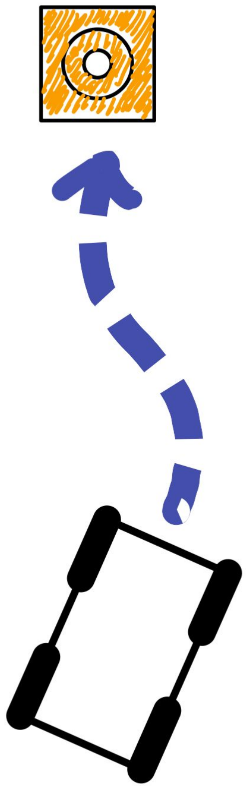
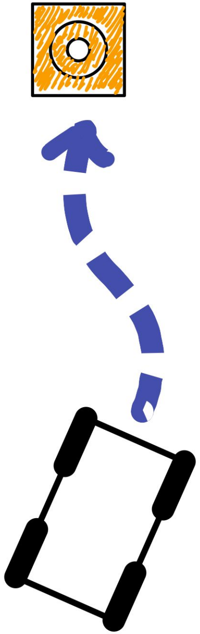
Visual-Servoing in Tesse

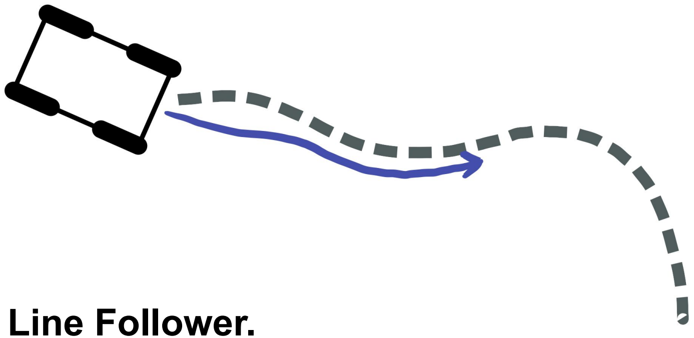
MIT Robotic Science + System (6.141) Team 11



Lab Goals + Overview



Parking Controller.

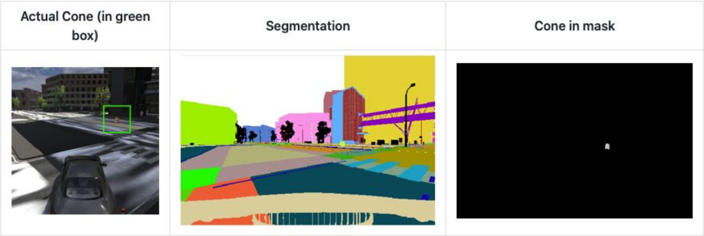


Line Follower.

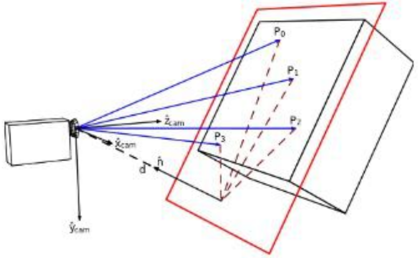
Implementing both of these w/ **computer vision on the TESSE car.

System Overview

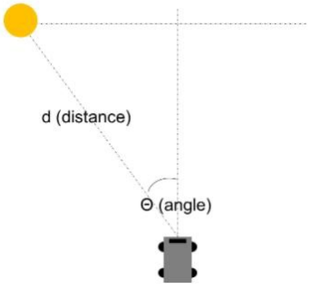
Camera Output



Cone/Line Detection w/ Color Segmentation

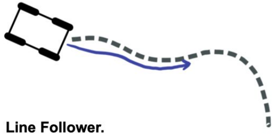


Homography + Transform



Controllers + Actuation

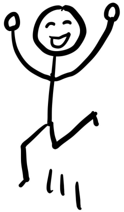
Parking Controller.



Line Follower.

