CSCI/ECEN 3302 Introduction to Robotics

Alessandro Roncone

aroncone@colorado.edu https://hiro-group.ronc.one

Upcoming Deadlines

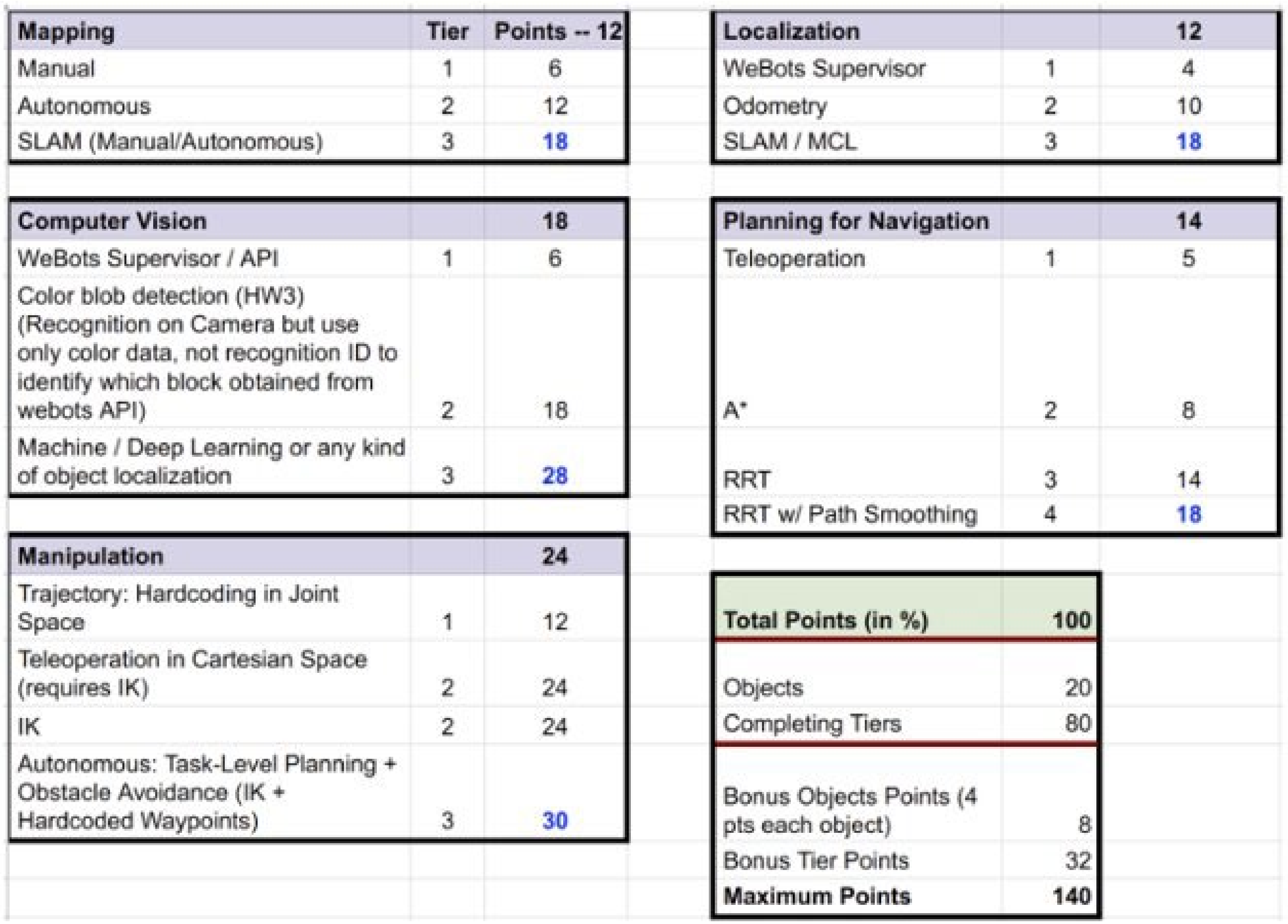
This is it!

No more assignments

- ~~Final Project Planning due Sunday, April 3~~
- HW3: Color Filtering + Blob Detection due Sunday, April 10
- Final Project Check-in due Sunday, April 17
- Final Project due Wednesday, April 27

No early submission EC or late days on any Final Project deadlines!

Final Project: Grocery Shopper Bot



Mapping	Tier	Points -- 12
Manual	1	6
Autonomous	2	12
SLAM (Manual/Autonomous)	3	18

Localization		12
WeBots Supervisor	1	4
Odometry	2	10
SLAM / MCL	3	18

Computer Vision		18
WeBots Supervisor / API	1	6
Color blob detection (HW3) (Recognition on Camera but use only color data, not recognition ID to identify which block obtained from webots API)	2	18
Machine / Deep Learning or any kind of object localization	3	28

Planning for Navigation		14
Teleoperation	1	5
A*	2	8
RRT	3	14
RRT w/ Path Smoothing	4	18

Manipulation		24
Trajectory: Hardcoding in Joint Space	1	12
Teleoperation in Cartesian Space (requires IK)	2	24
IK	2	24
Autonomous: Task-Level Planning + Obstacle Avoidance (IK + Hardcoded Waypoints)	3	30

Total Points (in %)	**100**
Objects	20
Completing Tiers	80
Bonus Objects Points (4 pts each object)	8
Bonus Tier Points	32
Maximum Points	**140**

Final Project: some advice

- Please focus on delivering a full product – ie, Tier 1 – first.
  - Tier 0 will give negative scores
- Some capabilities require more effort/work than others
  - E.g., CV or Manipulation w.r.t. Mapping
  - You may want to plan for more than 1 person to help out and/or have agile teams
- Integration is one of the hardest things in robotics
  - You will need to allot significant time for this
  - This final project is very comparable to real world robotics
- This is your opportunity to work on what you care about!

Upcoming Deadlines

- ~~Final Project Planning due Sunday, April 3~~
- HW3: Color Filtering + Blob Detection due Sunday, April 10
- Final Project Check-in due Sunday, April 17
- Final Project due Wednesday, April 27

No early submission EC or late days on any Final Project deadlines!

