CSCI/ECEN 3302
Introduction to Robotics

Alessandro Roncone

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

Extra credit #2!!! Inbox ×

Alessandro Roncone
to CSCI3302_010_20221_B, ECEN3303_010_20221_B, Shivendra, Anuj, Supraja

Fri, Mar 25, 11:30 AM (3 days ago)

Dear all,

I hope you are having a great spring break and that you are out and about enjoying life outside of the University. In case you were missing me, here am I with a second (and bigger) extra credit opportunity! And some reminders.

A. As I mentioned, on the last day of class (April 26th) Prof. Maja Mataric from USC will give an AMA-style intervention. To foster a productive and constructive session, I am asking you to take some time to do the following:

1. Watch the recording of Maja's presentation when I (virtually) invited her to give a talk at CU last Fall at this link: https://drive.google.com/file/d/1ZRSbOCM0WUkD46-bTVNRvNKw17R3KsVf/view (it's also in the canvas modules). I find that talk extremely inspiring and I strongly recommend you think critically about how what she says; it may impact your interest and future career in the fields of Robotics and AI!! Lots of good content there--for example, Human Augmentation vs Human Automation, or doing things that matter.
2. Read the collection of blog articles from Rodney Brooks (MIT Professor, one of the most important contributors to robotics and AI) called Dated Predictions (https://rodneybrooks.com/category/dated-predictions/). This blog series is now at its fifth installment (it started in 2018), and in there Brooks gives a yearly update on the future of Self Driving Cars, Robotics/AI/ML, and Space. His predictions are: 1) very aligned to what I and most people in the field think; 2) a good contrast with what you may have heard from public figures (you know who I'm thinking about) and the press; 3) mostly correct (at least, much more than the aforementioned public figures and the press). The articles are very long, so I recommend you to use your favorite article-to-podcast app to listen to them (I use Pocket http://getpocket.com).
3. Discuss! Share your opinions! Share your disagreements if you have any! Starting from next week, we will open a Piazza post to begin discussing these broader topics. **Active, constructive, and collegial participation in this activity will directly contribute to your participation grade**---which may increase your grade by a letter or more!

B. Also, remember that the mid-semester review form is still open and that we still need to hit the quota for extra credit! We have 118 students enrolled but only 78 answered (i.e. only 66%). This is the easiest thing to do and it will basically add ~half a letter grade only if we reach 107 responses! https://forms.gle/gLbqF9jBaX2Uz6CY6

C. Thank you to the 28 people that filled the Robotics AMA form. Your questions are great! More importantly, they are very insightful and will make for a very interesting AMA session. I haven't collected the questions yet (I will next week), so if you haven't already you are encouraged to spend 20 seconds to complete the poll! https://canvas.colorado.edu/courses/81803/quizzes/236883

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!

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

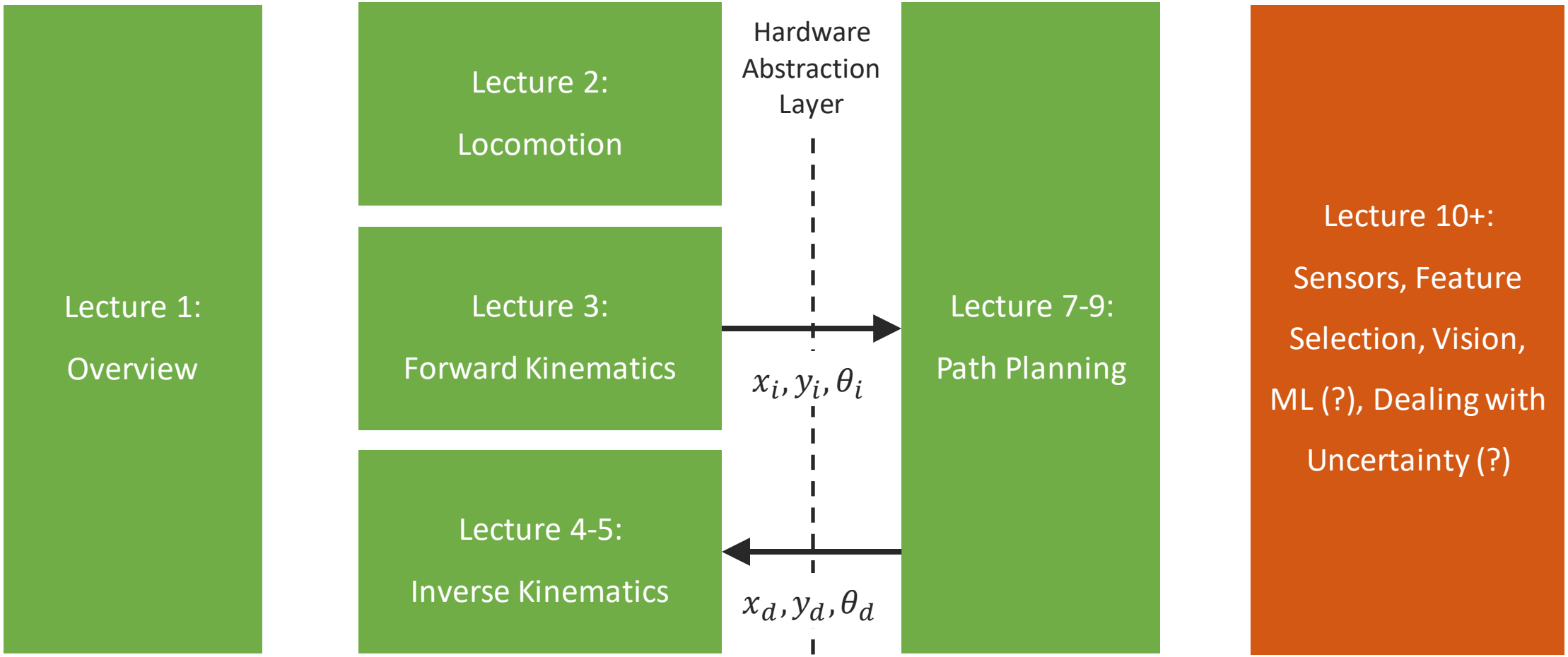
Computer Vision		18
WeBots Supervisor / API	1	6
Color blob detection (HW3) (Recognition on Camera but use only color data, not recognition ID to identify which block obtained from webots API)	2	18
Machine / Deep Learning or any kind of object localization	3	28

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

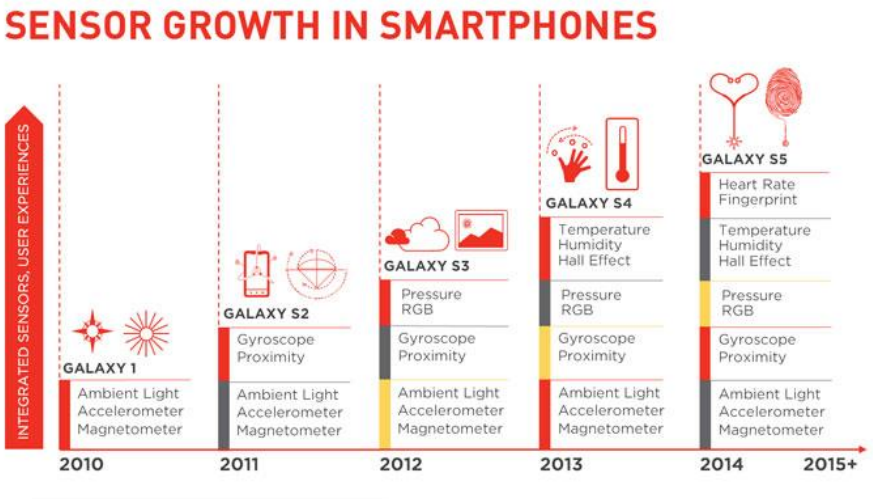
Total Points (in %)	100
Objects	20
Completing Tiers	80
Bonus Objects Points (4 pts each object)	8
Bonus Tier Points	32
Maximum Points	**140**

Roadmap


Lecture 1:
Overview
Lecture 2:
Locomotion
Lecture 3:
Forward Kinematics
Lecture 4-5:
Inverse Kinematics
Hardware
Abstraction
Layer
x_i, y_i, θ_i
x_d, y_d, θ_d
Lecture 7-9:
Path Planning
Lecture 10+:
Sensors, Feature
Selection, Vision,
ML (?), Dealing with
Uncertainty (?)