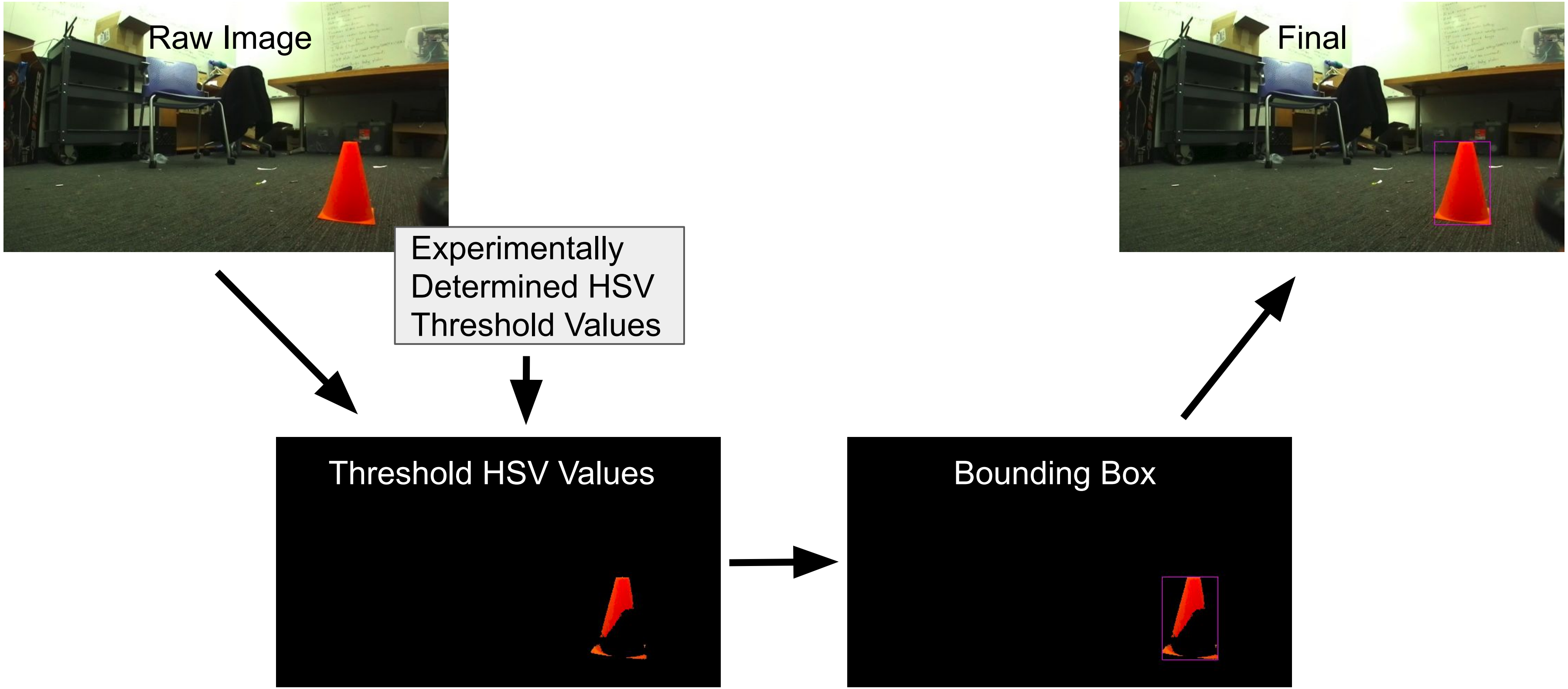
**Implementing both of these w/ computer vision on the TESSE car.

Car Executes Commands



Happy Company Management Ppl

Color Segmentation works well for Cone Detection



Color Segmentation works well for Cone Detection (mostly)

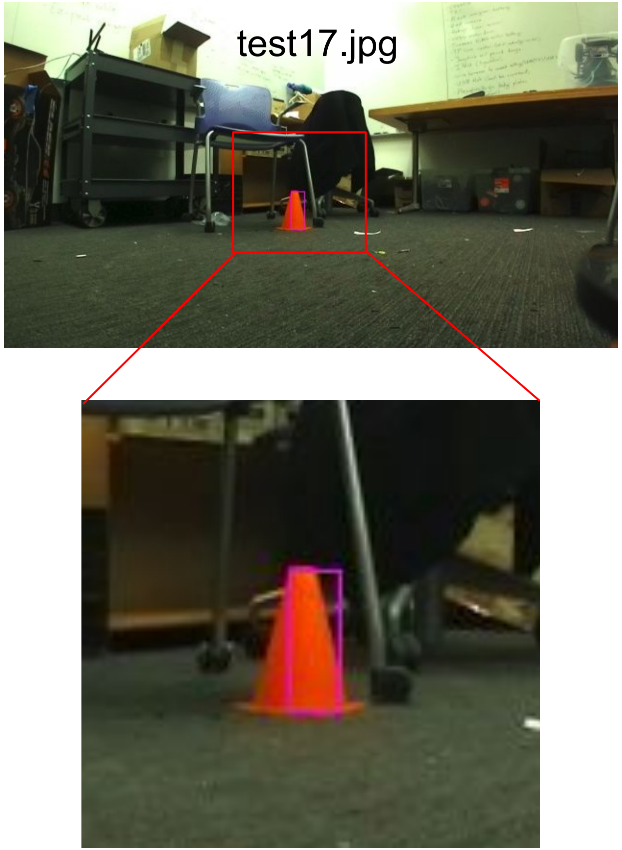
Average IOU: 0.74

Bottom 4 IOUs:

