# Lightstick orientation; line fitting

**Prof. Ramin Zabih**

**http://cs100r.cs.cornell.edu**

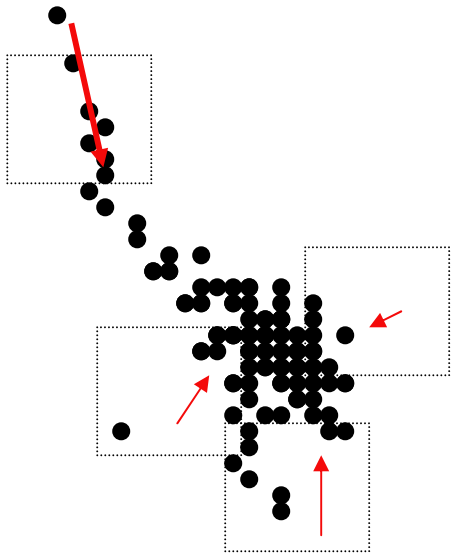
Cornell University
Computer Science

# Administrivia

- Assignment 4 out, due next Friday
- Quiz 5 will be Thursday Oct 18
- TA evals for Gurmeet and Devin
- Course midterm eval!
- Dense box algorithm, RDZ's way

# ♦ Mean shift algorithm

- Mean shift: centroid minus midpoint
  - Within a square

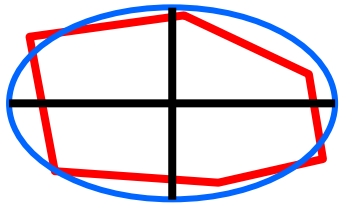


# Gift-wrapping algorithm

- Convex hull: smallest convex polygon containing the points
- Gift-wrapping:
  - Start at lowest point
  - Find new point such that all the other points lie to the left of the line to it
    - I.e., the largest angle
  - Repeat

# Computing orientation

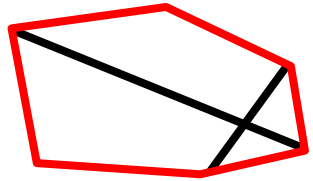
- We have a convex shape
  - Now what?



- We could fit the polygon with an ellipse
- Then use the major and minor axis
  - Major axis would tell us orientation
    - Small minor axis is a sanity check
  - Ellipse fitting will be a guest lecture on 11/20

# Polygon major/minor axis

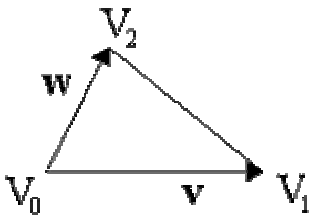
- We can look for the pair of vertices that are the farthest from each other
  - Call this the major axis



- Closest pair can be the minor axis
  - Or perhaps the closest pair on opposite sides of the major axis?

# Cross product leftness

- The area of a triangle is related to the cross product of the edge vectors
  - http://geometryalgorithms.com/Archive/algorithm_0101/




$$A(\Delta) = \tfrac{1}{2}|\mathbf{v}\times\mathbf{w}|$$
$$= \tfrac{1}{2}|(\mathbf{V}_1-\mathbf{V}_0)\times(\mathbf{V}_2-\mathbf{V}_0)|$$

$$(\mathbf{V}_1-\mathbf{V}_0)\times(\mathbf{V}_2-\mathbf{V}_0) = \left(0,0,\begin{vmatrix}(x_1-x_0) & (x_2-x_0)\\ (y_1-y_0) & (y_2-y_0)\end{vmatrix}\right)$$