Sample of everyday sensors

Camera



US (ultrasonic)

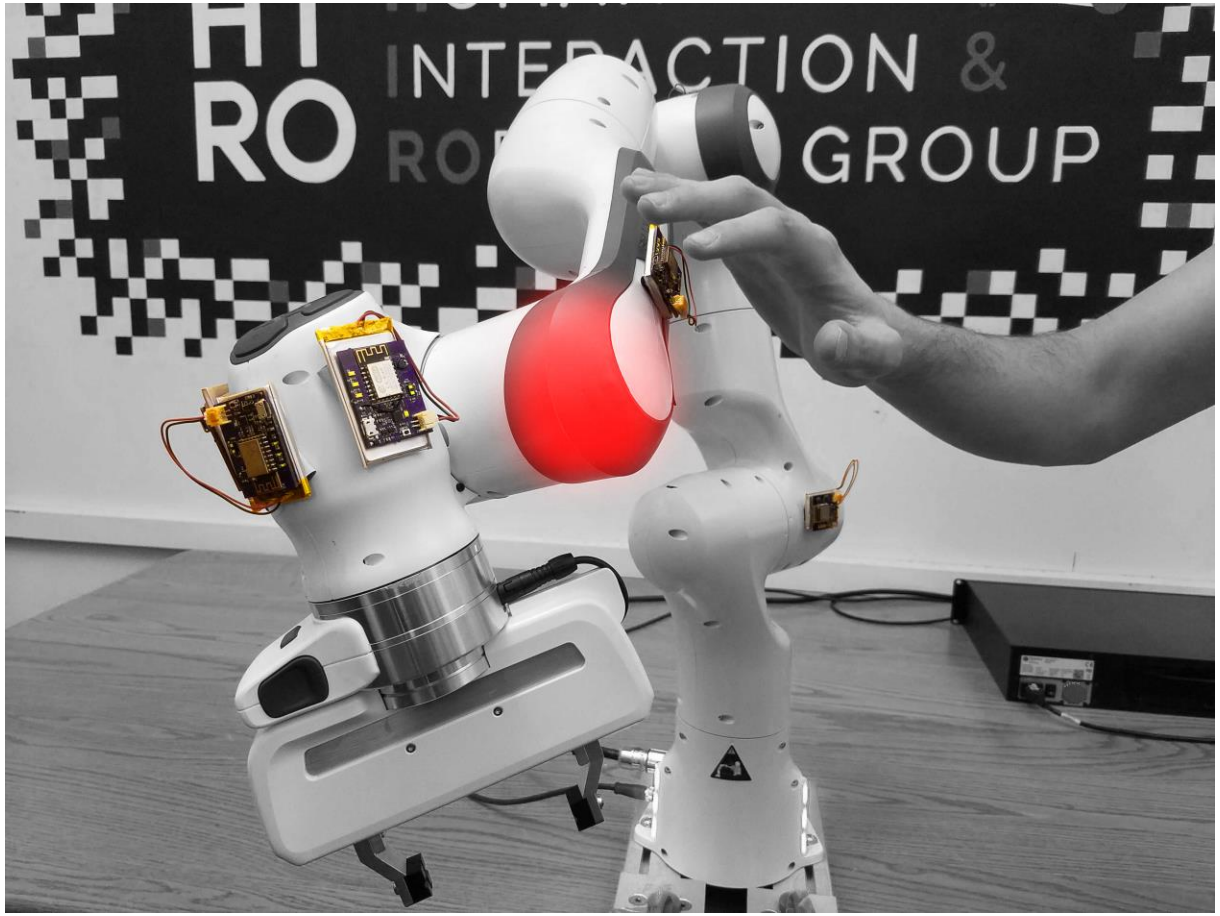
3D

PIR
(passive infrared)

Radar

So far: Robotics always benefits, never drives sensor development

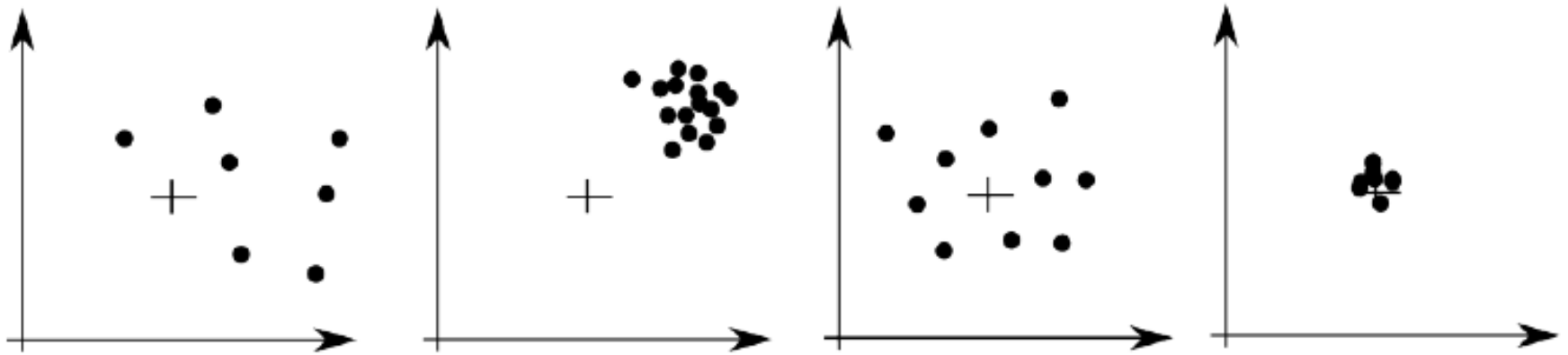
Artificial Skin for Robots



Sensor characteristics

- Active vs. Passive – emit signal into environment vs. consume
- Resolution – minimum difference between values
- Accuracy – difference between measured and true value
- Precision – reproducibility of results
- Linearity – how output varies as function of input
- Bandwidth – speed with which measurements are delivered
- Range – difference between highest and lowest reading
- Dynamic Range – ratio of lowest and highest reading
- Cross-sensitivity – sensitivity to environment

Precision vs. Accuracy



Neither precise nor accurate

precise and not accurate

accurate and not precise

precise and accurate

Performance of a laser scanner

- Range: difference between highest and lowest reading
- Dynamic range: ratio of lowest and highest reading
- Resolution: minimum difference between values
- Linearity: variation of output as function of input
- Bandwidth: speed with which measurements are delivered
- Cross-Sensitivity: sensitivity to environment
- Accuracy: difference between measured and true value
- Precision: reproducibility of results

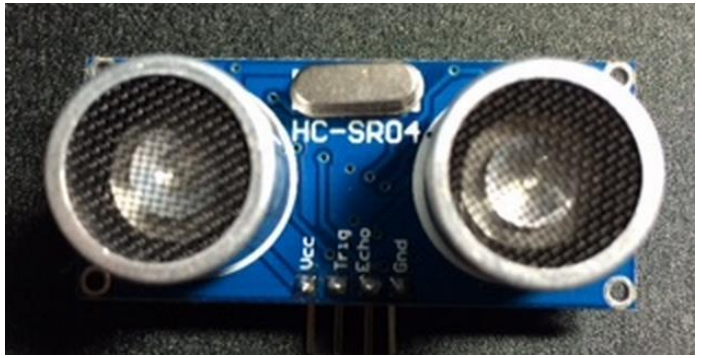
Hokuyo URG

Specification	URG-04LX
Power source	Regulated 5V ±5%
Interface	RS232, USB
Detection Distance	20 to 4000 (mm)
Guaranteed Accuracy (min to 1m)	±10mm
Guaranteed Accuracy (1m to max)	1% of detected distance