- 0.36, test17
- 0.42, test14
- 0.44, test15
- 0.49, test11

Hypothesis:

Poor lighting + Large distance from camera = Missed cone base



Homography can be used to map image coordinates to world frame

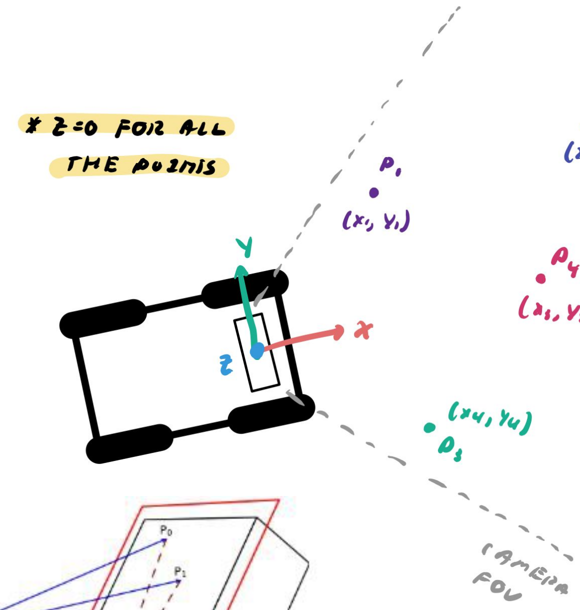
***homography helps us determine the position of object seen by the camera in the "world frame" as opposed to the "camera frame"

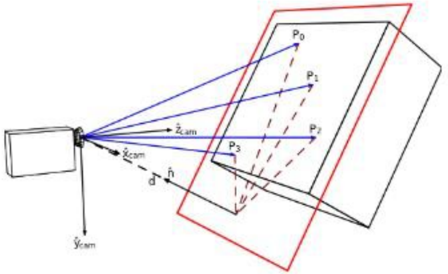
***we start by taking **points we know exist in the world** and are in the camera frame and using the intrinsic and extrinsic matrices, **map them to pixels** on the image

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D Image Coordinates | Intrinsic properties (Optical Centre, scaling) | Extrinsic properties (Camera Rotation and translation) | 3D World Coordinates

**choose four points to fully solve for the homography matrix (for a complete system of equations)





Homography/Camera Transformation

*****the intrinsic camera properties are specific to the camera** and acquired from the manufacturer/the camera itself

$$s\underbrace{\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}}_{\substack{\text{2D Image} \\ \text{Coordinates}}} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\substack{\text{Intrinsic properties} \\ \text{(Optical Centre, scaling)}}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{\substack{\text{Extrinsic properties} \\ \text{(Camera Rotation} \\ \text{and translation)}}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\substack{\text{3D World} \\ \text{Coordinates}}}$$

*****we calculate the extrinsic matrix based on the camera's mounting** location to the vehicle and the rotation of the camera's coordinate frame

**the rotation matrix comes from the camera's z axis pointing "forward," the translation comes from where the camera was mounted on the robot base

```
PSI = 0
THETA = -np.pi / 2
PHI = np.pi / 2

Rz = np.array([[np.cos(PSI), -np.sin(PSI), 0],
               [np.sin(PSI), np.cos(PSI), 0],
               [0, 0, 1]])

Ry = np.array([[np.cos(THETA), 0, np.sin(THETA)],
               [0, 1, 0],
               [-np.sin(THETA), 0, np.cos(THETA)]])

Rx = np.array([[1, 0, 0],
               [0, np.cos(PHI), -np.sin(PHI)],
               [0, np.sin(PHI), np.cos(PHI)]])

Rxyz = np.matmul(np.matmul(Rz, Ry), Rx)

# setup translation

T = np.array([[0.05], [1.03], [-1.5]])
EM = np.append(Rxyz, T, 1)
```

Homography/Camera Transformation

***we can then assume that **any point in the (x,y) frame of the car can be resolved from it's coordinates on the image** through a matrix called the homography matrix

***this matrix can be solved for using a system of equations (which is done automatically for us in OpenCV)

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

NOTE "S" NORMALIZATION FACTOR!!!

