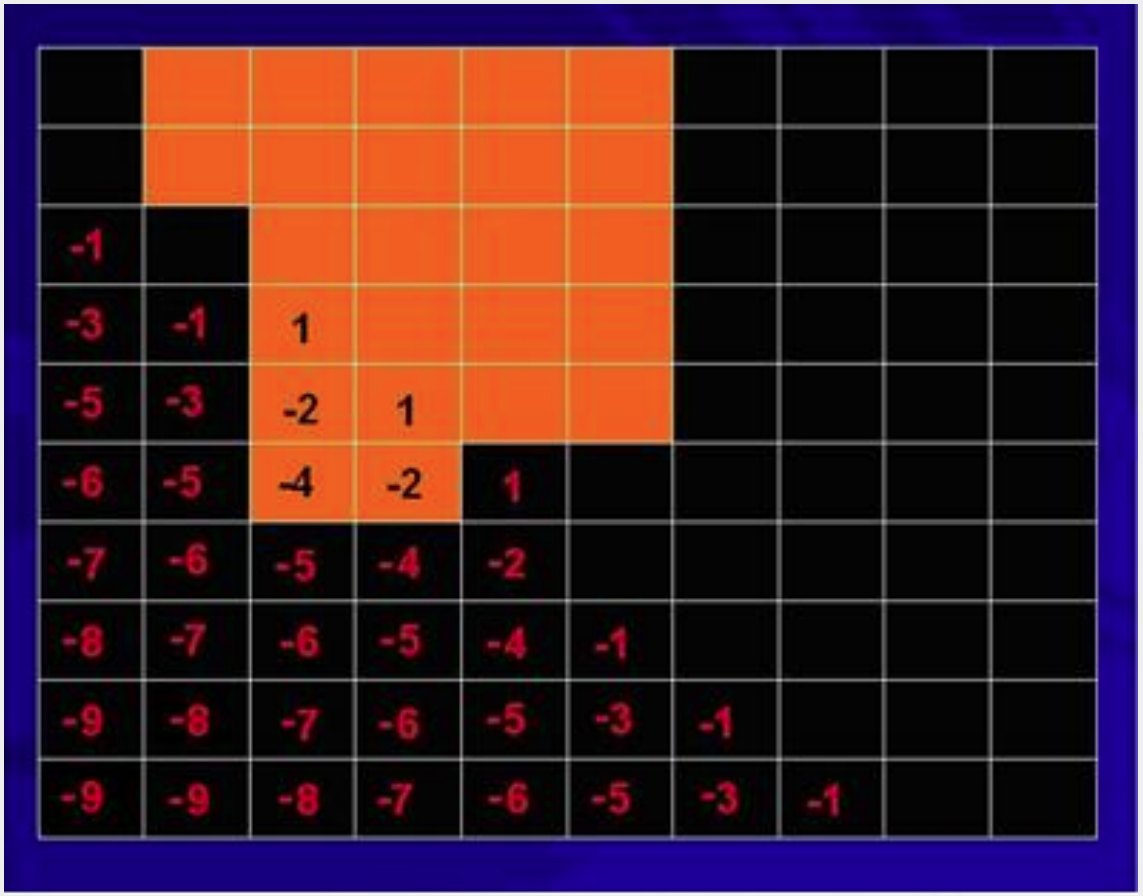
Example White Line



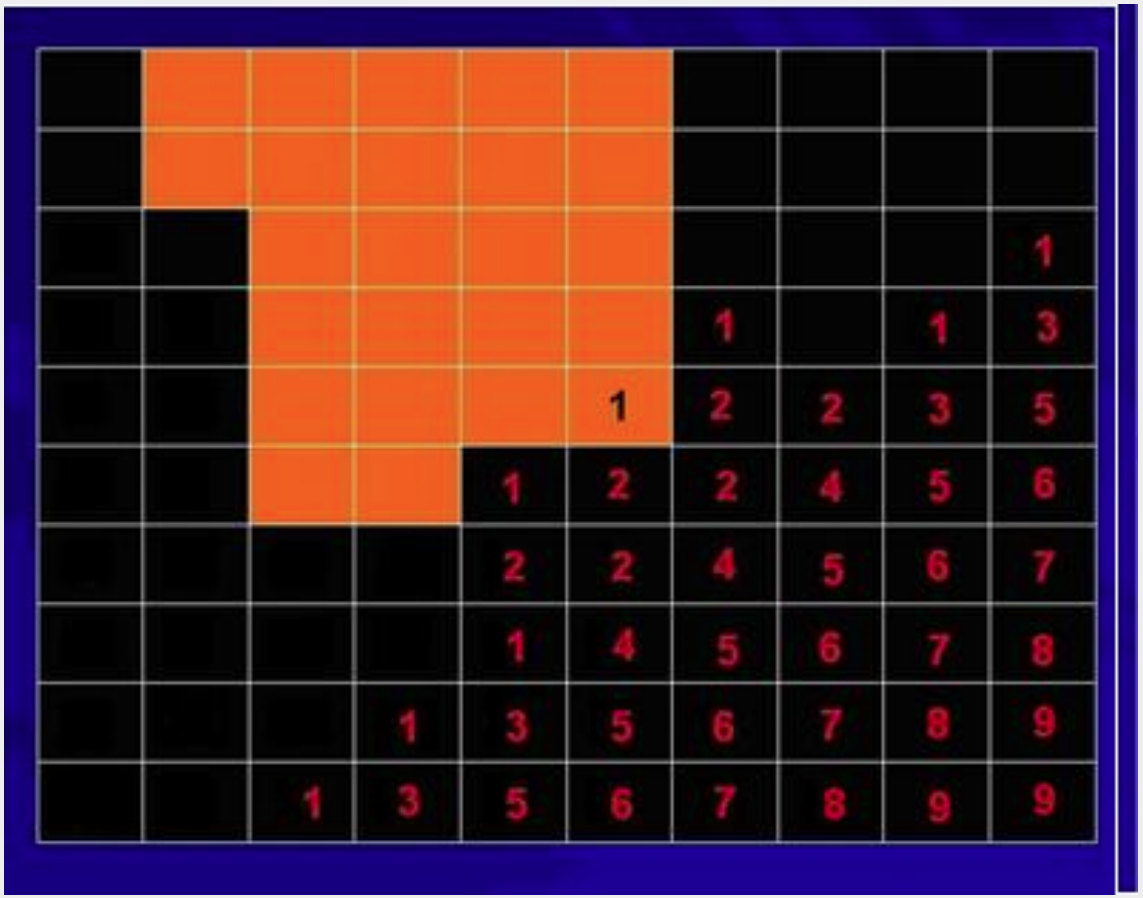
White Line Turn Matrix



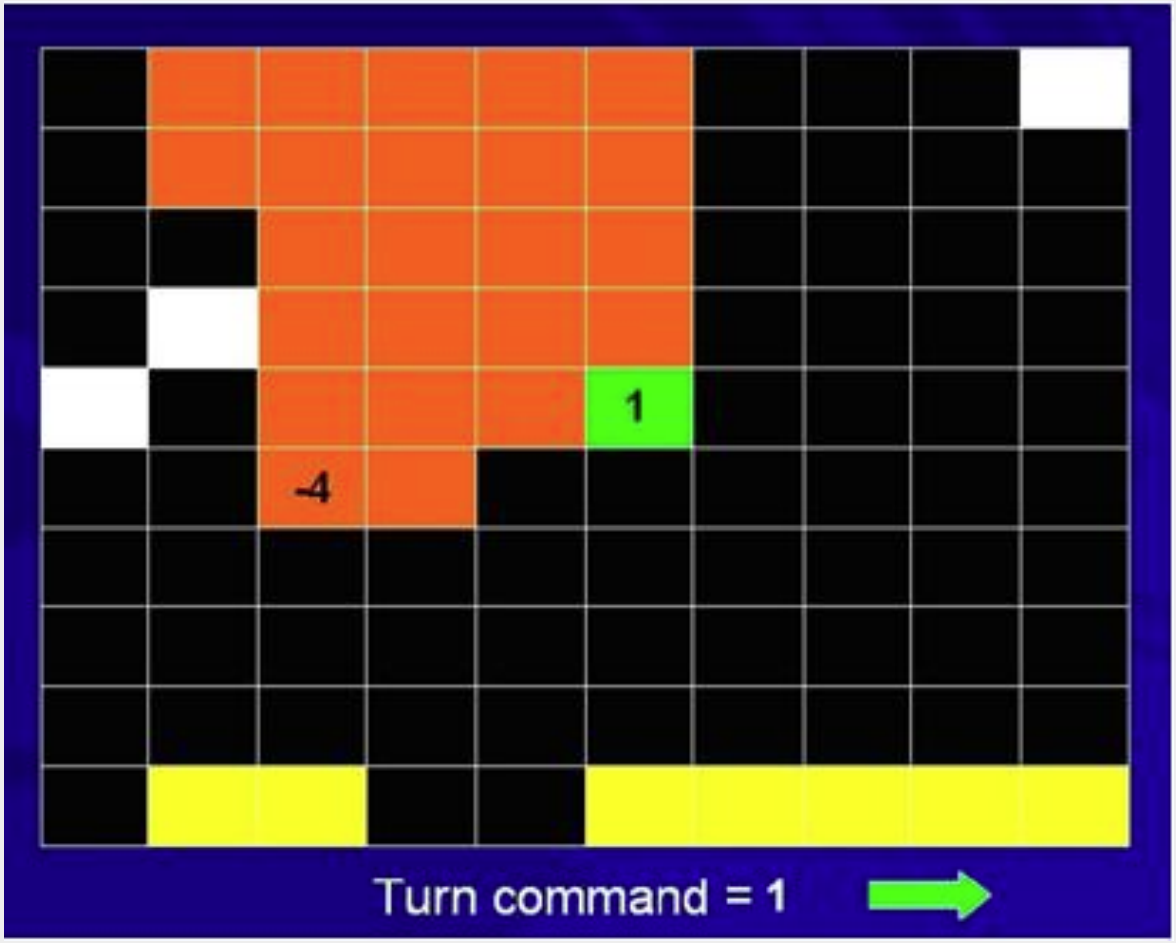
Obstacle Turn Left Matrix