Specifications	
Power source	5V +/-5%
Current consumption	0.5A (Rush current 0.8A)
Detection range	0.02 to approximately 4m
Laser wavelength	785nm, Class 1
Scan angle	240°
Scan time	100msec/scan (10.0Hz)
Resolution	1mm
Angular Resolution	0.36°
Interface	USB 2.0, RS232
Weight	5.0 oz (141 gm)

Performance of an ultrasonic sensor

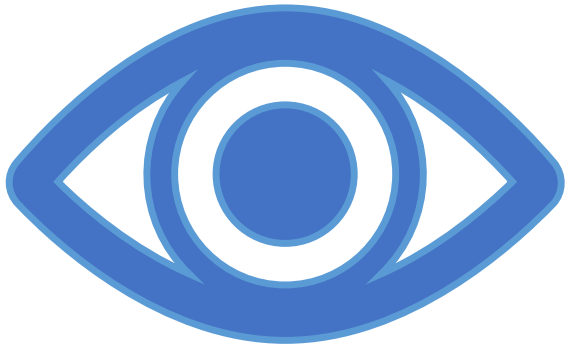
- Range: difference between highest and lowest reading
- Dynamic range: ratio of lowest and highest reading
- Resolution: minimum difference between values
- Linearity: variation of output as function of input
- Bandwidth: speed with which measurements are delivered
- Cross-Sensitivity: sensitivity to environment
- Accuracy: difference between measured and true value
- Precision: reproducibility of results



- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Currnt: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1" - 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm

Summary

- Sensors do not serve specific applications and no sensor solves a problem completely
- Many sensors observe the same phenomenon using different physical principles
- Different sensors have different trade-offs qualified in their different precision, accuracy, bandwidth, dynamic range and resolution
- There are smart ways to extract the desired information from a set of sensors and fuse them [Sensor Fusion]



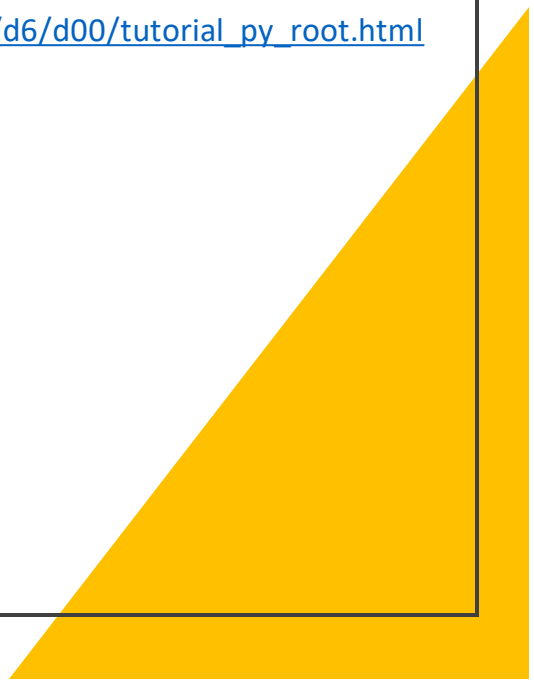
Computer Vision Fundamentals

(mostly to help with Final Projects)

Introduction to Computer Vision

A Practical Approach

- Software packages that will help:
  - **OpenCV**
    Tutorials at
    https://docs.opencv.org/3.3.0/d6/d00/tutorial_py_root.html
  - **Numpy**



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

- Given an image, typically we care more about the **foreground** than the **background**.
- If we can cut out the irrelevant parts of the image, we can spend our processing time more wisely!

Background Subtraction

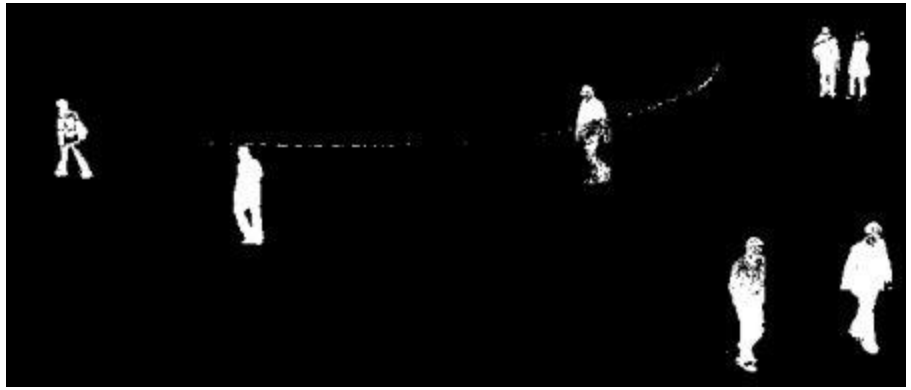
Objective:
Create a mask identifying foreground pixels

Required:
“Background” image to compare against

Method:
For every pixel in your new image, compare its value against the corresponding pixel from the background image. If they match, give it a value of 0. Otherwise, give it a value of 1.

To recover foreground elements:
Logical “AND” the new image with the mask.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('vtest.avi')

fgbg = cv2.createBackgroundSubtractorMOG()

while(1):
    ret, frame = cap.read()

    fgmask = fgbg.apply(frame)

    cv2.imshow('frame',fgmask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

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



What could go wrong here?

Background Subtraction

How can we make Background Subtraction more robust?

- Handle slight changes in illumination/lighting conditions
  - Instead of “Logical AND”…
  - …use a threshold to remove any pixels within a set distance
- Filter out high frequency, repetitive motion in the background (e.g., water ripples, tree branches)
  - Take a time-series of background images…
  - …and mask out any pixels that match any of the possible ‘backgrounds’
- Gracefully accommodate gradual shifts in background
  - Use background images sampled over a long time period
- Obtain background images even when we can’t get a clean photo
  - Take multiple images over a time window and find the mode pixel values (use median for environments with high, regular fluctuations)

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

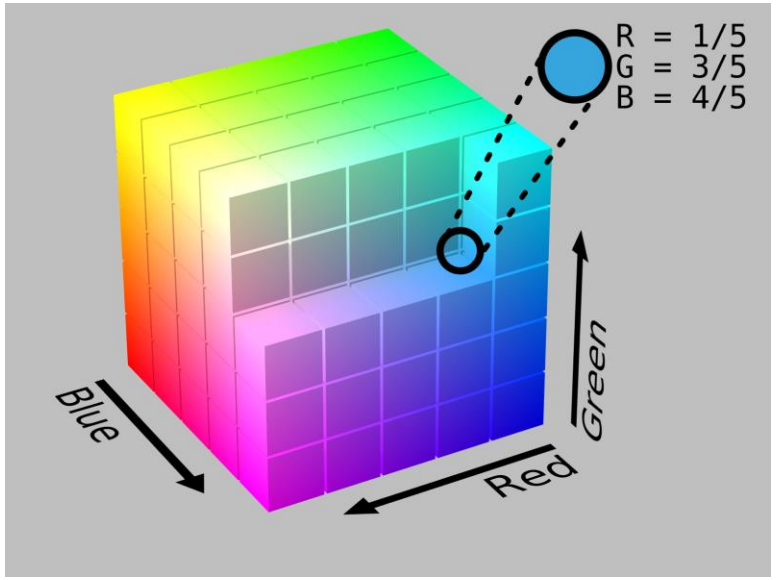
- Advantages
  - Easy to implement!
  - Reasonably fast, can be done in realtime
  - Works reasonably well for structured problems

- 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

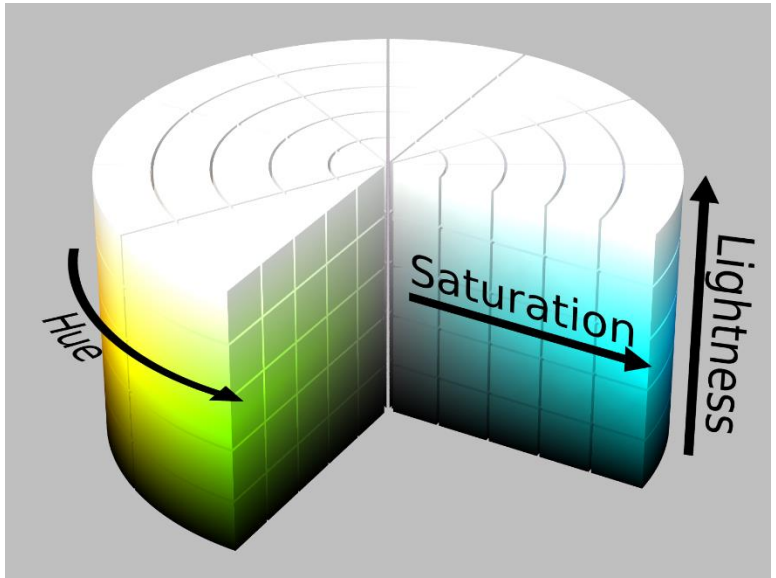
Quiz!