HOMOGRAPHY MATRIX "H"

```
PTS_GROUND_PLANE = np.array(
    [
        [PTS_GROUND_PLANE[0][0], PTS_GROUND_PLANE[0][1], 1],
        [PTS_GROUND_PLANE[1][0], PTS_GROUND_PLANE[1][1], 1],
        [PTS_GROUND_PLANE[2][0], PTS_GROUND_PLANE[2][1], 1],
        [PTS_GROUND_PLANE[3][0], PTS_GROUND_PLANE[3][1], 1],
    ]
)

PTS_IMAGE_PLANE = np.array(
    [
        [PTS_IMAGE_PLANE[0][0], PTS_IMAGE_PLANE[0][1], 1],
        [PTS_IMAGE_PLANE[1][0], PTS_IMAGE_PLANE[1][1], 1],
        [PTS_IMAGE_PLANE[2][0], PTS_IMAGE_PLANE[2][1], 1],
        [PTS_IMAGE_PLANE[3][0], PTS_IMAGE_PLANE[3][1], 1],
    ]
)

self.homography_matrix, err = cv2.findHomography(PTS_IMAGE_PLANE, PTS_GROUND_PLANE)

################################################################################
```

**now each pixel can be resolved to an x.y coordinate in the robot frame

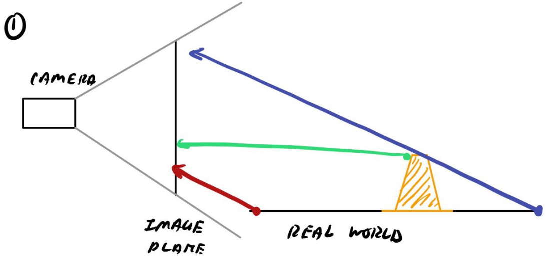
Some notes on the effectiveness of homography

***not a super technically-accurate way to describe this but homography is really like a **“feed-forward” transform of the camera data**

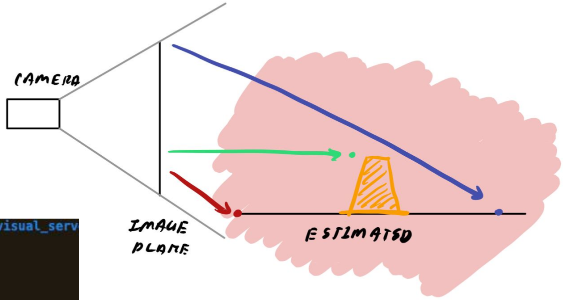
**essentially we just trust the physics of the camera are reliable, good enough but probably wouldn’t trust it to stop us from hitting a pedestrian (some verification might be nice)

**cannot extrapolate a third dimension here, also assumes camera is stable!

①



②

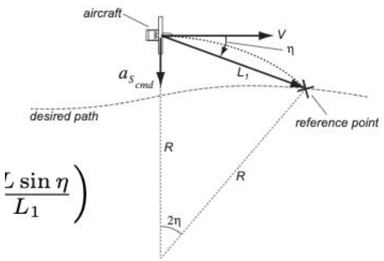


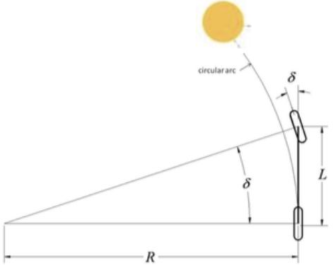
WROTE A TEST.PY
SEEMS ACCURATE
* NO GROUND TRUTH DATA *

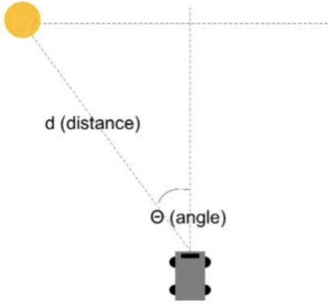
```
chip-core@chipcore-desktop:~/racecar_ws/src/visual_serv
[[ 2.5  1.   1. ]
 [ 2.5 -1.   1. ]
 [ 3.5  1.   1. ]
 [ 3.5 -1.   1. ]]
[[ 58.85345393 356.40099206   1.        ]
 [440.21460356 356.40099206   1.        ]
 [149.42672696 258.20049603   1.        ]
 [340.10730178 258.20049603   1.        ]]
[[-1.08855899e-20 -9.37500000e-03  2.72493744e-01]
 [ 6.43750037e-03 -3.12499950e-04 -1.49500010e+00]
 [-6.55841055e-21 -6.25000000e-03  1.00000000e+00]]
[[15.52864293]
 [-4.80571457]
 [ 1.        ]]
```