Obstacle Turn Right Matrix



Which Side is Safer?

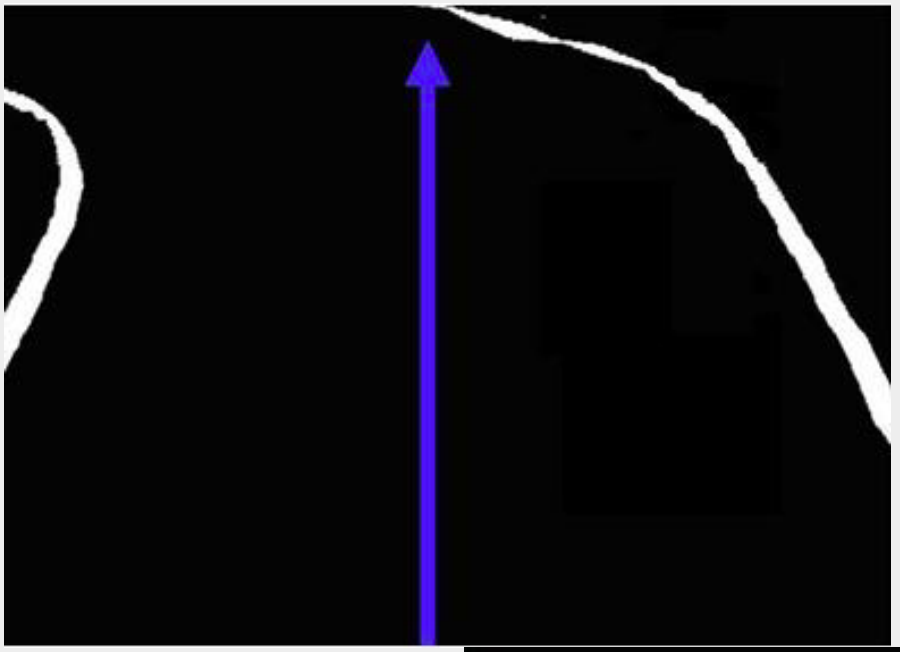


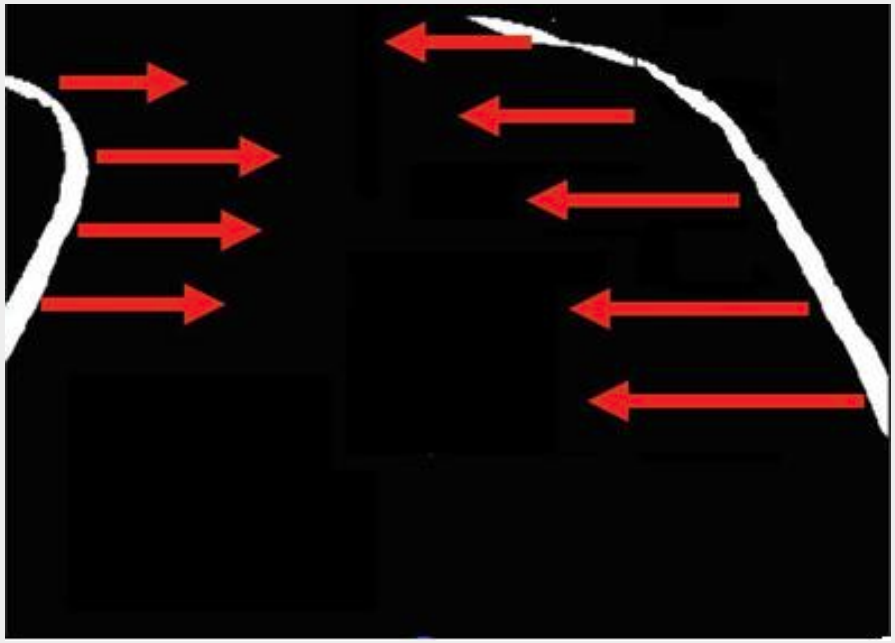
- To modify the command to avoid the orange obstacles, gaps between obstacle and lines are detected and compared to find the largest gap.
- If there is no gap large enough to move through in the current frame, the robot turns towards the most likely position of the gap until that gap is found.