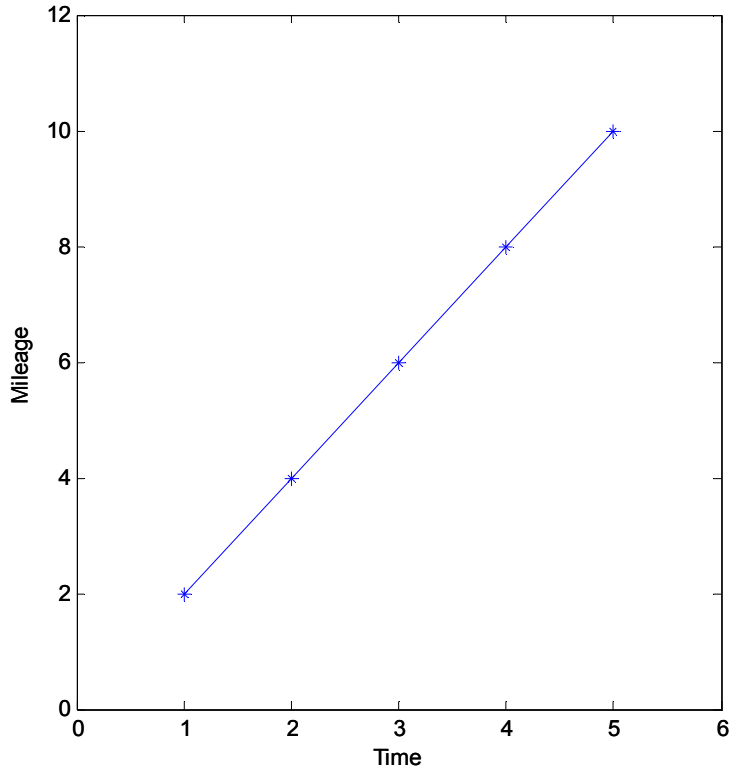
# A less smart test for leftness

- A pair of points defines a line $y=mx+b$
  - With a slope $m$ and intercept $b$
- Think of this as a way to predict a value of y given a value of x
  - We call x the independent variable
  - Example: x = date, y = DJIA
- If $m$ is positive and finite, what can we say about the points to the left of the line?
  - We have to be careful with directionality
  - Also: non-positive/non-finite cases

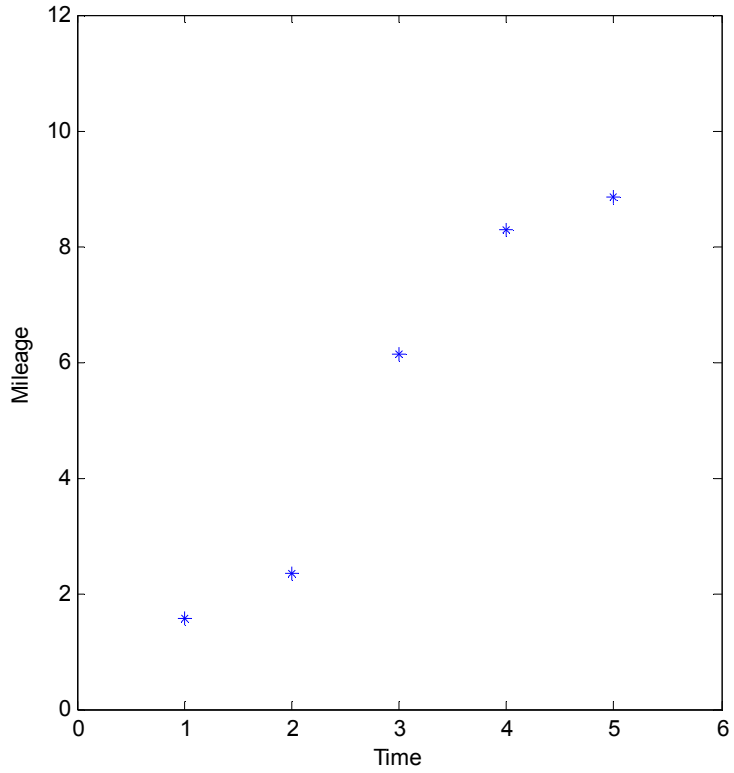
# New topic: robot speedometer

- Suppose that our robot can occasionally report how far it has traveled (mileage)
  - How can we tell how fast it is going?
- This would be a really easy problem if:
  - The robot never lied
    - I.e., it's mileage is always exactly correct
  - The robot travels at the same speed
- Unfortunately, the real world is full of lying, accelerating robots
  - We're going to figure out how to handle them