Pure Pursuit works well as a Parking Controller

How pure pursuit works







$$\delta = \tan^{-1}\left(\frac{2L\sin(\theta)}{d}\right)$$

Our Controller

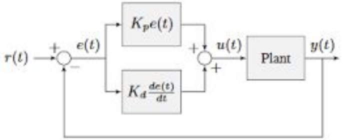
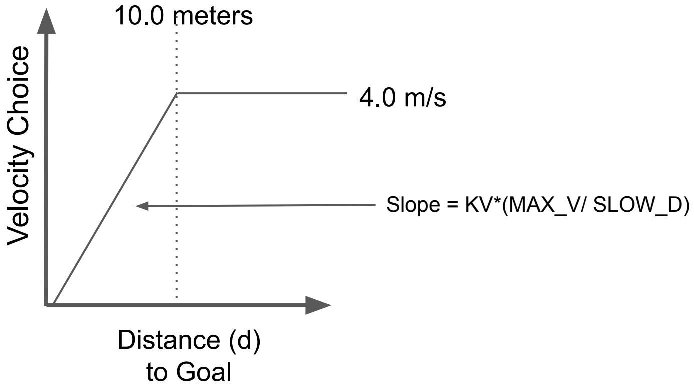


Figure 2.2: PD controller block diagram

```
curr_theta = -1.0 * np.arctan(self.WHEELBASE / R)
d_theta = self.KP * curr_theta + self.KD * (curr_theta - self.old_angle)
self.old_angle = d_theta
```



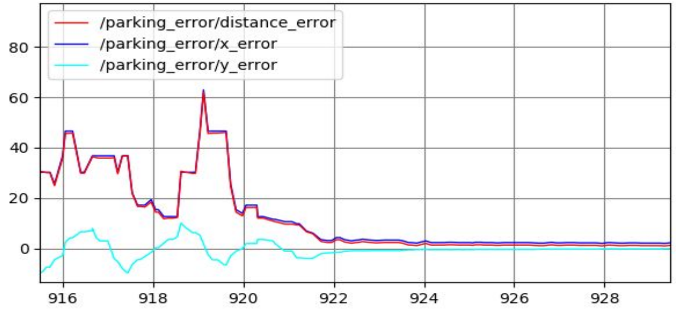
Parking Controller Demo

video

Ununed Controller

KP = 1.0
KD = 0 .3
VMAX = 4.0
KV = 1.0

Tuned Controller

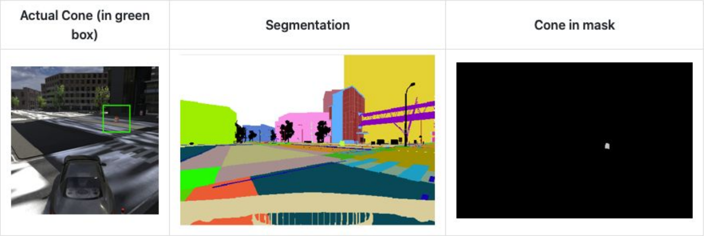


KP = 1.0
KD = 0 .6
VMAX = 4.0
KV = 1.0

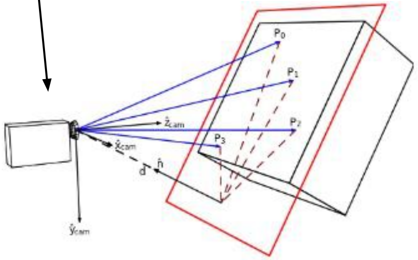
Line Detection + Following

Hough Line Transforms

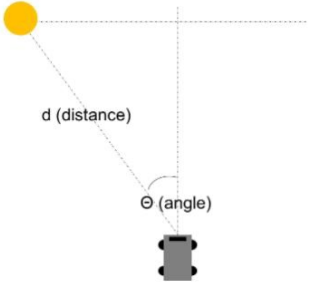
Camera Output



Cone/Line Detection w/ Color Segmentation

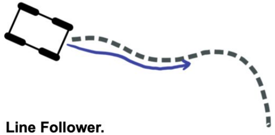


Homography + Transform



Controllers + Actuation

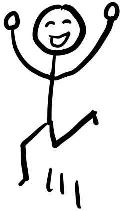
Parking Controller.



Line Follower.

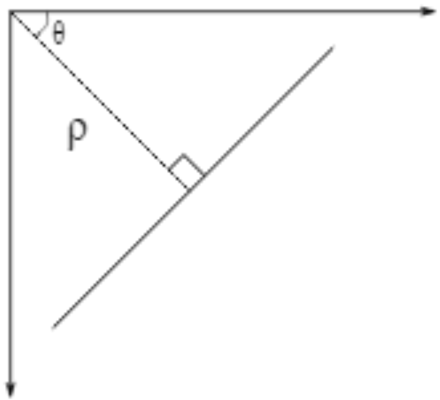
**Implementing both of these w/ computer vision on the TESSE car.