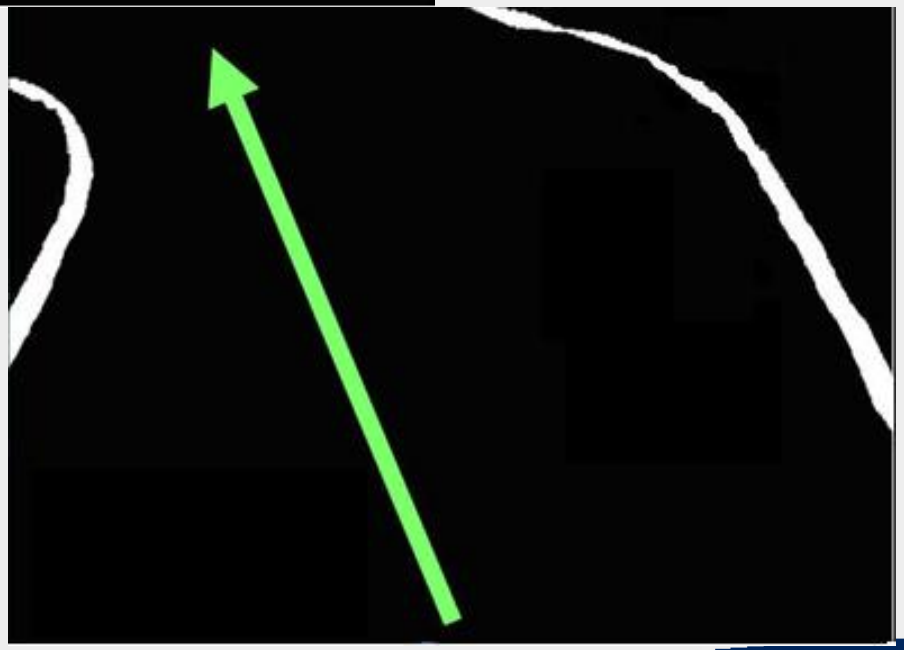
- After one or more gaps large enough to move through are found, two matrices like that used for the white line command are used to determine the strength of command needed to maneuver around the obstacles.
- One matrix isused for determining a maneuver to the left of the obstacles and one for maneuvering to the right of the obstacles.

- Then the numbers of squares between obstacle and line in the occupancy grid are used to compare relative sizes of gaps. When multiple gaps of the same size are found then the one that required the smallest command is chosen.
- Due to the nature of the occupancy grid many times the smallest turn command from the matrices and the largest gap corresponded resulting in the same command.

- The command given using the white lines is then compared to the command given by the orange obstacles.
- When the directions of the command are the same then the strongest command is returned to the robot. When the directions of command are different the orange command is given precedence.