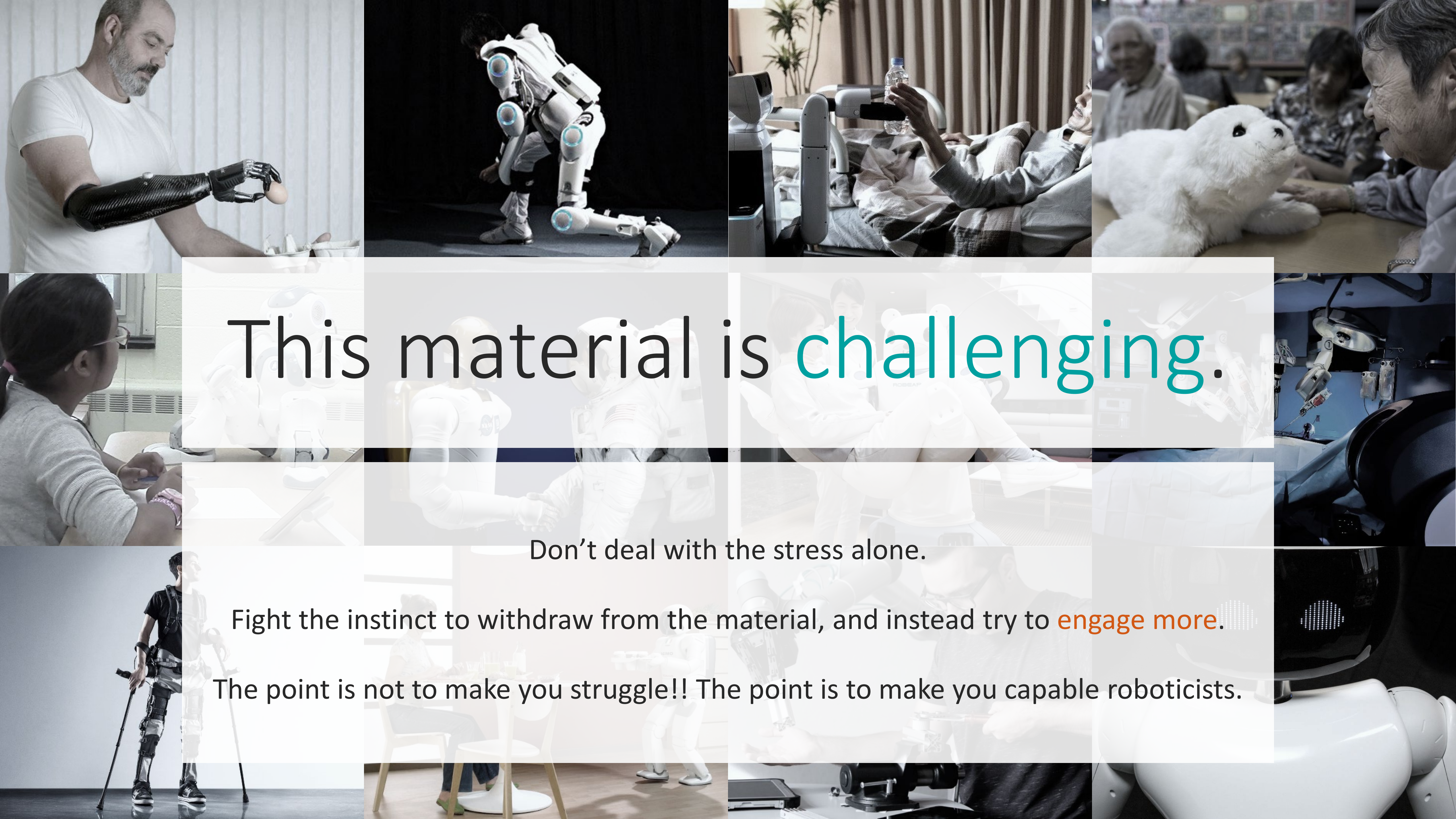
Upcoming Lectures

- Maja Mataric AMA – April 26th – Last day of lectures!
- Kinova Robotics – Guest lecture – April 19th – 30m
- Robotics AMA – April 12th – next week!

  https://canvas.colorado.edu/courses/81803/quizzes/236883

EC Opportunities

- Mid-semester review did not meet the quota ☹
- Maja Mataric AMA + Robotics AMA – please see my email and contribute to https://piazza.com/class/ky8w23o9yhn4yb?cid=139
- Remember: you most likely will need EC to get an A
  - Active participation → $\pm$1 letter grade!



This material is challenging.

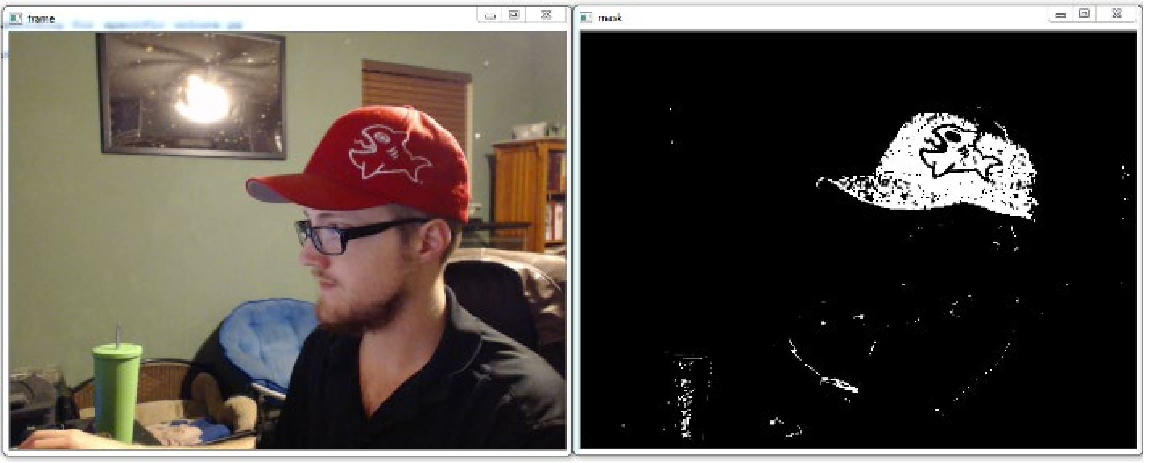
Don’t deal with the stress alone.

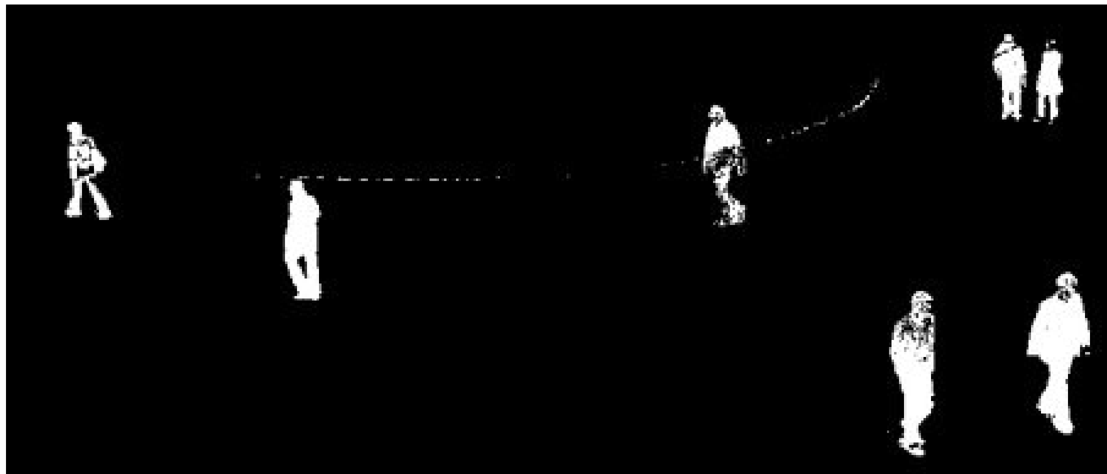
Fight the instinct to withdraw from the material, and instead try to engage more.

The point is not to make you struggle!! The point is to make you capable roboticists.

Chapter 13

RECAP
Computer Vision





Introduction to Computer Vision: A Practical Approach

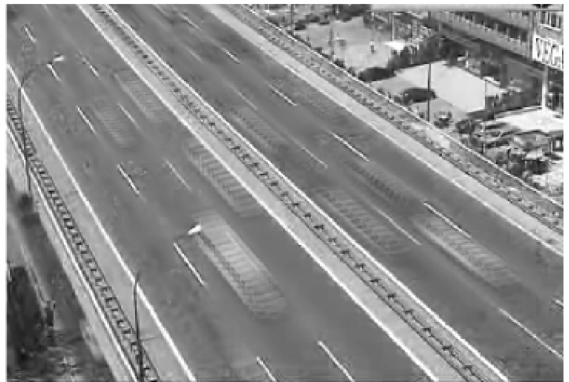
Algorithms that can help with well-structured perception problems:

- Background subtraction
  - Isolate only the relevant foreground elements
- Color filtering
  - Isolate only the pixels within certain color ranges
- Blob detection
  - Discover connected components in your image
- Camera Calibration
  - Remove distortions that come from your sensor’s lens

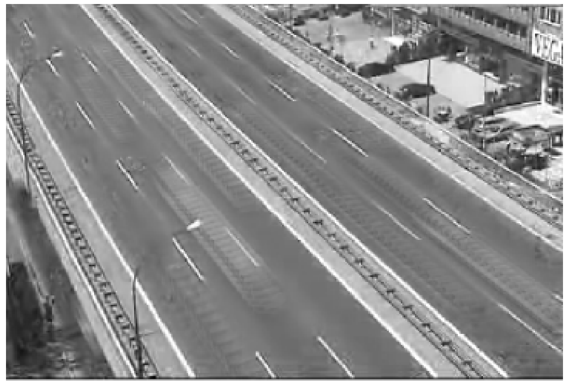
Background Subtraction: Mean Filter

- Assume the background is the mean of the previous n frames:
  - $I_{bg}(x,y,t) = \frac{1}{n}\sum_{i=0}^{n-1} I(x,y,t-i)$
- Use a threshold value T to determine if actually different enough to consider foreground:
  - $\left| I(x,y,t) - \frac{1}{n}\sum_{i=0}^{n-1} I(x,y,t-i) \right| > T$
- Background estimate changes with n:

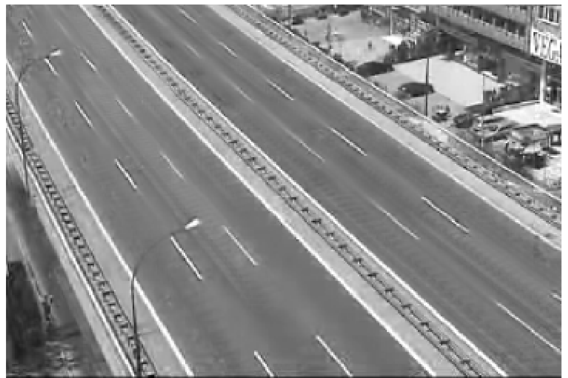
$\boldsymbol{n = 10}$



$\boldsymbol{n = 20}$



$\boldsymbol{n = 50}$



Background Subtraction

- Advantages
  - Easy to implement!
  - Reasonably fast, can be done in realtime
  - Works reasonably well for structured problems
- Disadvantages
  - Frame accuracy depends on object speed and framerate
  - High memory requirements if multiple backgrounds stored
  - Uses a single global threshold value for all pixels
    - Fails for bimodal backgrounds
    - Fails for slow-moving objects
    - Fails for high-variance lighting conditions

$n = 10$

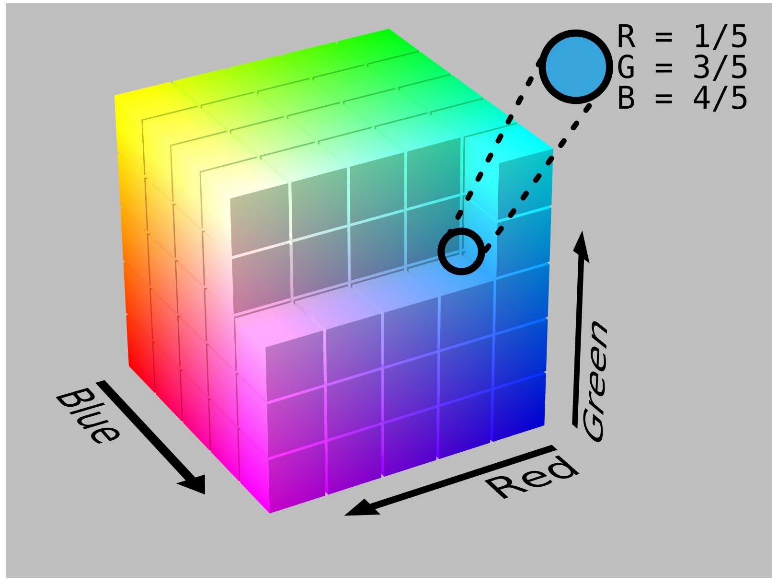
$n = 20$

$n = 50$

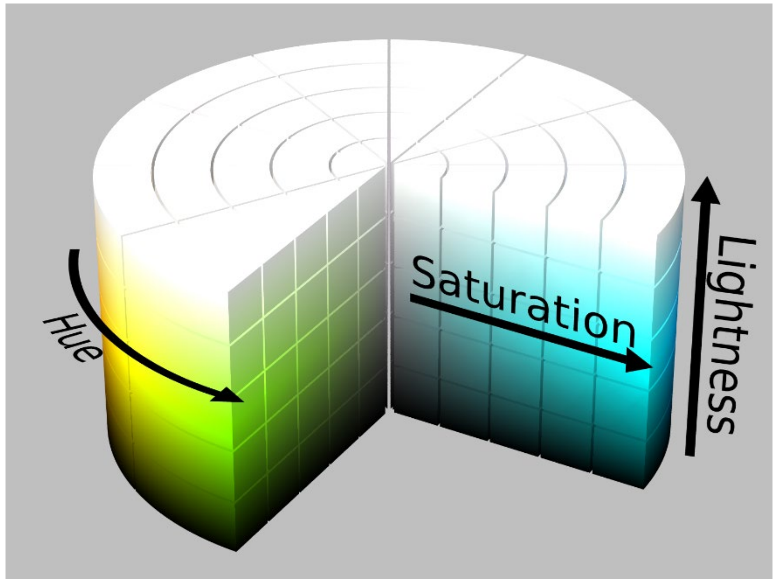
Color Filtering

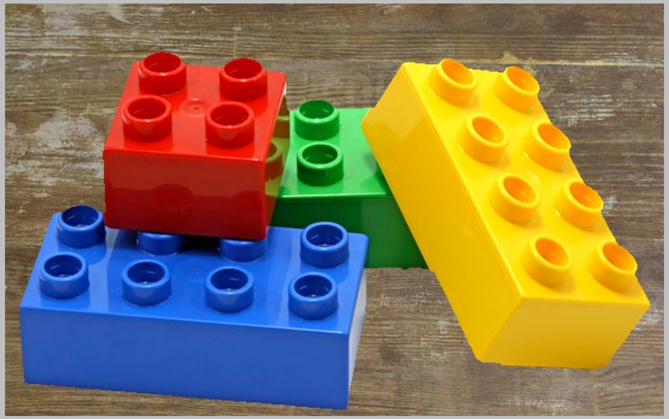
HSV filtering is highly recommended!

RGB Space:



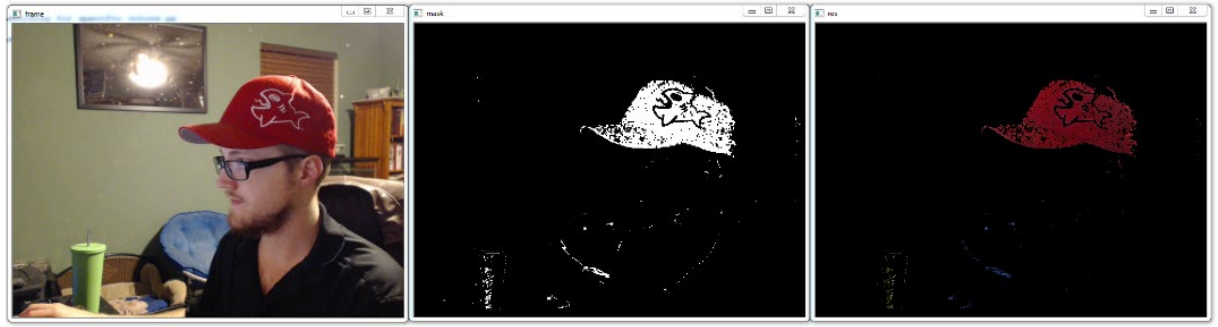
HSV Space:





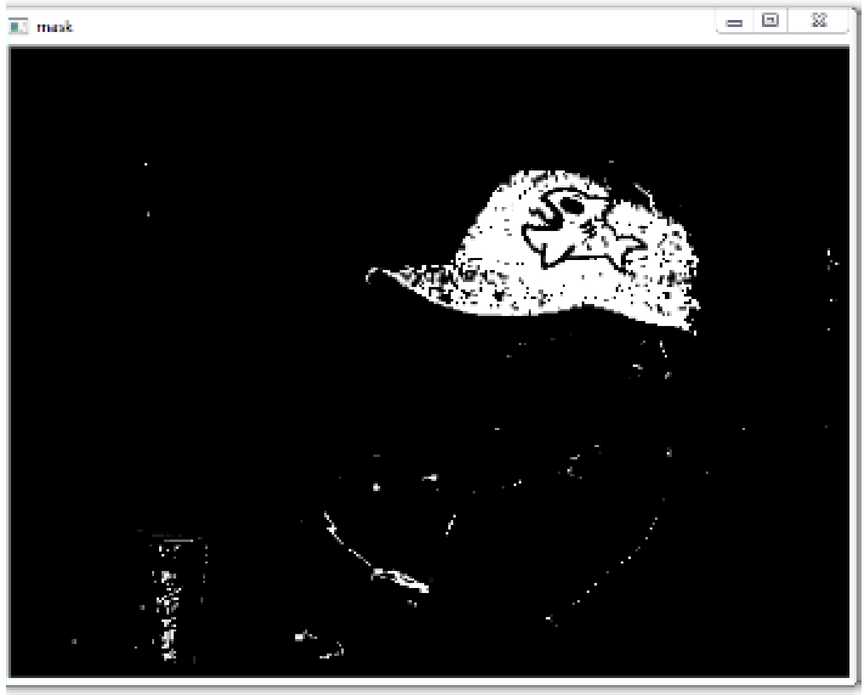
Color Filtering

- **Goal:** Find the color of the object you're looking to track
  - Capture images under various lighting conditions
  - Use an image editor to sample pixel values from the target object
  - Choose a color range based on this information
- Mask out all values that don't match the color range



```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    res = cv2.bitwise_and(frame,frame, mask= mask)

    cv2.imshow('frame',frame)
    cv2.imshow('mask',mask)
    cv2.imshow('res',res)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

Color Filtering



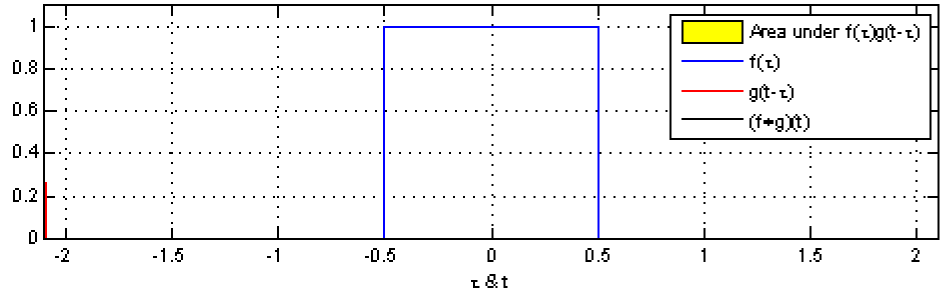
Color filtering often produces results like this, with regions of the target object getting filtered away

How can we get rid of this noise?

Color Filtering: Simple Blur

- Blurring will remove sharp discontinuities
- Simple blur: Convolve image with averaging kernel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



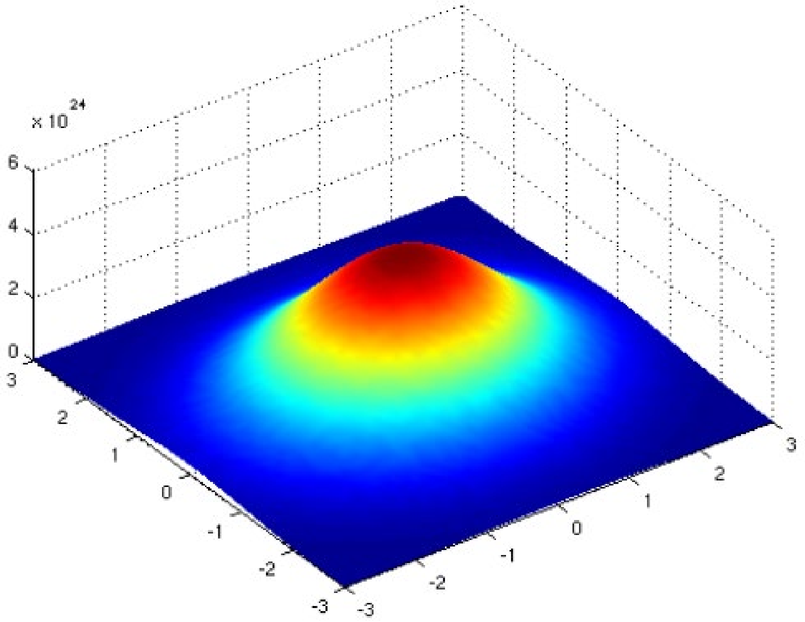
Center pixel will be averaged with its neighbors

Gaussian Blur

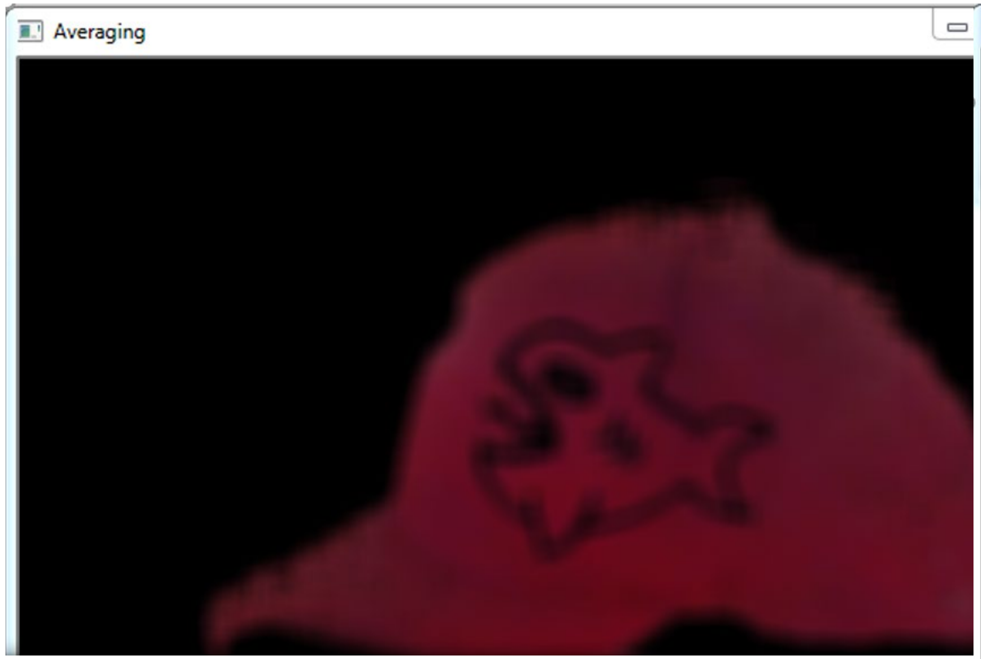
$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

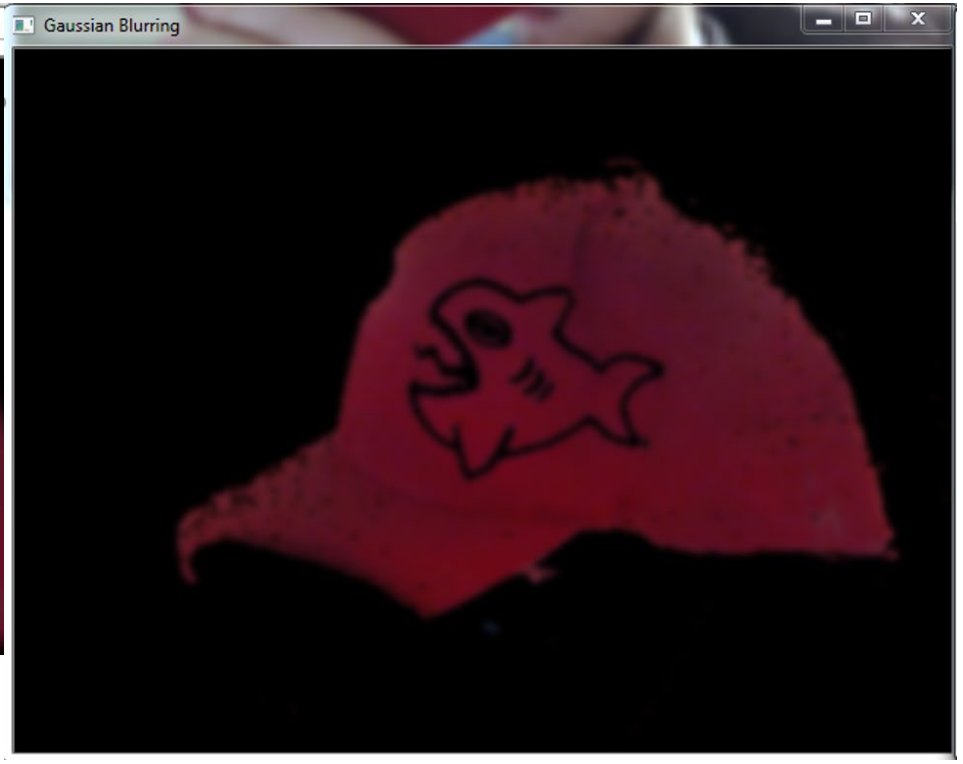
$$\mathcal{N}(\mathbf{x}|\mu,\mathbf{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)}$$

Define blur kernel as a Gaussian distribution rather than a flat average (1/9th kernel looks like a table)



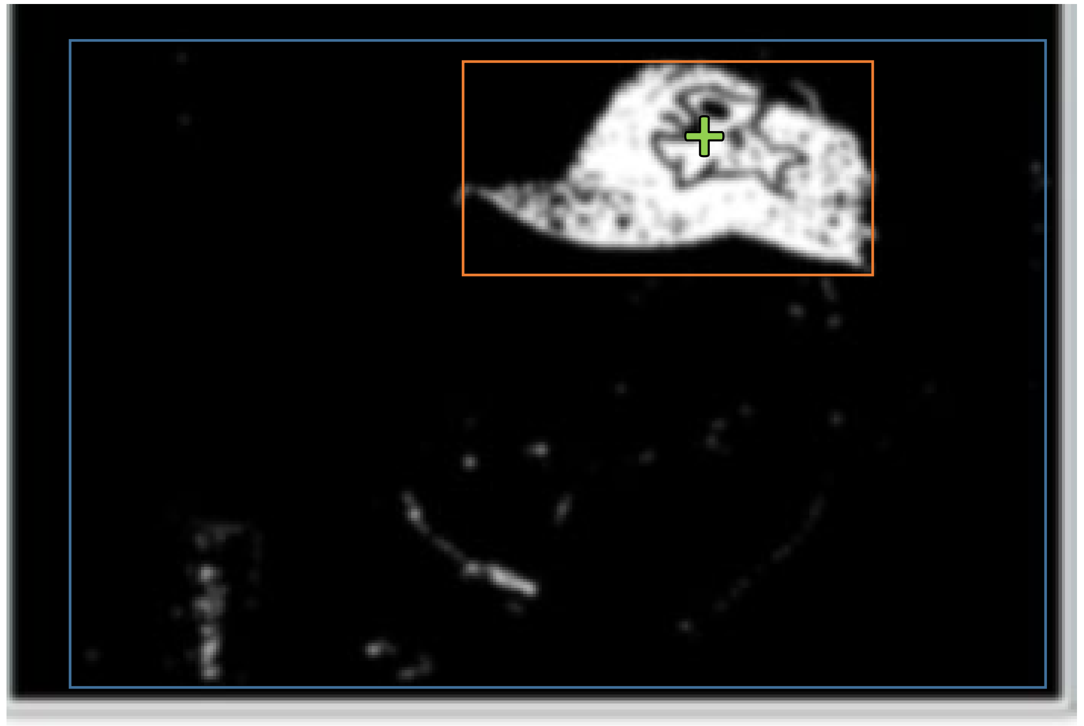
Simple (Averaging) vs. Gaussian Blur



