3D Object recognition

- Background
- n-point perspective algorithms
- Geometric hashing
- View-based recognition

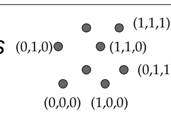
Background

- Recognition as pose estimation
- Object pose defines it embedding in three dimensional space
  - 3 degrees of positional freedom
  - 3 degrees of rotational freedom
- Underlying mathematical methods find applications in other areas of image analysis
  - camera calibration
  - image registration

Modeling polyhedral objects

- How can we represent a polyhedral object so that we could recognize it from an abritrary image?
  - Collection of positions of three dimensional points corresponding to the vertices of the polyhedra
  - Collection of edges that meet at those vertices
  - Colors of the planar facets that are bounded by edges
  - Shapes and colors of surface markings on the planar facets

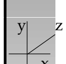
Modeling polyhedral objects



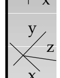
- Collection of 3-D points
  - coordinates of points are expressed in object coordinate system
  - when we see an image of the object this means that there is an instance of the object in the world
    - so, we can think of the object model as being transformed to the world coordinate system
    - think of the world coordinate system as a coordinate system used to describe locations of points in a workspace for a robot

Object and world coordinate systems

- What is the object to world coordinate system transformation?
  - it is a rigid body transformation
    - translation of the object
    - rotation of the object
    - it is called a rigid body transformation because translations and rotations do not change the distances between points - i.e., the set of points in the object and world coordinate systems are congruent.



Object and world coordinate systems

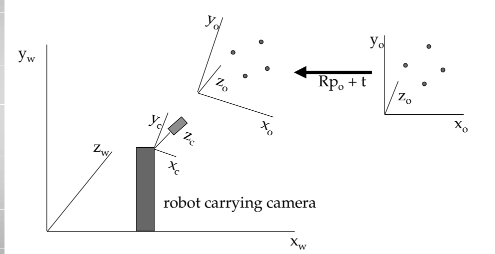


- Let $p_o = (x,y,z)^T$ be the coordinates of a point in the object coordinate system
  - we first rotate p using rotation matrix R that determine the orientation of the object in the world coordinate system: $p_R = Rp_o$
  - we then translate p_R by the translation vector, t, to determine its position in the world coordinate system: $p_w = Rp_o + t$

World and camera coordinates

- The camera coordinate system is another 3-D coordinate system in which
  - the x-y plane is the image plane,
  - the z axis is orthogonal to the image plane, and
  - the image plane is distance f from the center of projection, which is given coordinates (0,0,0)
- This is generally NOT the same coordinate system as the world coordinate system
  - we can place our camera anywhere in our workspace
  - in particular, it may be at the end of a robot arm that *moves* through the workspace
- But, we will assume these systems are aligned

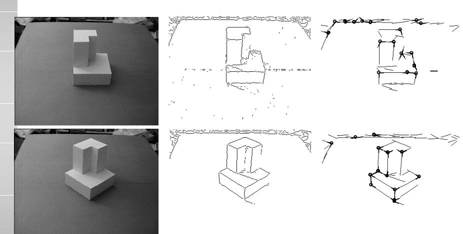
Coordinate frames



Choosing the points

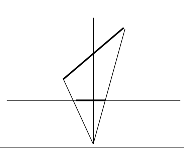
- Given a 3-D object, how do we decide which points from its surface to choose as its model?
  - choose points that will give rise to detectable features in images
  - for polyhedra, the images of its vertices will be points in the images where two or more long lines meet
    - these can be detected by edge detection methods
  - points on the interiors of regions, or along straight lines are not easily identified in images.

Example images



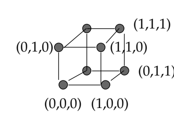
Choosing the points

- Example: why not choose the midpoints of the edges of a polyhedra as features
  - midpoints of projections of line segments are not the projections of the midpoints of line segments
  - if the entire line segment in the image is not identified, then we introduce error in locating midpoint



Modeling polyhedral objects - collection of edges

- Represent each edge by its end points
  - encode which edges meet at vertices
- Same rigid body transformation can be used to model the object to world coordinate transformation
  - line segment transforms to line segment defined by transformation of its end points
- But image analysis is now much harder - must find long edges in image



N- point pose recovery

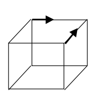
- Given:
  - a 3-D model organized as collection of points
  - Image of a scene suspected to include an instance of the object, segmented into feature points
- Goal
  - **Hypothesize** the pose of the object in the scene by matching (collections of) n model points against n feature points, enabling us to solve for the rigid body transformation from the object to world coordinate systems, and
  - **Verify** that hypothesis by projecting the remainder of the model into the image and matching
    - look for edges connecting predicted vertex locations
    - surface markings

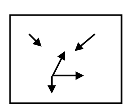
4-3-2-?