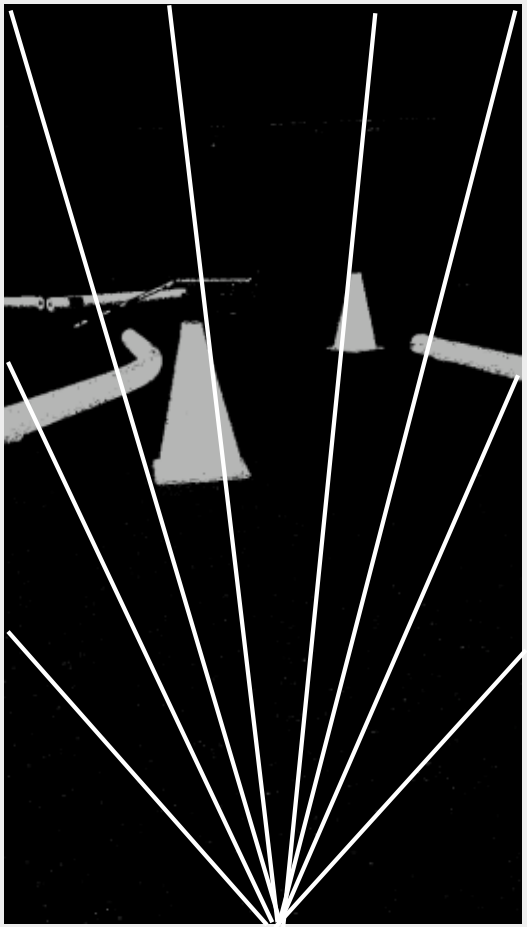




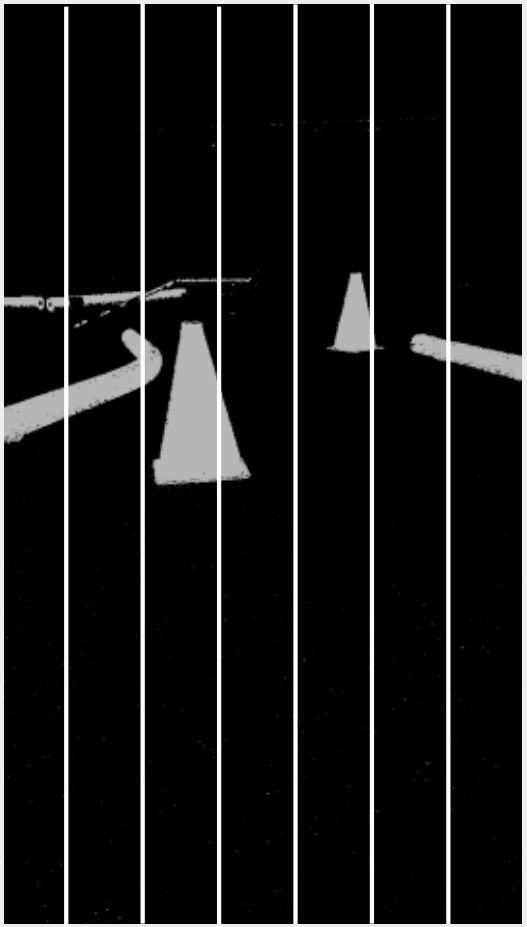
Priority Matrix

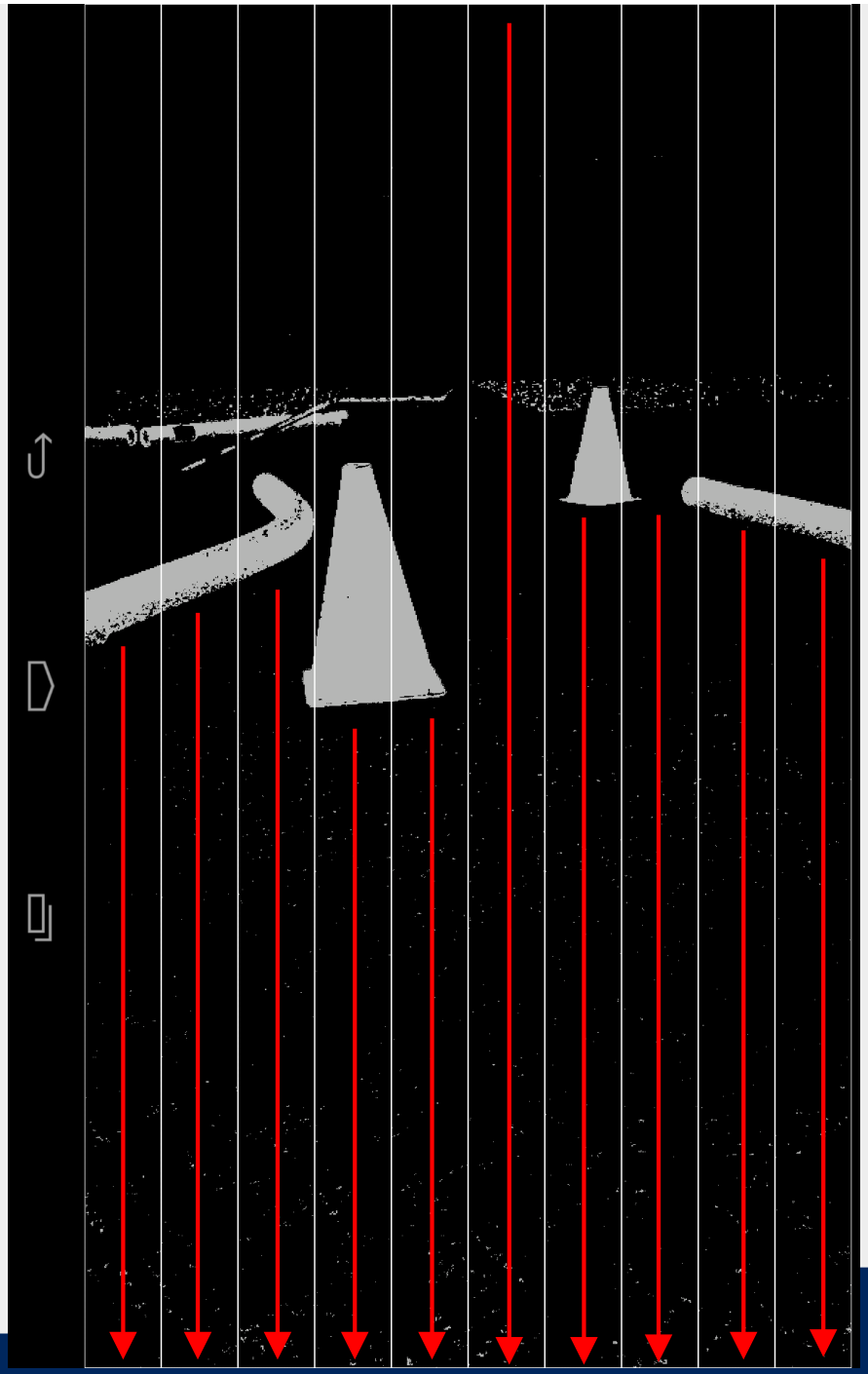
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
6	4	10	4	5
5	4	7	10	10
10	8	9	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10