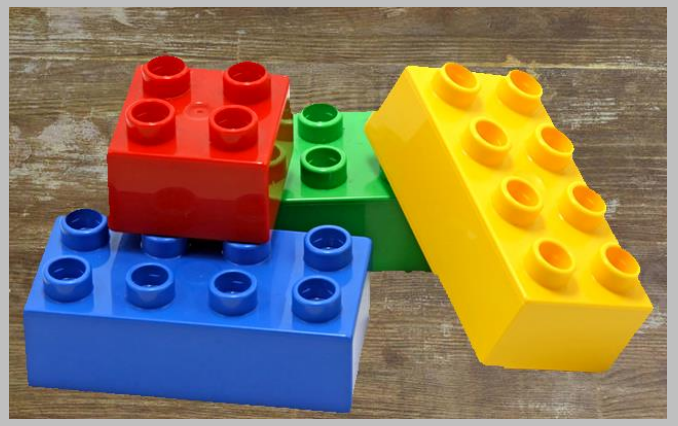
Color Filtering

RGB Space:



HSV Space:



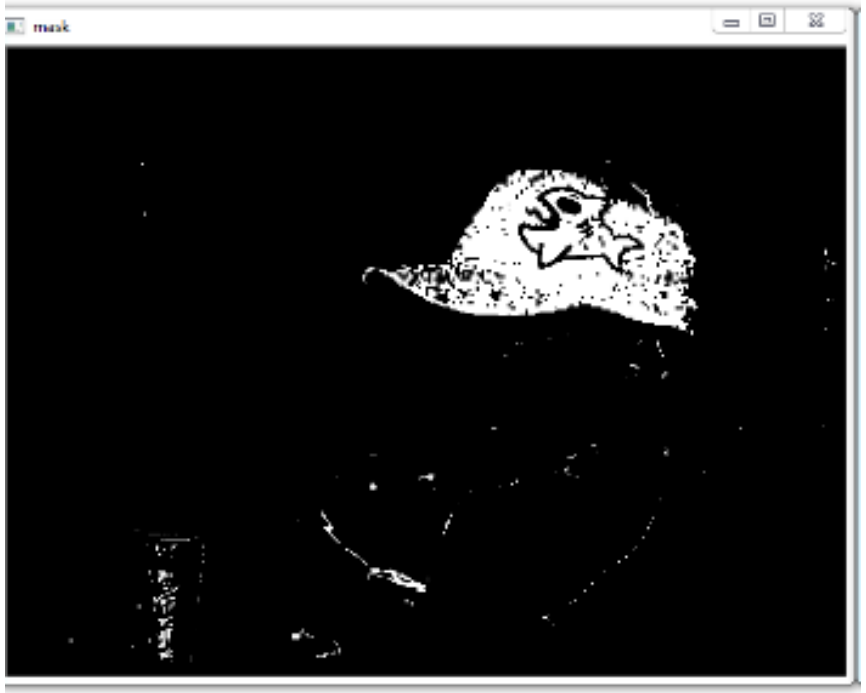


Color Filtering

- **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

```python
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



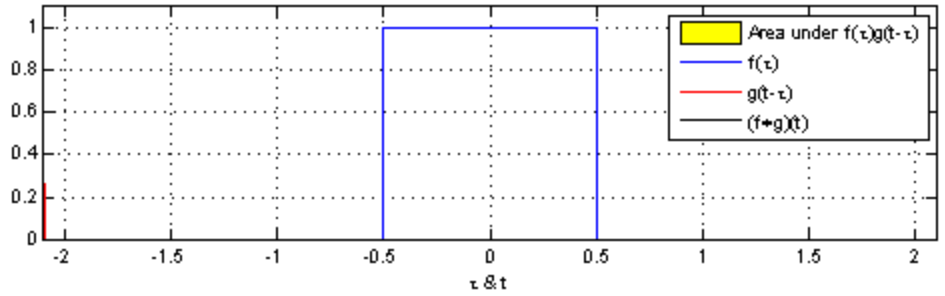
Color filtering often produces results like this, with regions of the target object getting filtered away

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



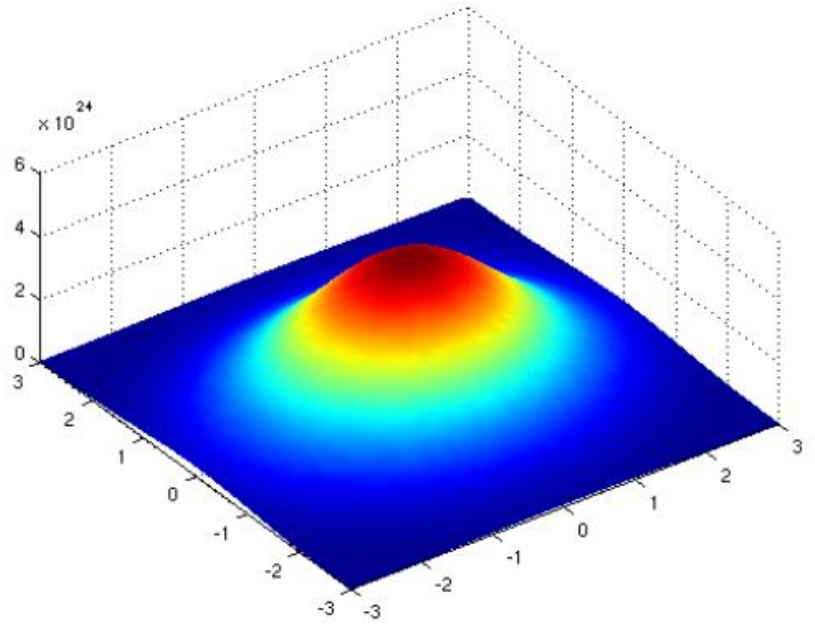
Center pixel will be averaged with its neighbors

Gaussian Blur

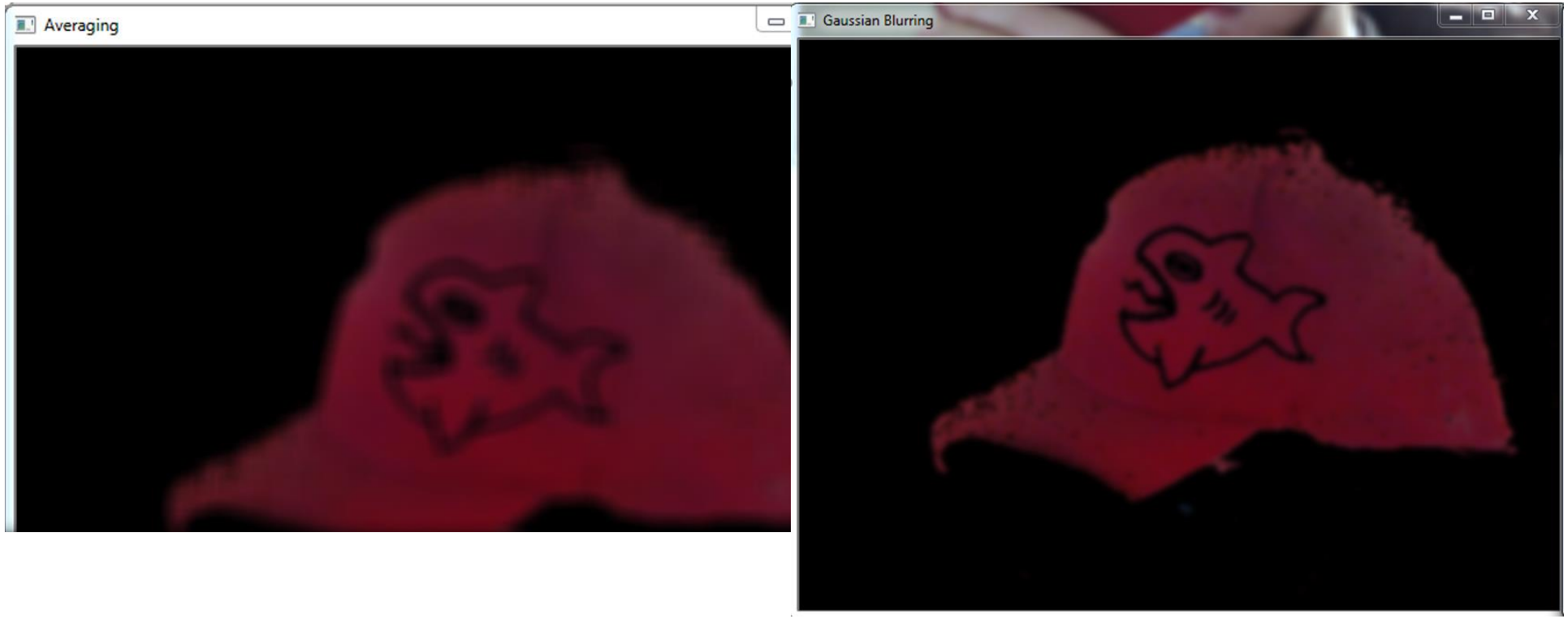
$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

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

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/

Object Detection

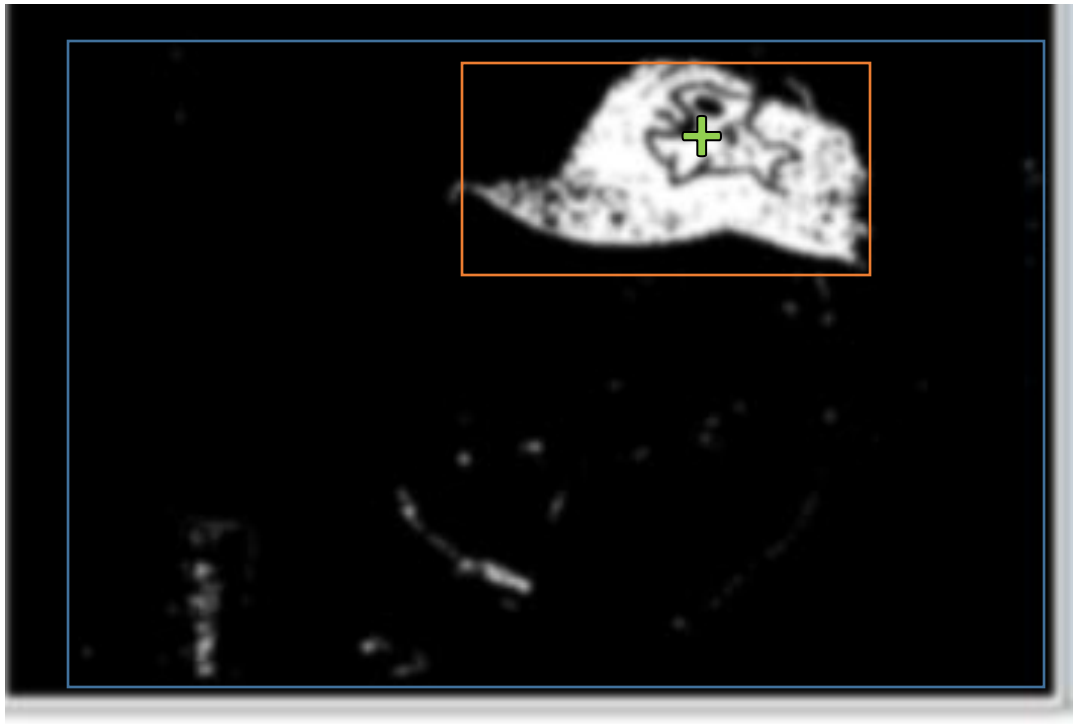
Now that we've been able to isolate our target objects, how do we figure out where they are in the scene?