- 4 - point perspective solution
  - unique solution for 6 pose parameters
  - computational complexity of n^4m^4
- 3 - point perspective solution
  - generally two solutions per triangle pair, but sometimes more
  - reduced complexity of n^3m^3

4-3-2-?

- 1 - point perspective!
  - Object resting on a known ground place
  - Reduces problem to only 3 degrees of freedom
- Solved by matching a single oriented edge point from the image to an oriented model edge.
  - Complexity of O(mn)





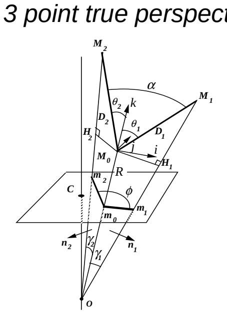
Reducing the combinatorics of pose estimation

- Big problem: we are looking for an object in an image but the image does not contain the object
  - then we would only discover this after comparing all n^4 quadruples of image features against all m^4 quadruples of object features.
- How can we reduce the number of matches?
  - consider only quadruples of object features that are
    - simultaneously visible - extensive preprocessing
    - diameter 2 subgraphs of the object graph
      - but in some images no such subgraphs might be visible

Reducing the cominbatorics of pose estimation

- Reducing the number of matches
  - consider only quadruples of image features that
    - are connected by edges
    - are "close" to one another
      - but not too close or the inevitable errors in estimating the position of an image vertex will lead to large errors in pose estimation
    - form diameter 2 subgraphs of the image vertex graph
  - more generally, try to group the image features into sets that are probably from a single object, and then only construct quadruples from within a single group

3 point true perspective

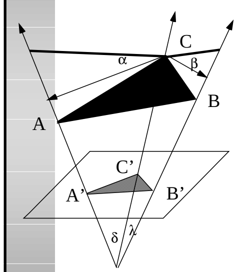


Exact Perspective

Triangle pose estimation algorithms

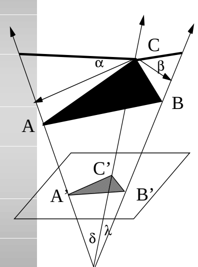
- There are two basic approaches to solving problems like pose estimation
  - Analytical methods based on constructing systems of equations and explicitly solving for unknown pose parameters
    - for triangle pose estimation this involves solving a quadratic equation in one pose angle, and then using the solutions to the quadratic equation to solve for remaining pose parameters
    - problem: errors in estimating location of image features can lead to either large pose errors or failure to solve the quadratic equation
  - Approximate numerical algorithms
    - finds solutions when exact methods fail due to image measurement error
    - more computation

Numerical method



- If distance, R, to C is known, then possible locations of A (and B) can be computed
  - they lie on the intersections of the line of sight through A' and the sphere of radius ||A-C|| centered at C
  - Once A and B are located, their distance can be computed and compared against the actual distance ||A-B||

Numerical Method



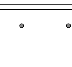
- Not practical to search on R since it is unbounded
- Instead, search on one angular pose parameter, α.
  - $R_a = R_c \cos \delta \pm ||A-C|| \sin \alpha$
  - $R_b = R_c \cos \lambda \pm ||B-C|| \sin \beta$
  - $R_c = ||A-C|| \cos \alpha / \sin \delta$
- This results in four possible lengths for side ||A-B||

Geometric hashing

- Consider the following simple 2-D recognition problem.
  - We are given a set of object models, M_i
  - each model is represented by a set of points in the plane
    - $M_i = \{P_{i,1}, \ldots, P_{i,ni}\}$
  - We want to recognize instances of these point patterns in images from which point features (junctions, etc.) have been identified
    - So, our input image, B, is a binary image where the 1's are the feature points
    - We only allow the position of the instances of the M_i in B to vary - orientation is fixed.
    - We want our approach to work even if some points from the model are not detected in B.