V Decision

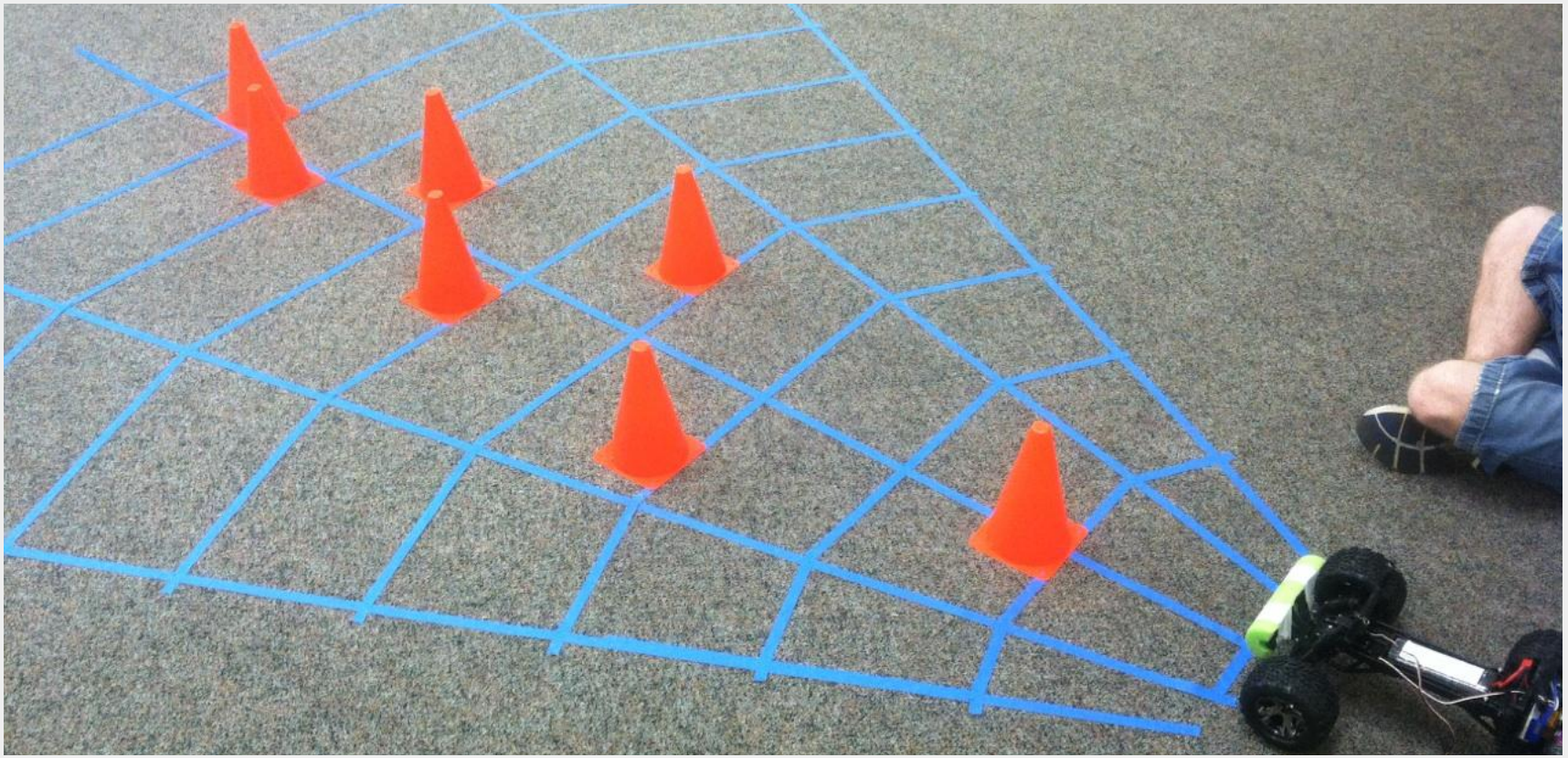


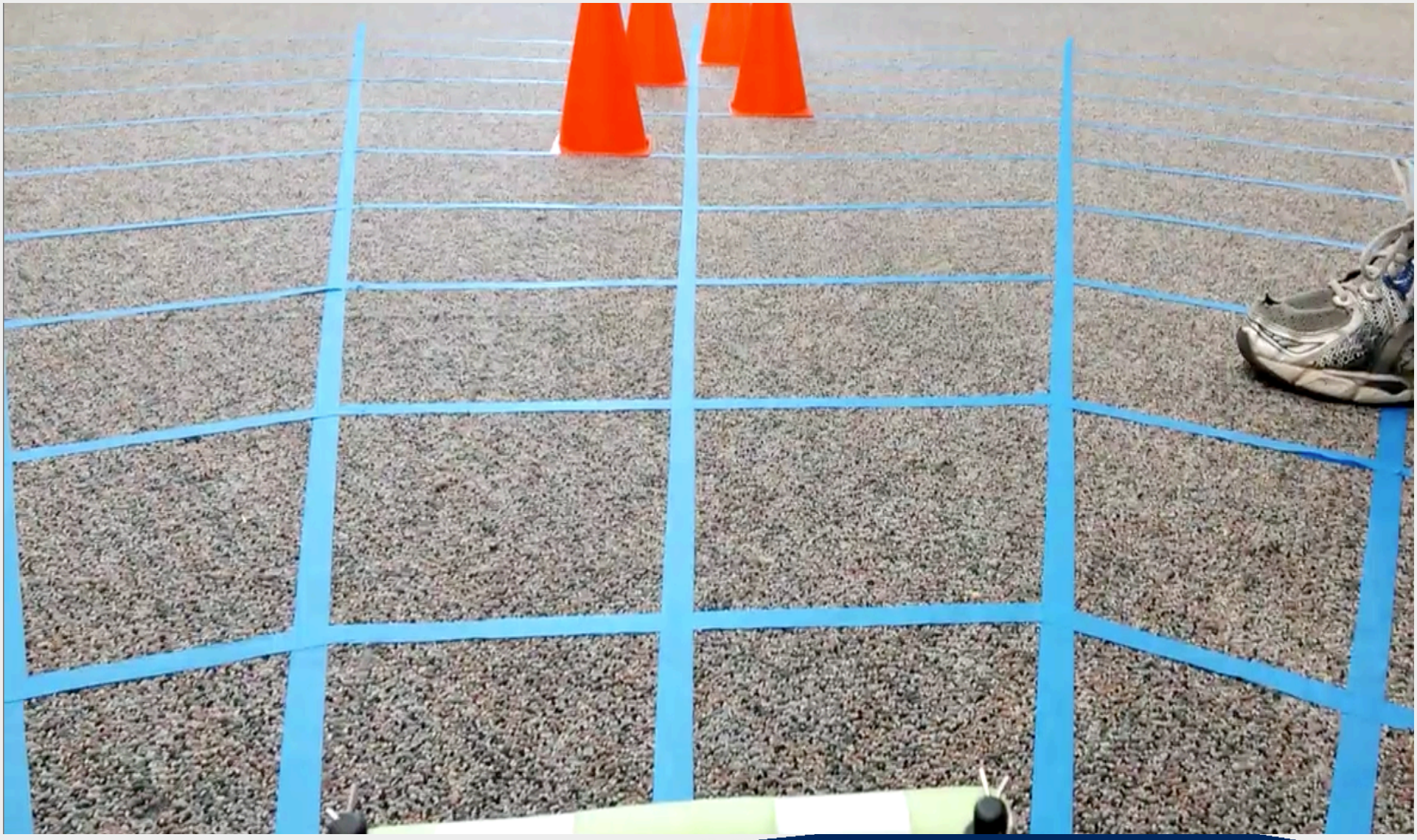
Column Decision



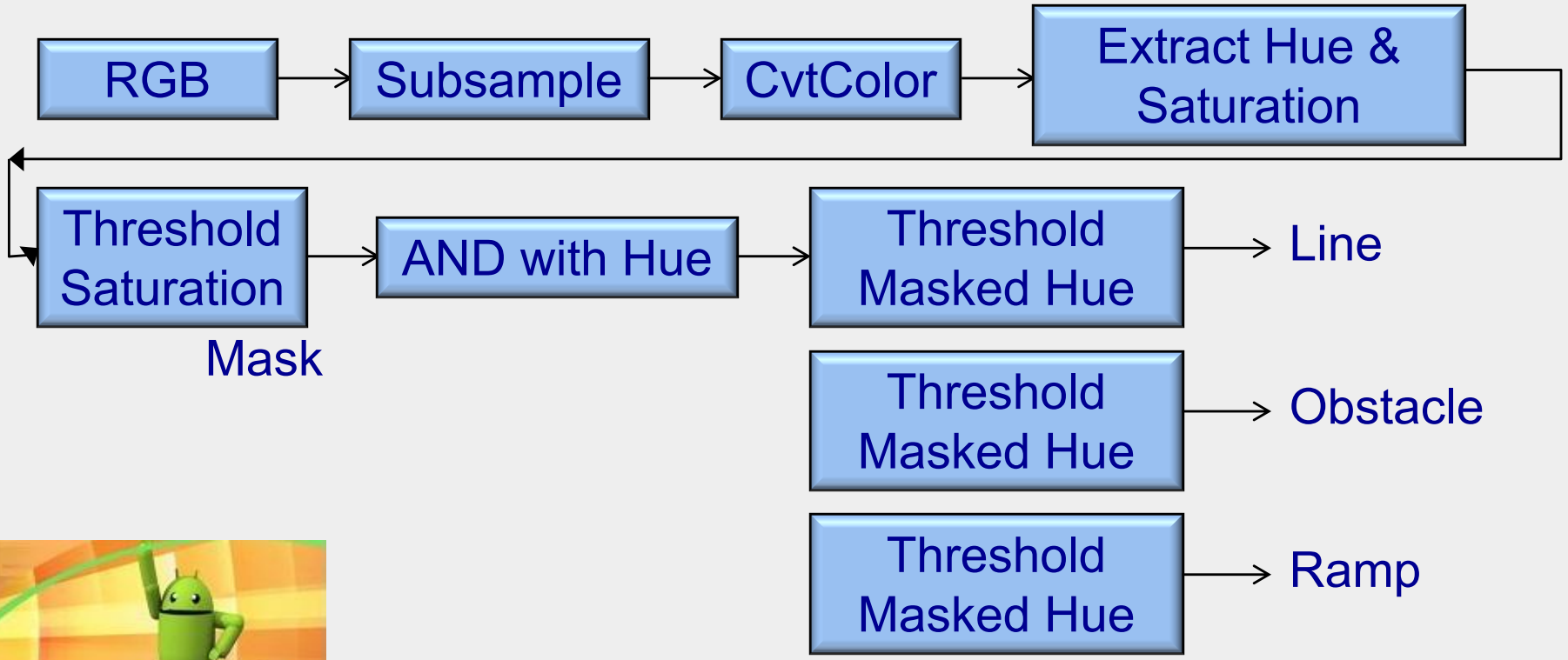


Grid System





- Do we need to process the entire image?
- Of course not.



resize(Mat src, Mat dst, Size dsize, double fx, double fy, int interpolation)

```
Size zero;
zero = new Size(0.0, 0.0);
```

- src – Source image.
- dst – Destination image.
- dsize – he destination image size. If it is zero, then it is computed as: dsize = Size(round(fx*src.cols), round(fy*src.rows)). Either dsize or both fx or fy must be non-zero.
- fx – The scale factor along the horizontal axis. When 0, it is computed as (double)dsize.width/src.cols
- fy – The scale factor along the vertical axis. When 0, it is computed as (double)dsize.height/src.rows
- interpolation – The interpolation method: INTER NEAREST or INTER LINEAR

Is it Occupied?

```
Rect mRect;

int mCount = 0;

mRect.width = 100;
mRect.height = 100;
mRect.x = 150;
mRect.y = 200;

mSub = mIntermediateMat.submat(mRect);

mCount = Core.countNonZero(mSub);
```

If mCount >= xyz, then it is occupied