Recurring Theme:
Simplify problem into a form addressable by established techniques

Blob Detection

Observations? Notice anything that could be problematic?

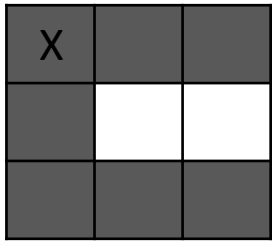
Blob Detection:
Often want to find Bounding Box and Centroid



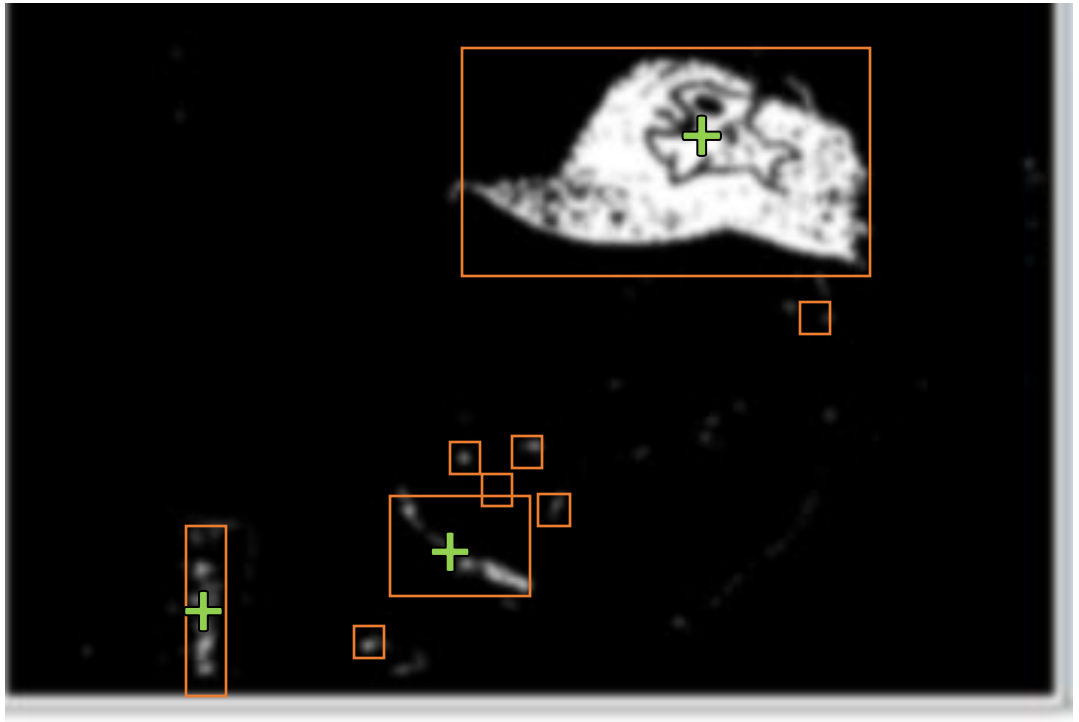
Blob Detection Concept:

- Given: Image mask (can be composited)
- Output: List of blobs “blobList[]” (lists of pixel coordinates)

1. Scan over each value in the image mask until finding a non-zero value at position (x,y).
2. Initialize tmpBlobList[]
3. Expand(x,y):
    1. If mask(x,y) is 0, return.
    2. Set mask(x,y) to 0, add (x,y) to tmpBlobList[]
    3. For each neighbor (nx, ny) of (x,y): Expand(nx,ny)
4. Add tmpBlobList to blobList[]

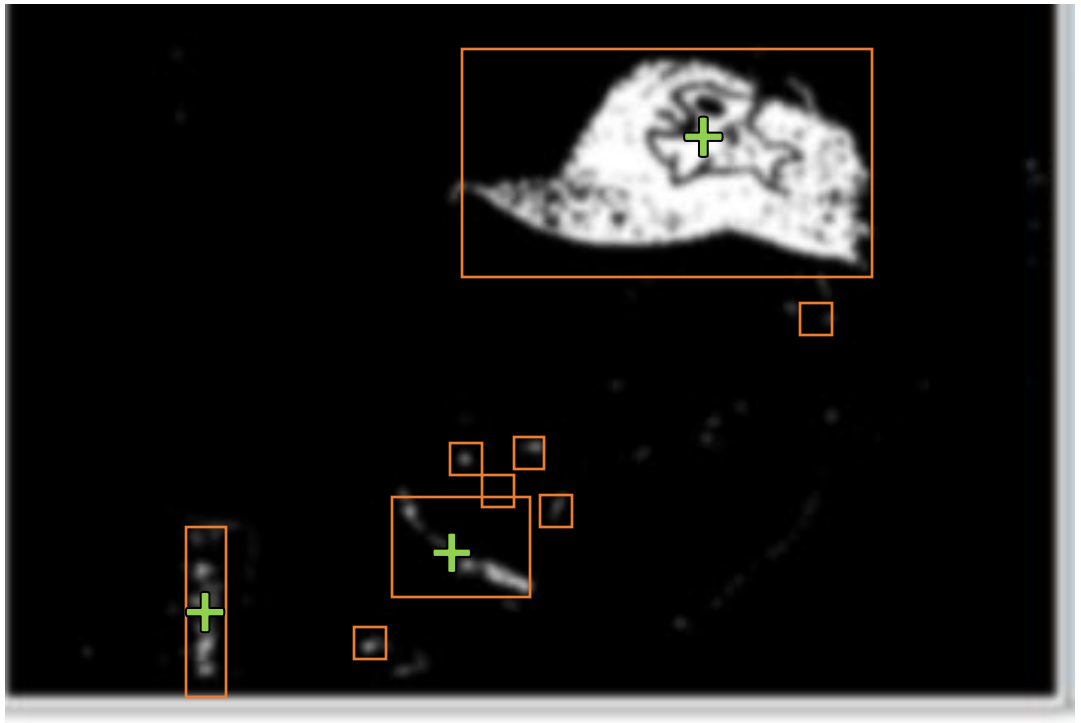


Making Blob Detection Work



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