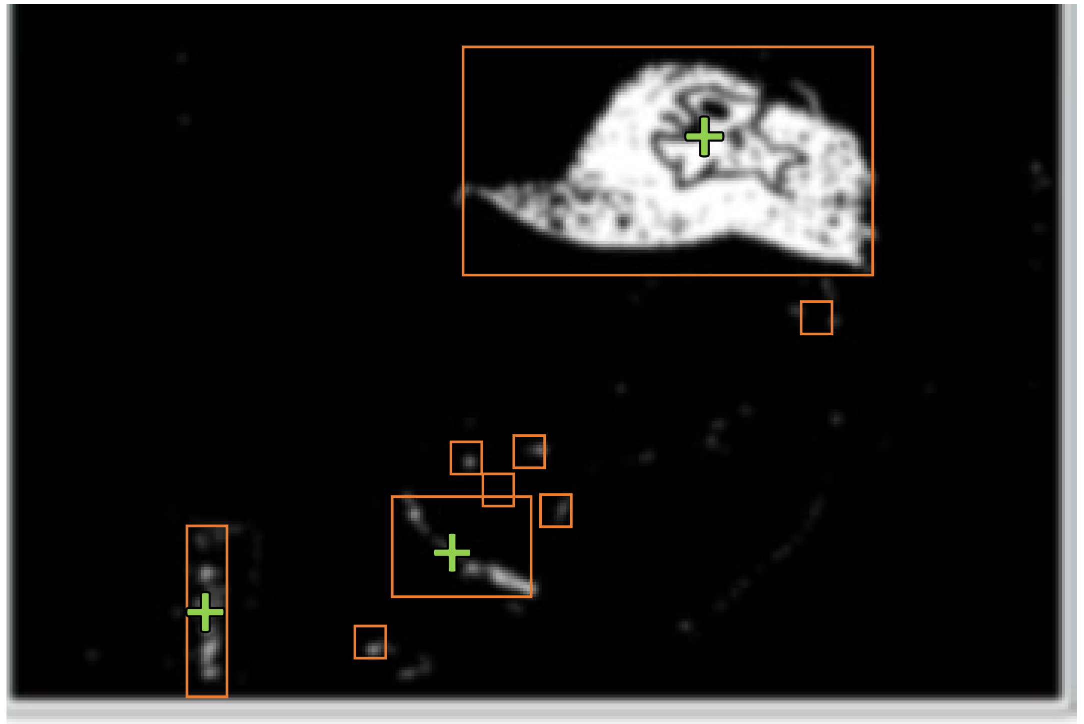


https://pythonprogramming.net/blurring-smoothing-python-opencv-tutorial/?completed=/color-filter-python-opencv-tutorial/

Blob Detection:
Often want to find Bounding Box and Centroid

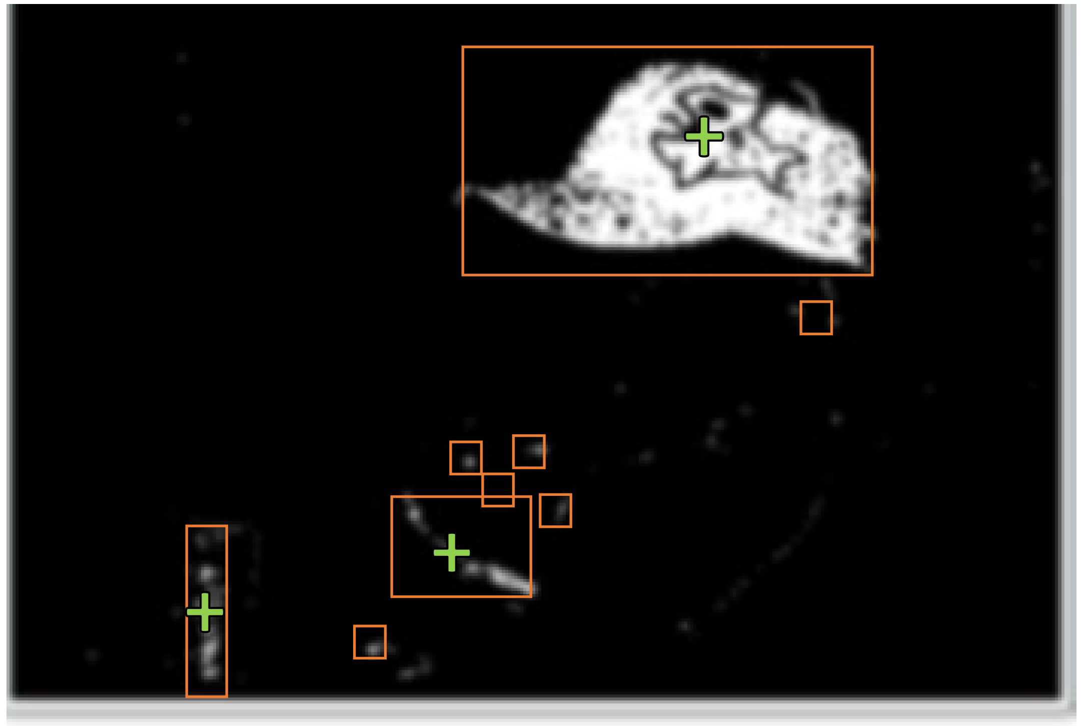


Making Blob Detection Work



How do we avoid situations like this one for our hat detector?

Making Blob Detection Work



Bring more prior knowledge into the problem!

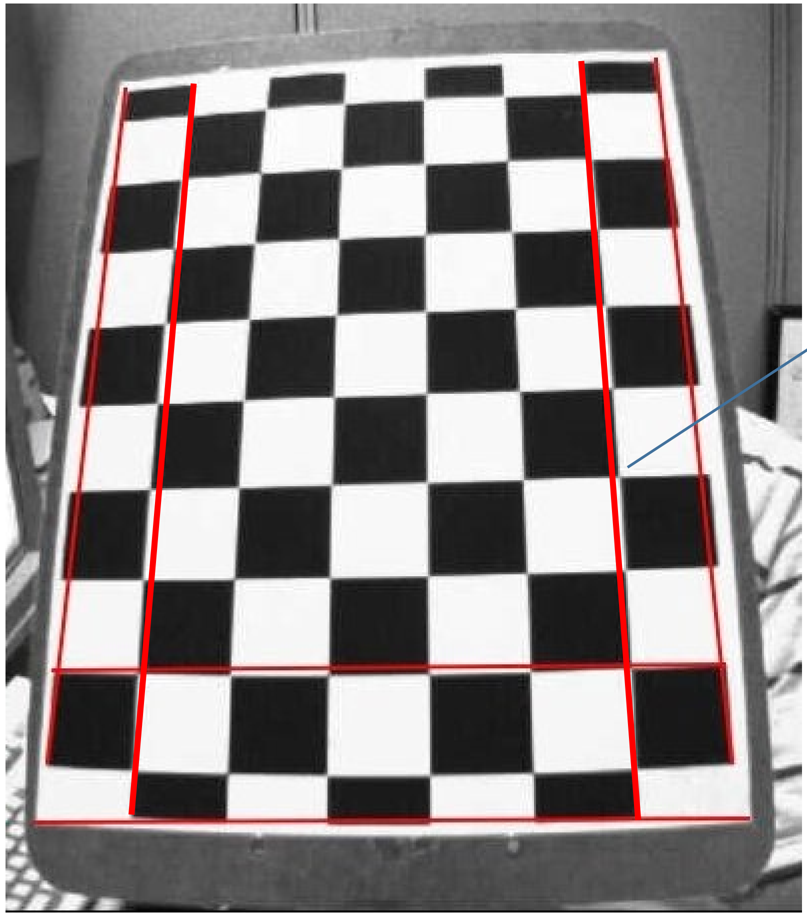
Currently using:

- Color

Not currently using:

- Size
- Shape
- Volume
- Location
- Movement Characteristics
- …

Regular Camera Image of Chessboard Pattern



Note how the lines can't fit the supposedly straight edges along each column

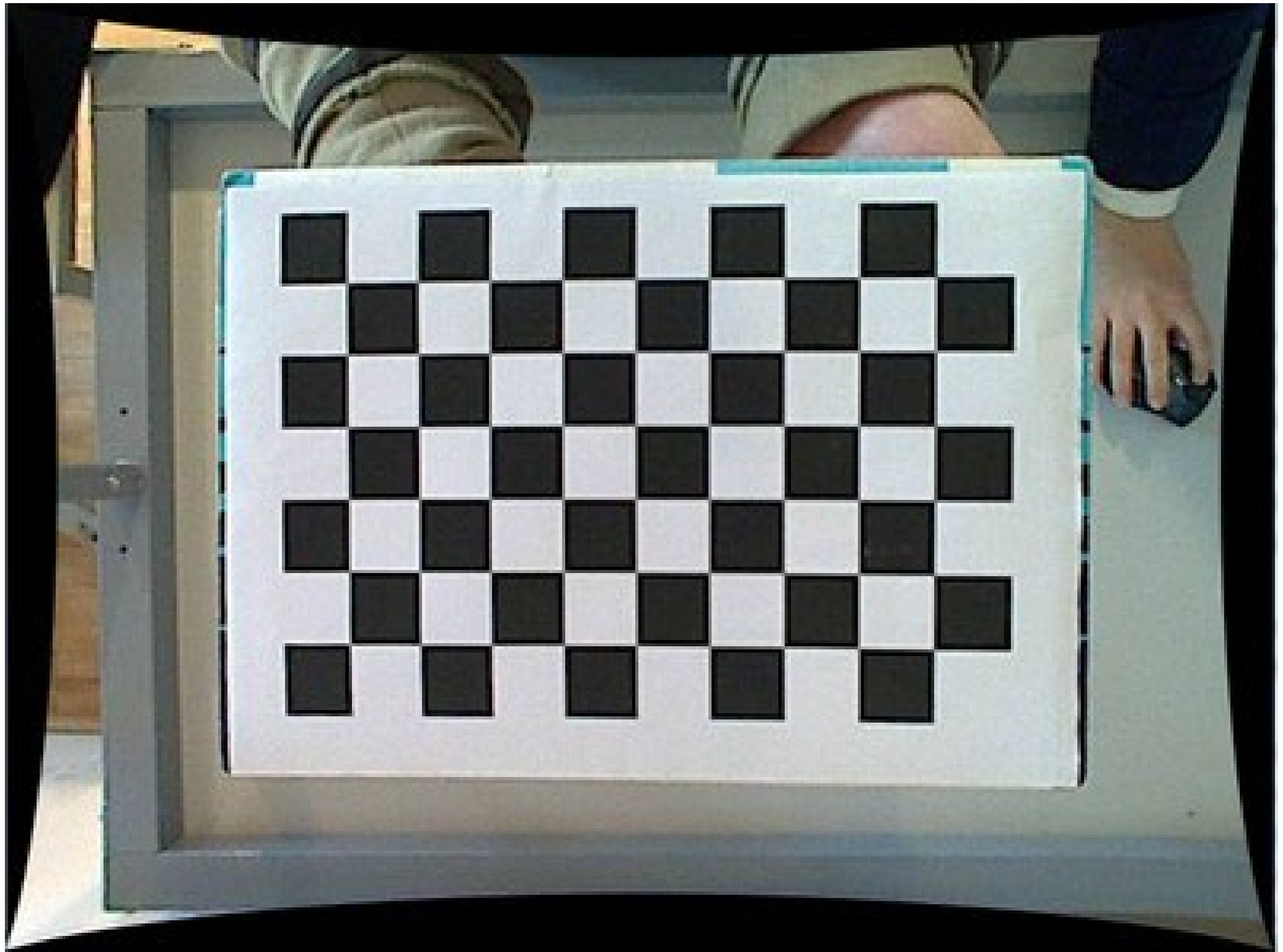
Pixels to Meters

Now that we know how to isolate relevant sections of our image, we must translate from pixel space to real world coordinates!

…Except camera lenses introduce distortions!

Your images will be increasingly warped the further from the center of the camera you go.

“Fixed” Camera Image of Chessboard Pattern



Fixing Images with Camera Calibration

- Two sources of distortion:
  - Radial distortion from the lens
    - Causes perceived curvature / fisheye effect

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

  - Tangential distortion from image lens not being parallel to imaging plane
    - Causes distance misrepresentation

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

  - Must find distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$