# The ideal robot



# The real (lying) robot

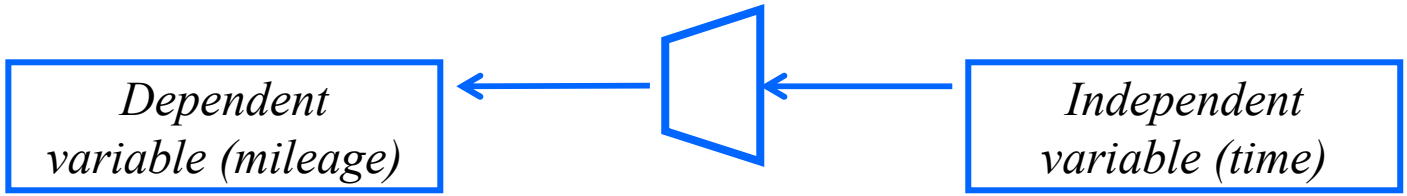


# Speedometer approach

- We are (as usual) going to solve a very general version of this problem
  - And explore some cool algorithms
  - Many of which you will need in future classes
- The velocity of the robot at a given time is the change in mileage w.r.t. time
  - For our ideal robot, this is the slope of the line
    - The line fits all our data exactly
- In general, if we know mileage as a function of time, velocity is the derivative
  - The velocity at any point in time is the slope of the mileage function

# Estimating velocity

- So all we need is the mileage function
- We have as input some measurements
  - Mileage, at certain times
- A mileage function takes as input something we have no control over
  - Input (time): independent variable
  - Output (mileage): dependent variable

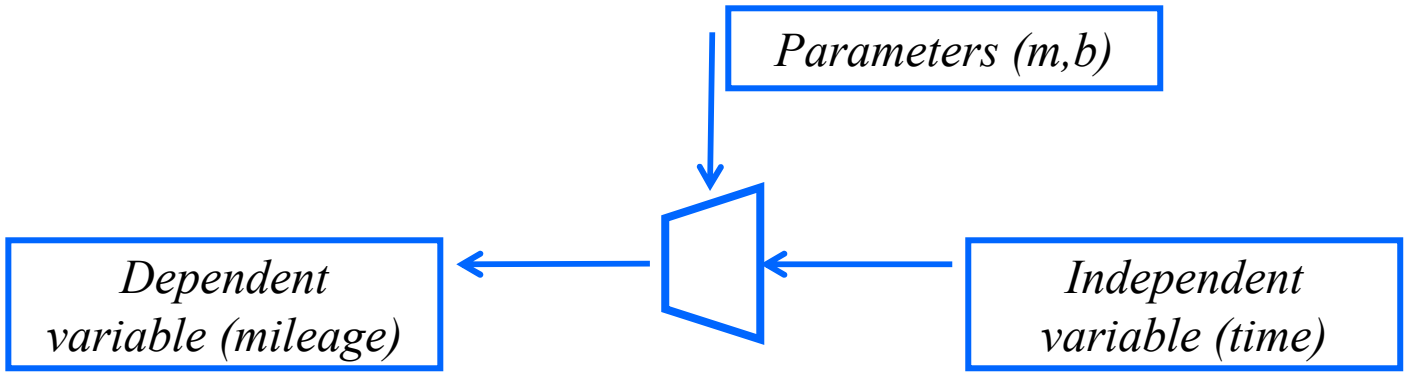


# Basic strategy

- Based on the data, find mileage function
  - From this, we can compute:
    - Velocity (1st derivative)
    - Acceleration (2nd derivative)
- For a while, we will only think about mileage functions which are lines
- In other words, we assume lying, non-accelerating robots
  - Lying, accelerating robots are much harder

# Models and parameters

- A **model** predicts a dependent variable from an independent variable
  - So, a mileage function is actually a model
  - A model also has some internal variables that are usually called parameters $\theta$
  - In our line example, parameters are *m,b*