Making Blob Detection Work



Bring more prior knowledge into the problem!

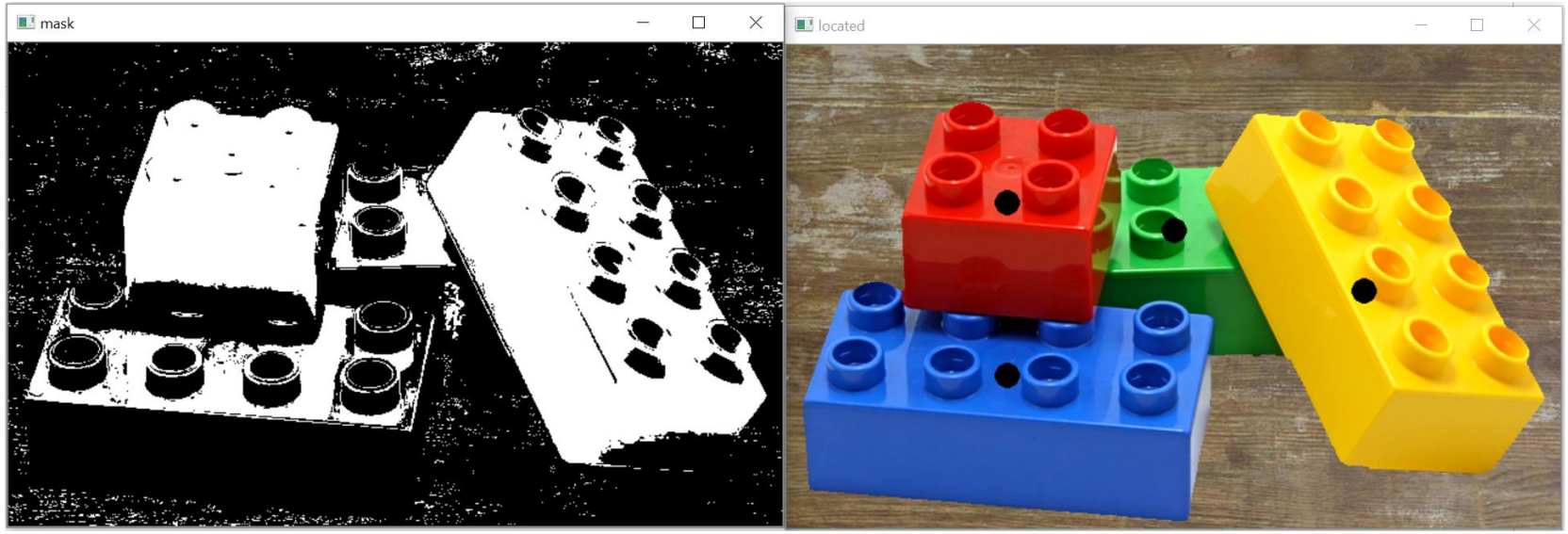
Currently using:

- Color

Not currently using:

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

Homework 3 (Color Filtering + Blob Detection): Results



Issue: Blobs are blending together
Fix: Use tighter color range bounds

Issue: Mask doesn't look right
Fix: Make sure that you're checking against the copy of *img_mask* in your *get_blobs* function!

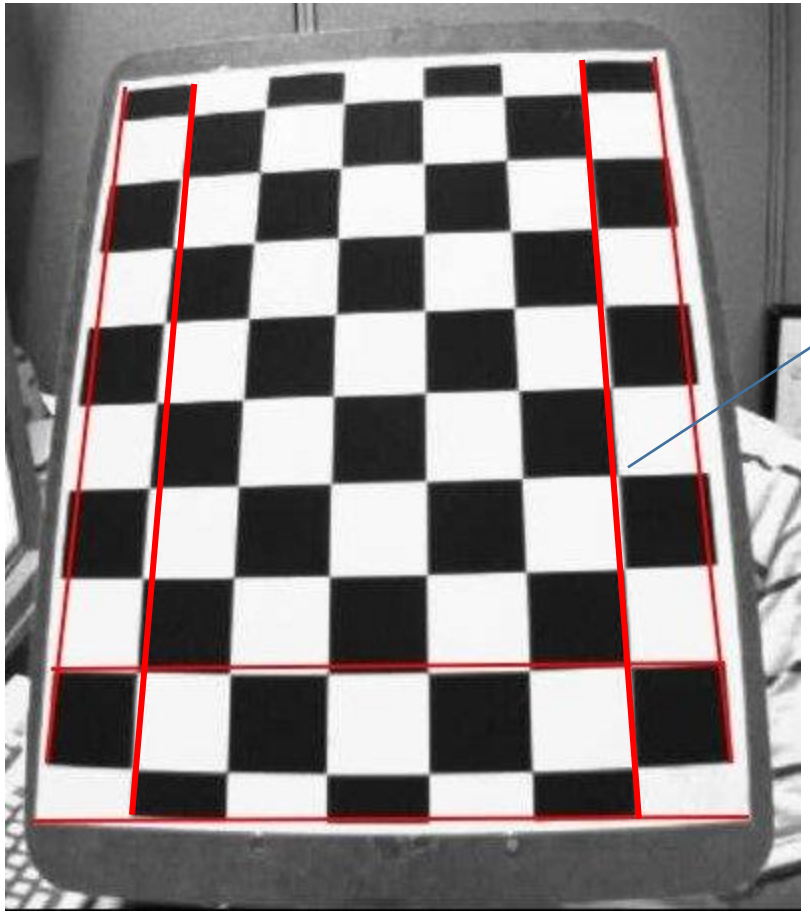
Issue: Mask *still* doesn't look right
Fix: Make sure your *do_color_filtering* function is passing in the pixel color, not the coordinates, to the color bounds check.

NGMI

Group Exercise: Estimating Traffic

How can we determine how many vehicles travel on this road every hour?

Regular Camera Image of Chessboard Pattern



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