- Also need Intrinsic and Extrinsic Camera Parameters:
  - Focal length (f_x, f_y)
  - Optical centers (c_x, c_y)

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Learning Calibration Parameters

- Given a predetermined pattern (e.g., Chessboard), corner-to-corner distances are known within the image

- Given multiple images of the pattern in different parts of the frame at different orientations, we can figure out which parameters will give proper corner distances.

- Python Tutorial on OpenCV Camera Calibration: https://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

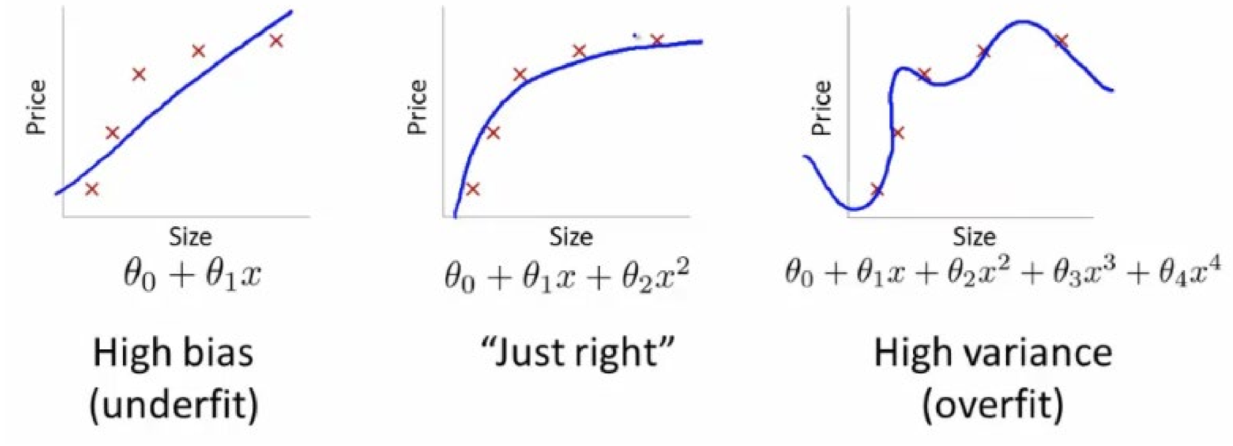
- Video Tutorial on OpenCV Camera Calibration (C++): https://www.youtube.com/watch?v=HNfPbw-1e_w&list=PLAp0ZhYvW6XbEveYeefGSuLhaPlFML9gP&index=14

Machine Learning

An Introduction

Introduction to Machine Learning

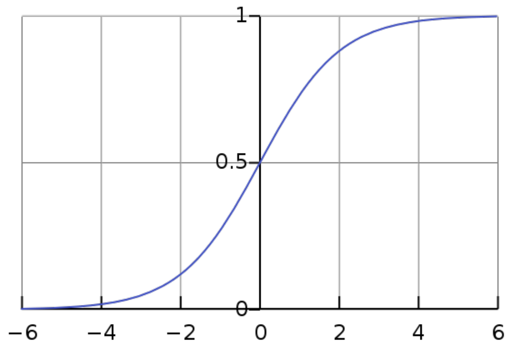
- Regression: How much is this house worth?
- Classification: Is this a photo of a dog or ice cream?



Linear Regression to Logistic Regression

- Linear Regression gives us a continuous-valued function approximation
  - Models relationship between scalar dependent variable y and one or more variables X
- Logistic regression allows us to approximate **categorical** data
  - Pick a model function that squashes values between 0 and 1

$$F(x) = \frac{1}{1 + e^{-x}}$$

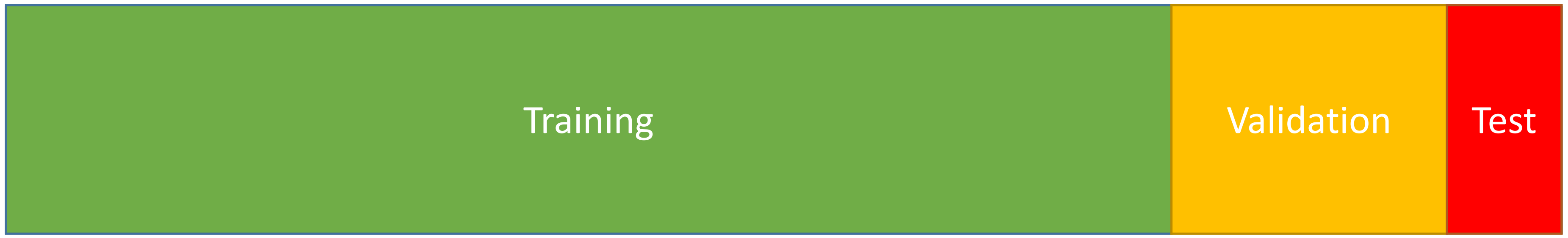


  - Apply it to a familiar function: $g(X) = \beta_0 + \beta_1 x + \epsilon$