Car Executes Commands



Happy Company Management Ppl

How Hough Line Transforms Work



lines normally represented as

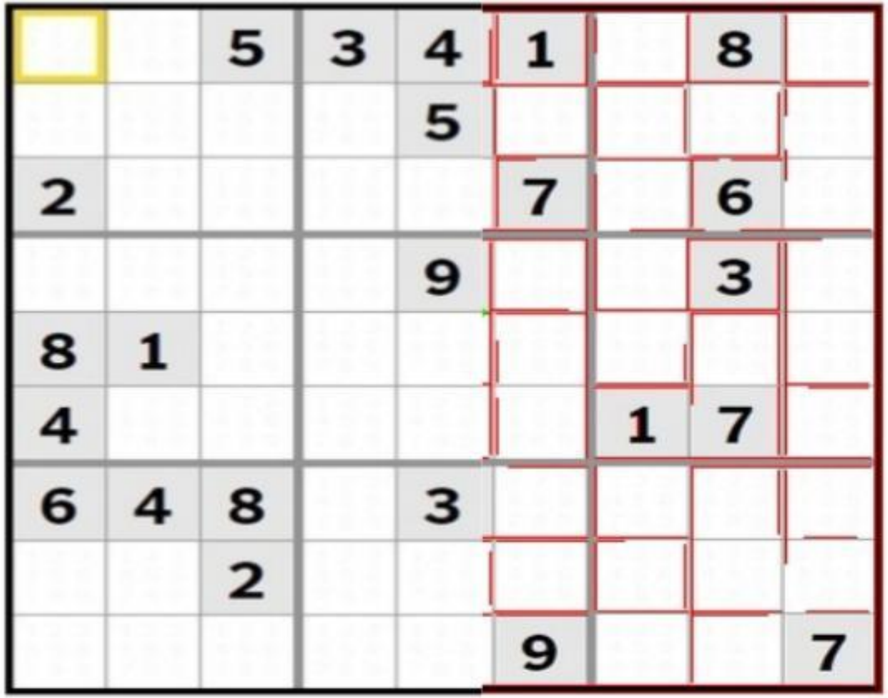
$y = m*x + b$

can now be represented as

$\rho = x*\cos(\theta) + y*\sin(\theta)$

successful lines fit more points in our images

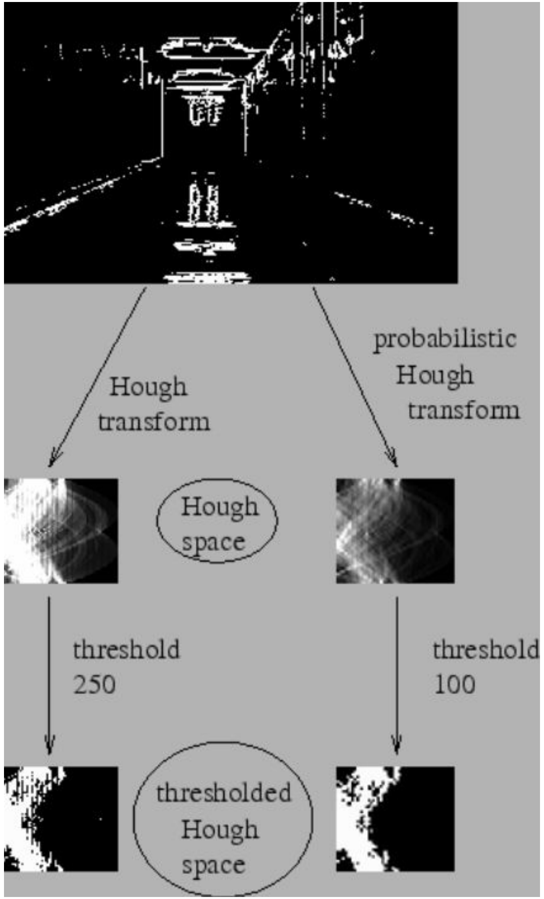
Without Hough Lines



With Hough Lines (Red)

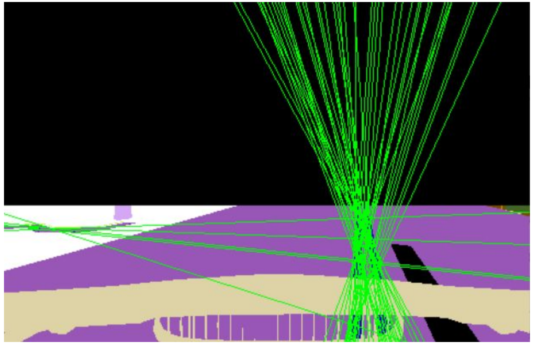
A Probabilistic Hough Line Transform approach was used