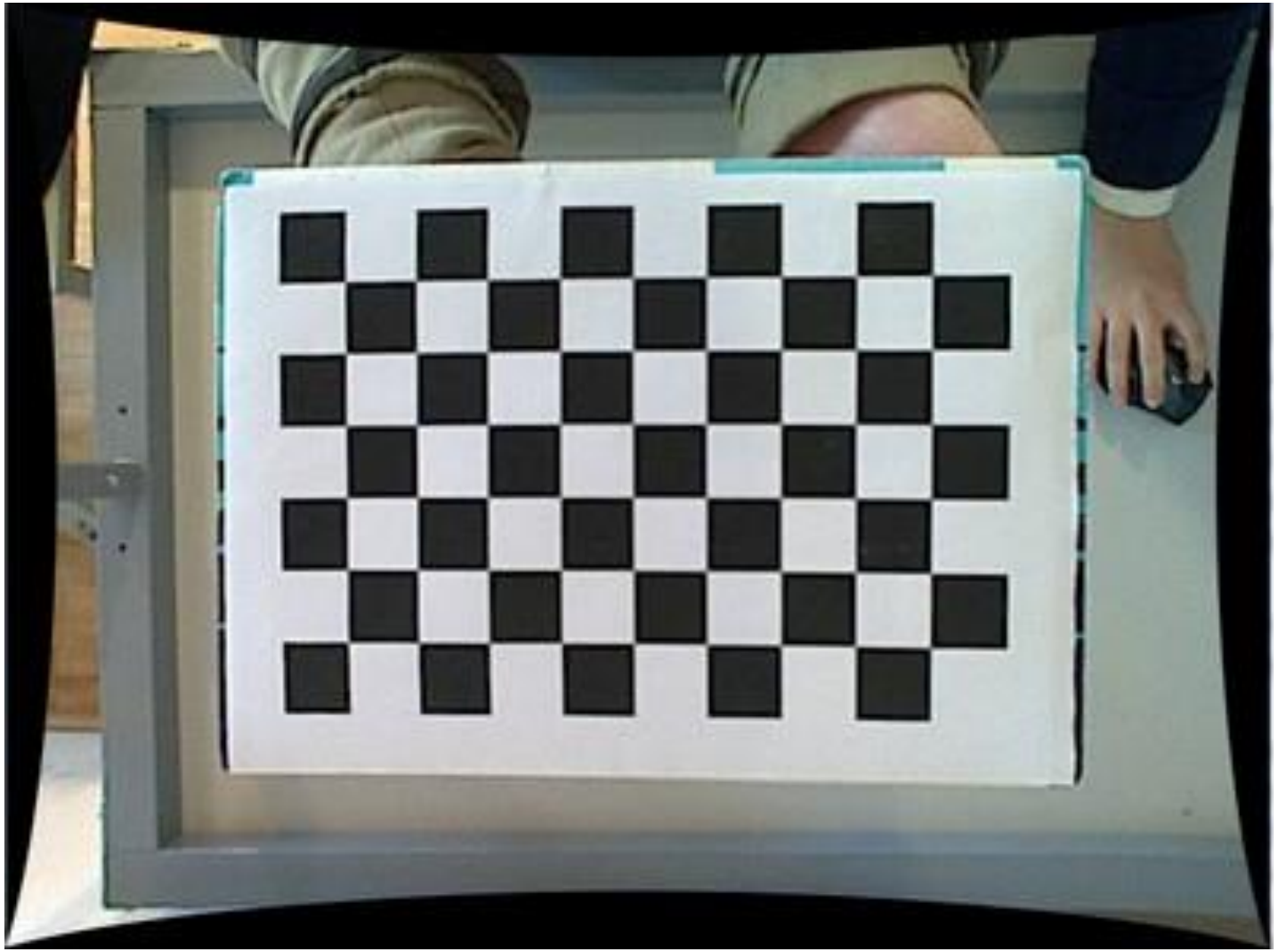
Pixels to Meters

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

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

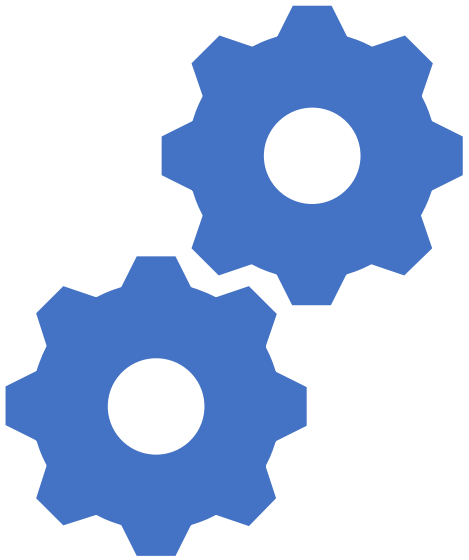
  - Must find distortion coefficients $(k_1, k_2, k_3, p_1, p_2)$

- 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



Intro. to Machine Learning

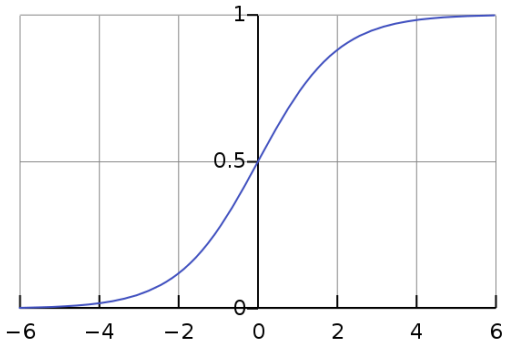
Introduction to Machine Learning

- 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)"

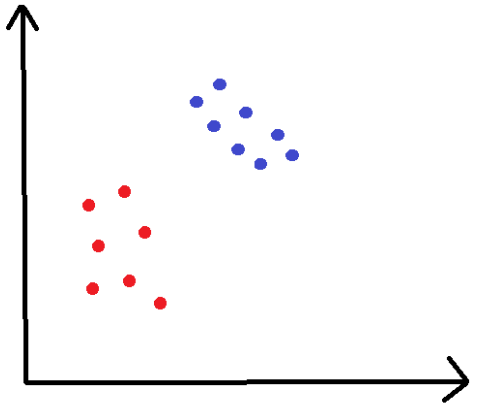
Unsupervised Learning

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

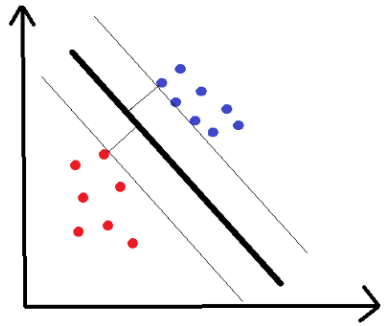
"You guess, I have no feedback to give you. Good luck!"

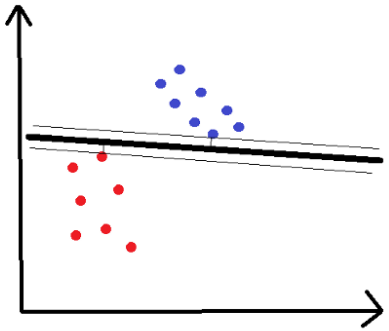
Supervised Learning: Support Vector Machines

- Is this dot red or blue?



- Draw a separating line!
  (Or hyperplane)



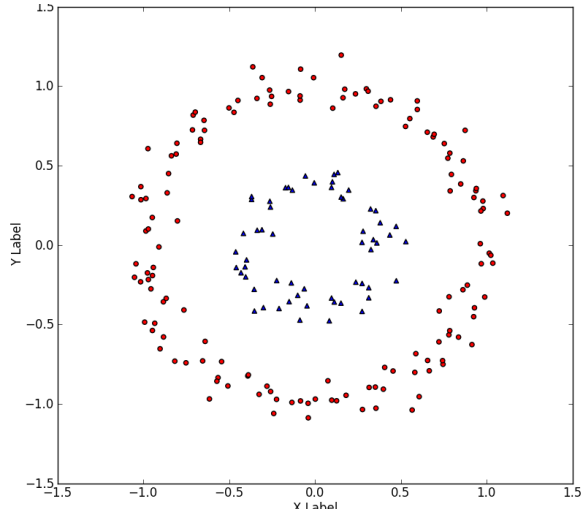