$$P(Y=1) = F(g(x)) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * x)}}$$

Training and Testing Your Algorithms

- Partition your data into TRAIN, VALIDATE, and TEST
- Train your model on TRAIN
- Evaluate your model on VALIDATE
- TRAIN and VALIDATE can be swapped around
- You only get to run on TEST once, ever!



Types of Learning

Supervised Learning

- Given a dataset of (X, y) pairs
- X: Vector representing a data point
- y: Label or value that is the correct answer for $f(X) = y$

"You guess, I tell you the correct answer, you update, repeat"

Reinforcement Learning

- Given some reward function R
- $R(s, a, s')$ gives the value of moving from state s to s' via action a

"You guess, I tell you if you're on the right track (sometimes)"

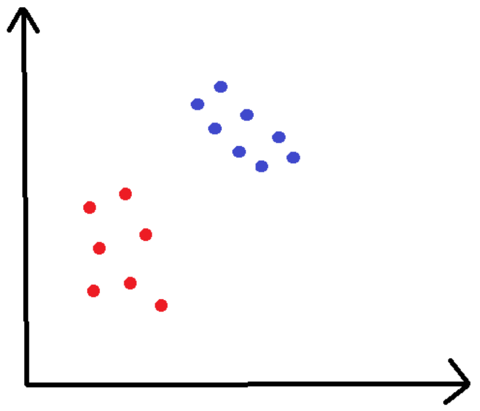
Unsupervised Learning

- Given a dataset of X's
- X: Vector representing a data point
- No labels (answers) given!

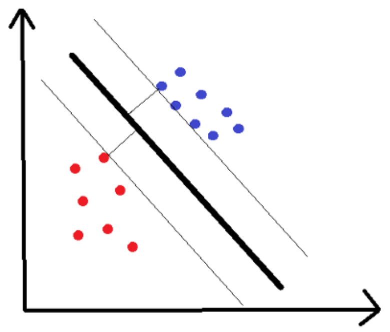
"You guess, I have no feedback to give you. Good luck!"

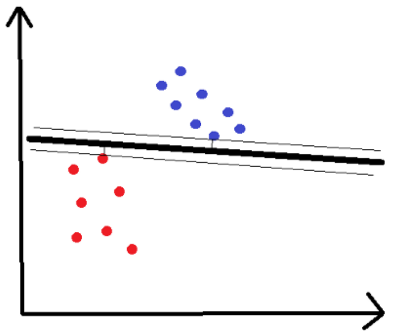
Supervised Learning: Support Vector Machines

- Is this dot red or blue?



- Draw a separating line!
  (Or hyperplane)



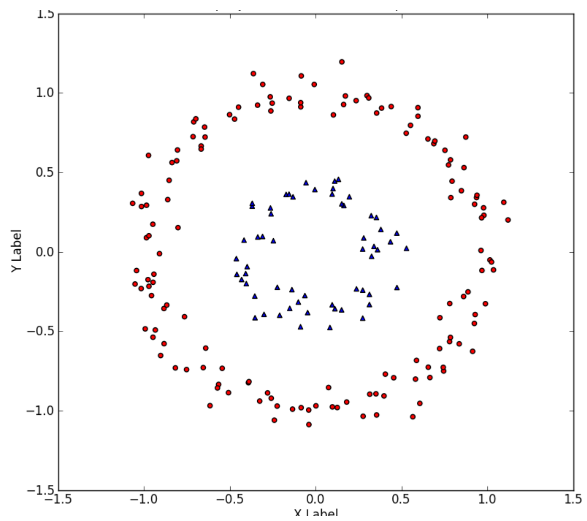


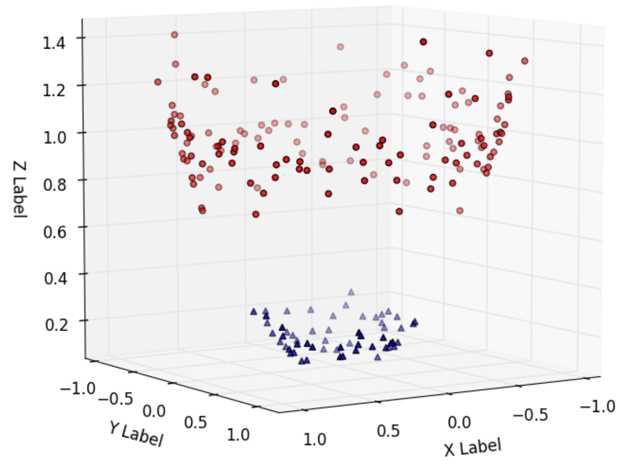
Supervised Learning: Support Vector Machines

What happens if the data doesn't separate cleanly?

- Add a cost for misclassified examples and let the optimizer take care of it!

- Add more dimensions to the data!
    - $x^2, y^2, x^2 + y^2, \cos(x), xy$, etc.





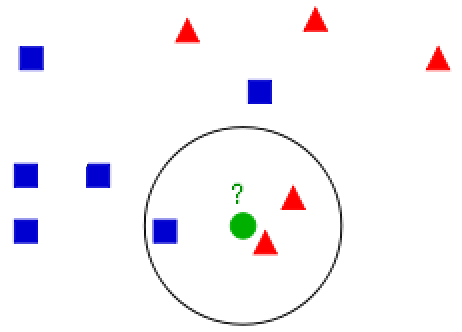