- Reduces necessary computation by just using a random subset of points
- Directly returns end points of hough lines rather than parameters



All Hough Lines are Averaged to Approximate the Line to Follow

1. Begin with many hough lines

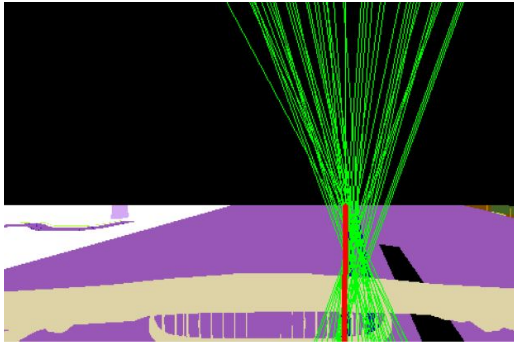


2. Average start points for all lines and end points for all lines to approximate the ends of the average line

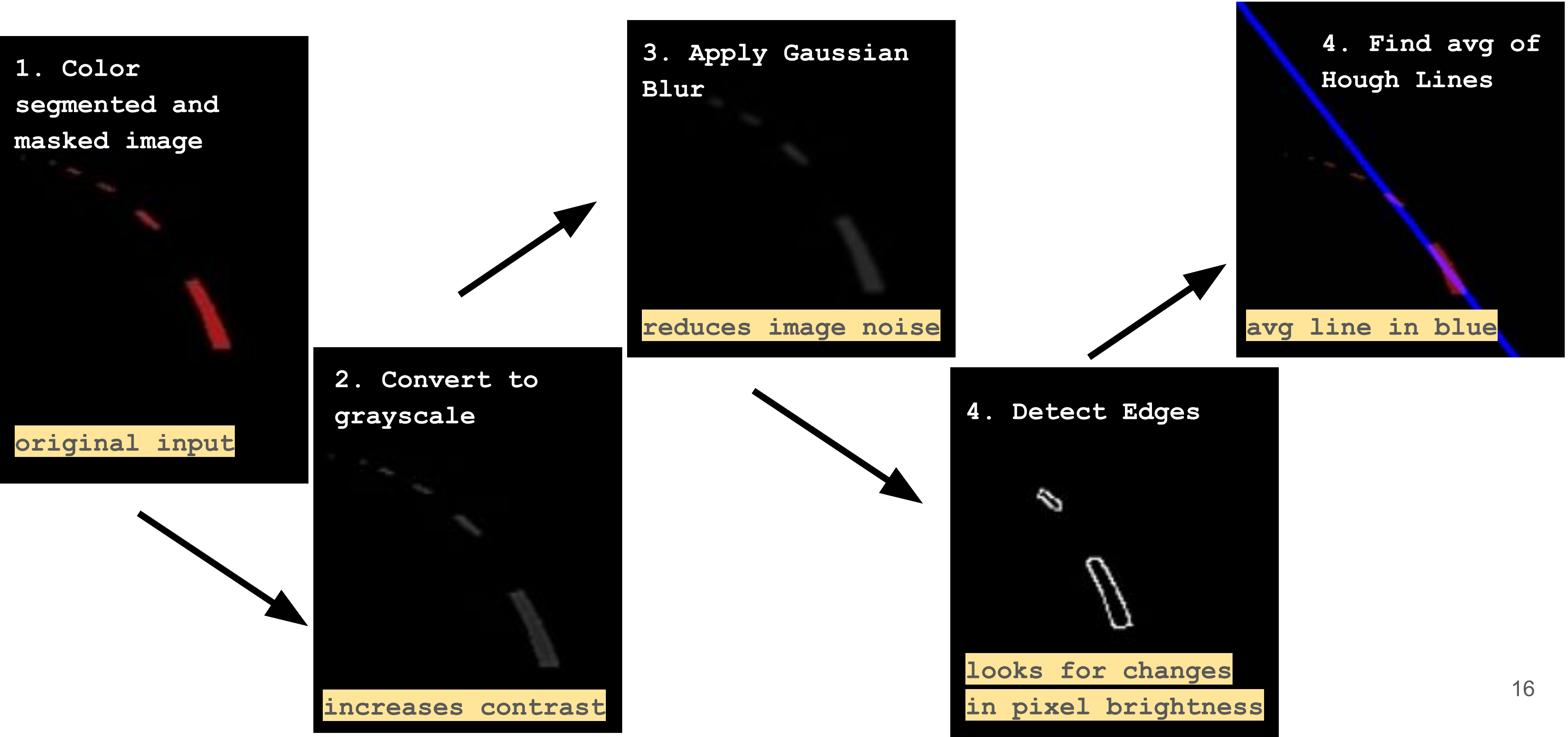
3. Use average line endpoints to determine its slope and intercept

m = (y2-y1)/(x2-x1)