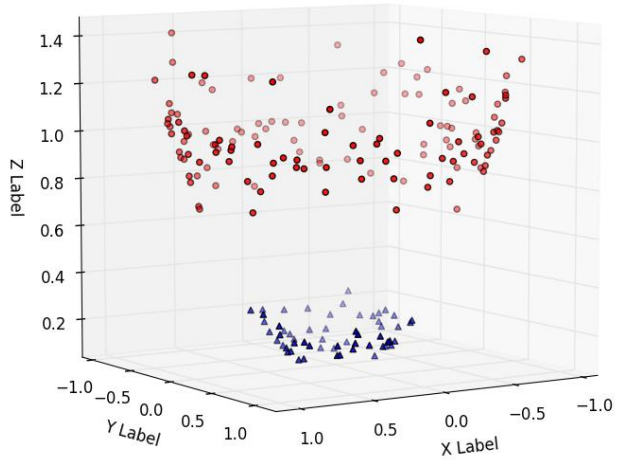
Supervised Learning: Support Vector Machines

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.





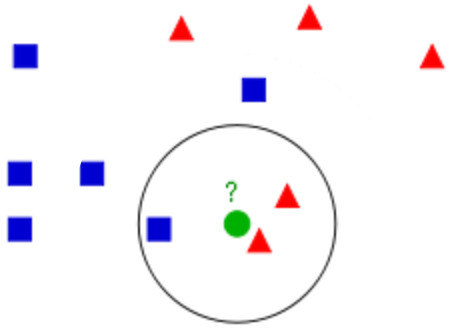
Supervised Learning: Support Vector Machines

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

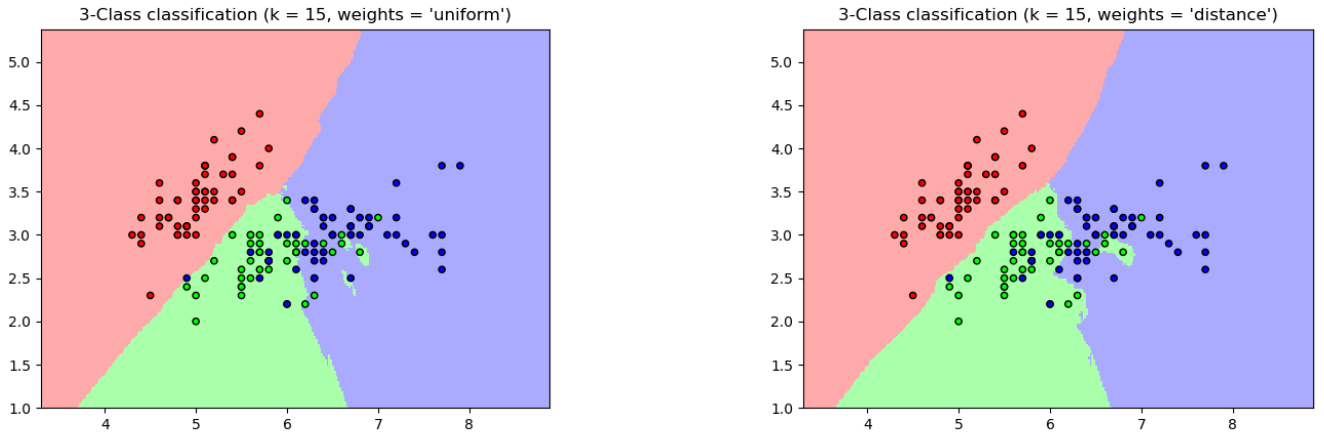
K-Nearest Neighbor

- 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



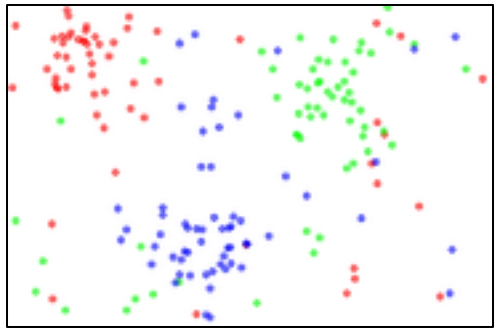
Weighted K-Nearest Neighbor

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

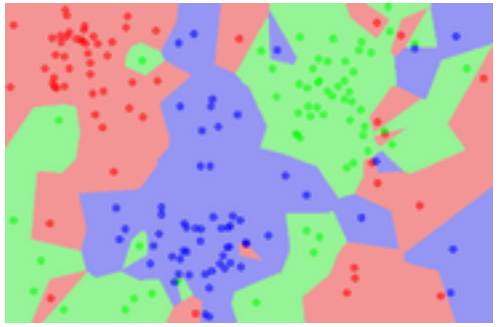


Reduced K-Nearest Neighbor

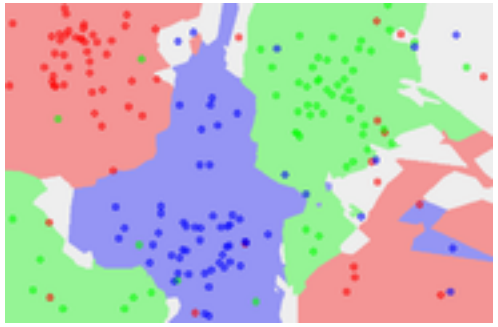
Dataset



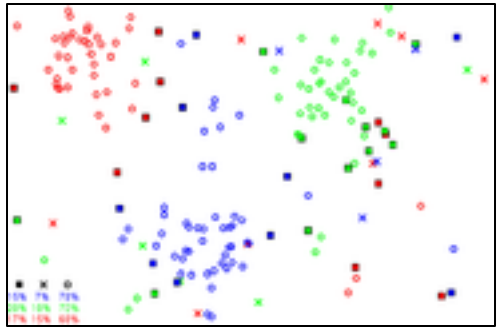
1-NN Classification



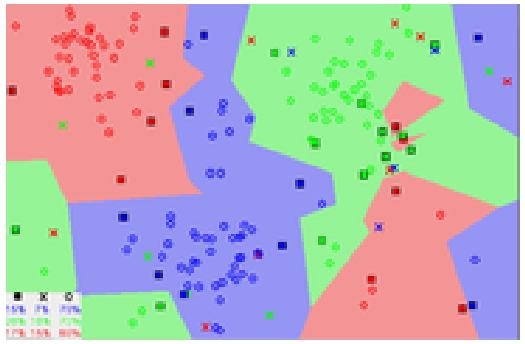
5-NN Classification



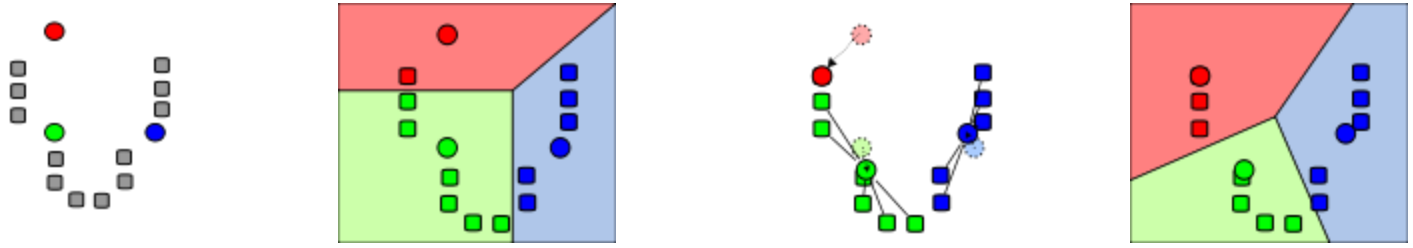
Reduced Dataset



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.