Supervised Learning: Support Vector Machines

Practical details:

- Scikit-Learn (sklearn) Python Package has excellent documentation
  - http://scikit-learn.org/stable/modules/svm.html
- For easier problems, can pretty much use it out of the box to get decent results

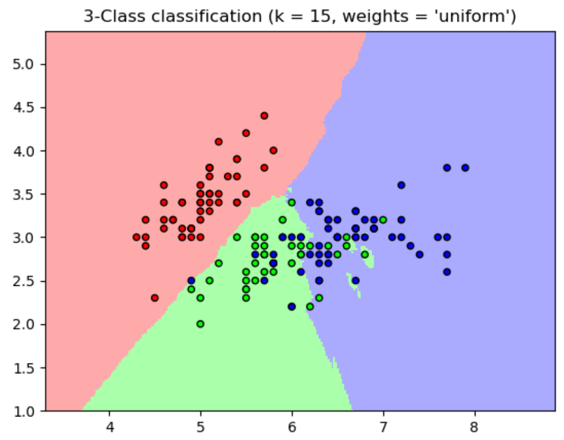
K-Nearest Neighbor

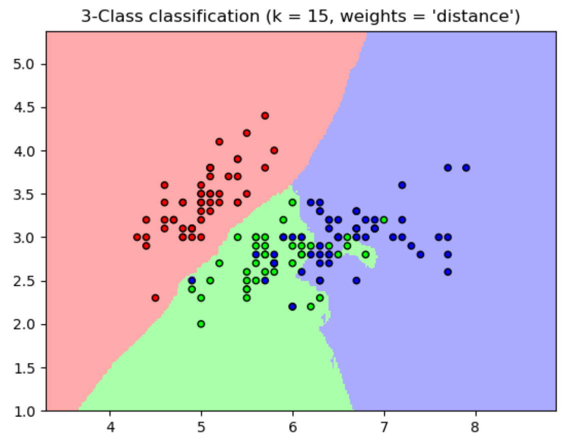
- Can be used for classification or regression
- Simple idea:
  - For a given data point p, find the K nearest labeled points
  - Assign the majority label to p
- Caveats:
  - The order that you label points can matter!
  - Slow! Lots of comparisons required.
  - Choosing 'K' has a big impact



Weighted K-Nearest Neighbor

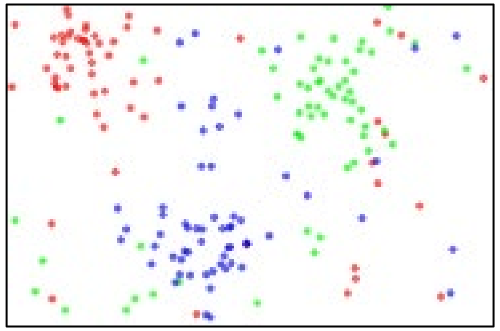
Weight each sample's influence by how far away it is from p



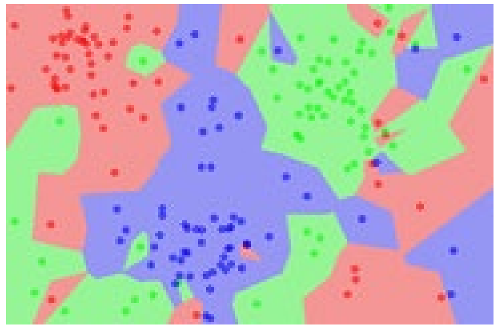


Reduced K-Nearest Neighbor

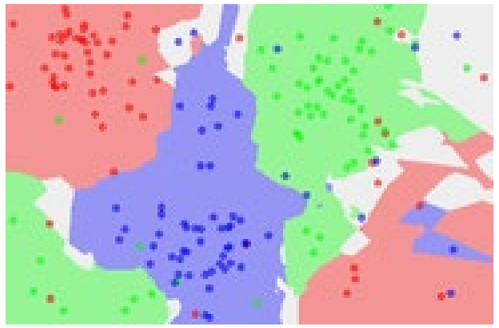
Dataset



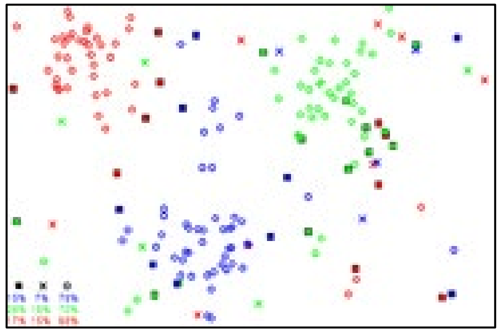
1-NN Classification



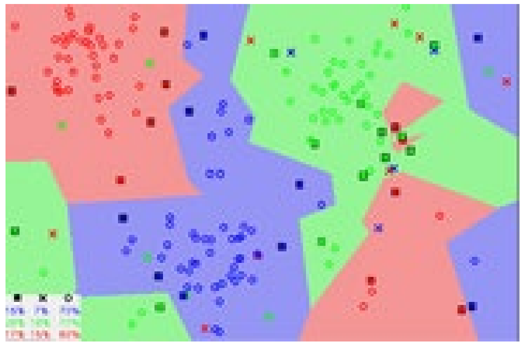
5-NN Classification



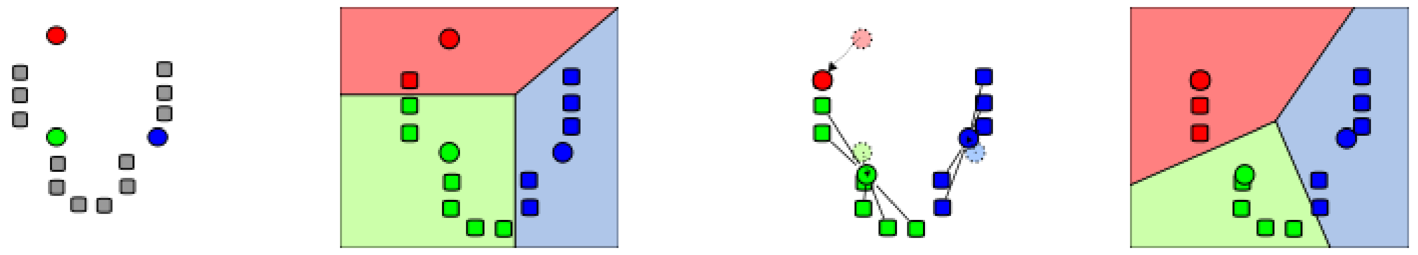
Reduced Dataset



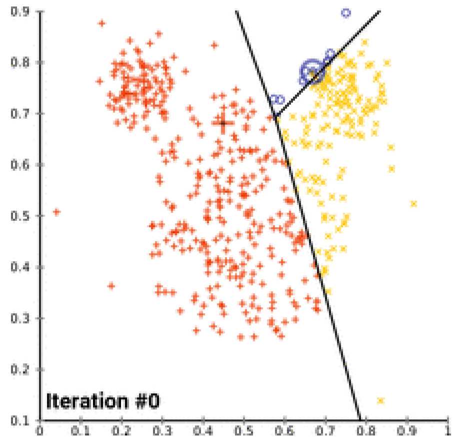
1-NN Classification



Unsupervised Learning: K-Means Clustering



1. Randomly initialize k clusters at random positions.
2. Classify every data point as belonging to a cluster by Euclidean distance measurement (closest cluster wins)
3. Calculate centroid of each cluster. Relocate cluster to centroid position.
4. Repeat 2-3 until centroids converge.


Iteration #0