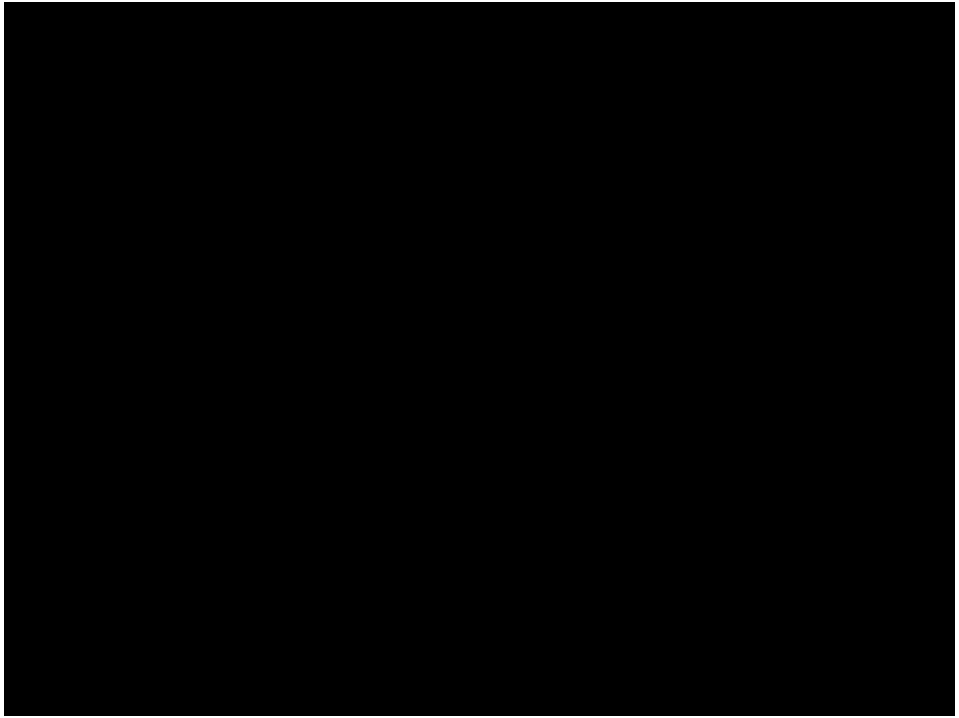
b = y1 - m*(x1)

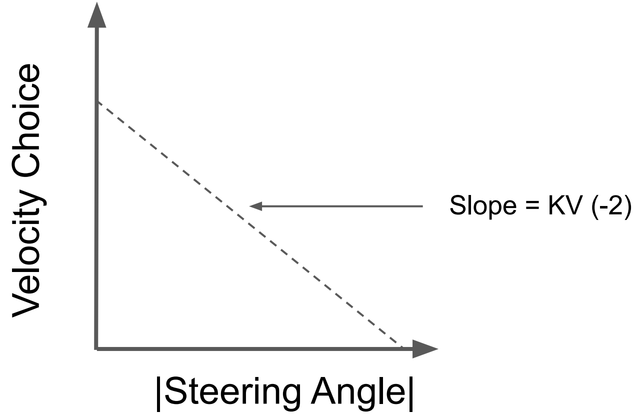


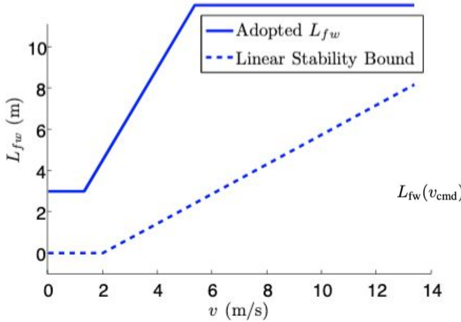
Hough Line Transforms successfully allowed us to approximate our desired follow line



Line Following Demo







V_MAX = 4, Lfw = 8.5

$$L_{\mathrm{fw}}(v_{\mathrm{cmd}}) = \begin{cases} 3 & \text{if } v_{\mathrm{cmd}} < 1.34 \text{ m/s ,} \\ 2.24\, v_{\mathrm{cmd}} & \text{if } 1.34 \text{ m/s } \leq v_{\mathrm{cmd}} < 5.36 \text{ m/s,} \\ 12 & \text{otherwise.} \end{cases}$$

Thank you!

We will now take questions.