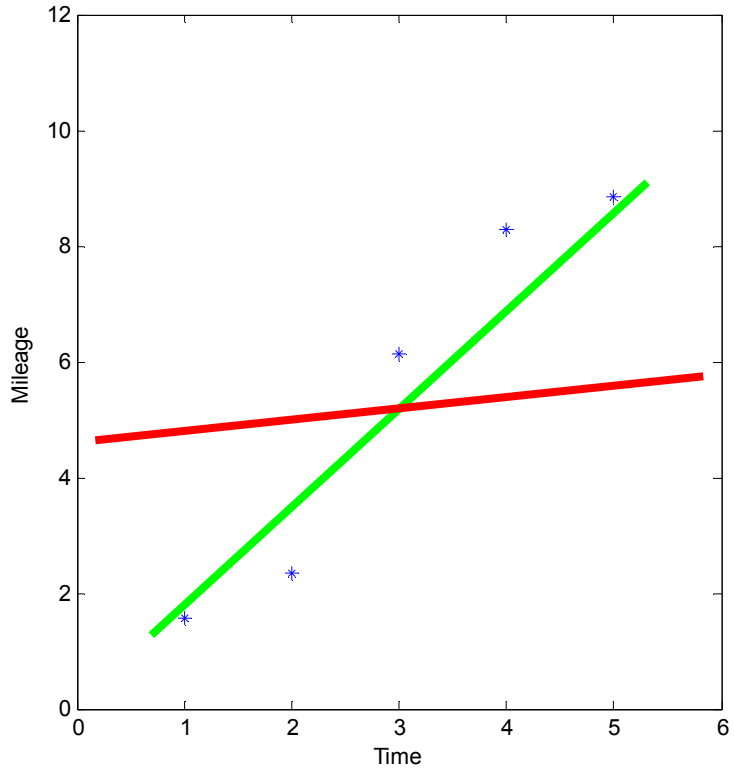


# How to find a mileage function

- We need to find the best mileage function
  - I.e., the best model
  - I.e., the best line (best m,b)
- We're going to define a function Good(m,b) that measures how much we like this line, then find the best one
  - I.e., the (m,b) that maximizes Good(m,b)
  - Such a function Good(m,b) is called a figure of merit, or an objective function
    - If you really want to impress your friends, you can tell them you're doing parameter estimation

# Figure of merit?

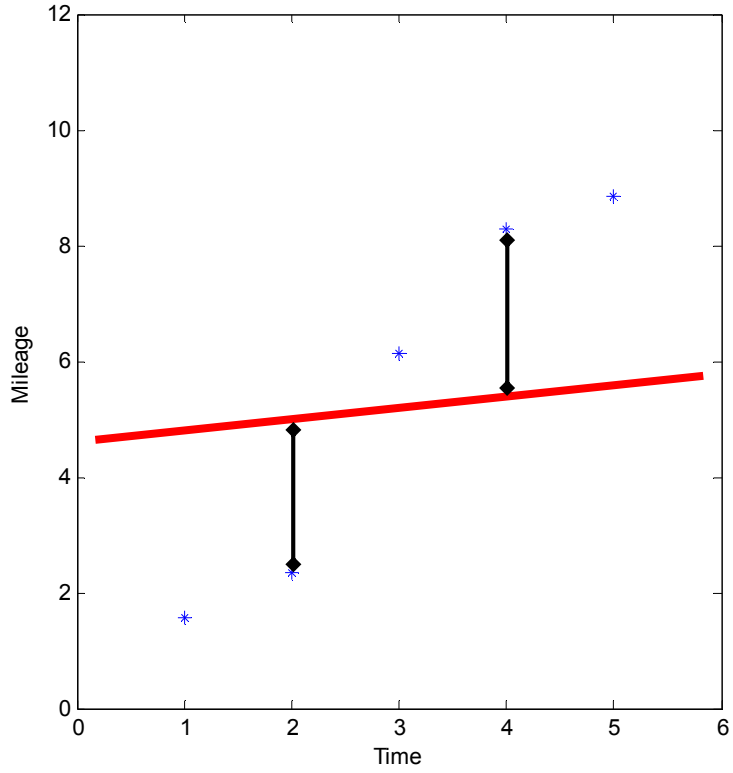
- What makes a line good versus bad?
  - This is actually a very subtle question



# Residual errors

- The difference between what the model predicts and what we observe is called a residual error (i.e., a left-over)
  - Consider the data point (x,y)
  - The model m,b predicts (x,mx+b)
  - The residual is y – (mx + b)
- Residuals can be easily visualized
  - Vertical distance to the line

# LS fitting and residuals




$$\text{Good}(m,b) = -\sum_i \left[ y_i - (mx_i + b) \right]^2$$