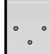
Geometric hashing

- We have already studied solutions to this problem
  - Correlation - match each of the M_i against B, finding locations at which the correlation is high
  - Hough transform - to speed up the basic correlation algorithm.
- Geometric hashing - we trade off preprocessing time for search time during recognition
  - we will search for all possible models simultaneously
  - correlation must search for them one at a time

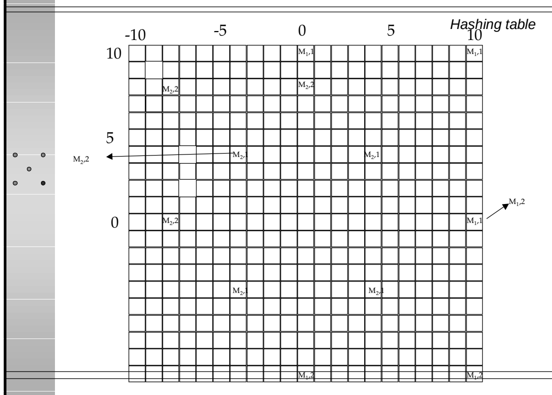


Geometric hashing

- Consider two models
  - $M_1 = \{(0,0), (10,0), (10,10), (0,10)\}$
  - $M_2 = \{(0,0), (4,4), (4,-4), (-4,-4), (-4,4)\}$



- We will build a table containing, for each model, all of the relative coordinates of these points given that one of the model points is chosen as the origin of its coordinate system
  - this point is called a basis, because choosing it completely determines the coordinates of the remaining points.
  - examples for M_1
    - choose (0,0) as basis obtain (10,0), (10,10), (0,10)
    - choose (10,0) as basis obtain (-10,0), (0,10), -10,10))



Hash table creation

- How many entries do we need to make in the hash table.
  - Mode M_i has n_i point
    - each has to be chosen as the basis point
    - coordinates of remaining n_i -1 points computed with respect to basis point
    - entry (M_i, basis) entered into hash table for each of those coordinates
  - And this has to be done for each of the m models.
  - So complexity is mn^2 to build the table
    - But the table is built only once, and then used many times.

Using the table during recognition

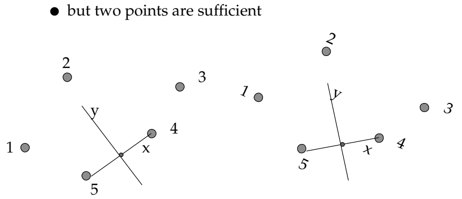
- Pick a feature point from the image as the basis.
  - the algorithm may have to consider all possible points from the image as the basis
- Compute the coordinates of all of the remaining image feature points with respect to that basis.
- Use each of these coordinates as an index into the hash table
  - at that location of the hash table we will find a list of (M_i, p_j) pairs - model basis pairs that result in some point from M_i getting these coordinates when the j'th point from M_i is chosen as the basis
  - keep track of the "score" for each (M_i, p_i) encountered
  - models that obtain high scores for some bases are recorded as possible detections

Some observations

- If the image contains n points from some model, M_i, then we will detect it n times
  - each of the n points can serve as a basis
  - for each choice, the remaining n-1 points will result in table indices that contain (M_i, basis)
- If the image contains s feature points, then what is the complexity of the recognition component of the geometric hashing algorithm?
  - for each of the s points we compute the new coordinates of the remaining s-1 points
  - and we keep track of the (model, basis) pairs retrieved from the table based on those coordinates
  - so, the algorithm has complexity $O(s^2)$, and is independent of the number of models in the database

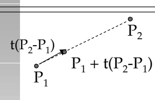
On to 3-D

- Consider the case where the point patterns can undergo not only translation, but also rotation
  - now one point is not a sufficient basis to compute the coordinates of the remaining points
  - but two points are sufficient



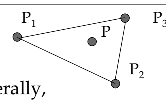
Revised geometric hashing

- Table constuction
  - need to consider all pairs of points from each model
    - for each pair, construct the coordinates of the remaining n-2 points using those two as a basis
    - add an entry for (model, basis-pair) in the hash table
    - complexity is now mn^3
- Recognition
  - pick a pair of points from the image (cycling through all pairs)
  - compute coordinates of remaining point using this pair as a basis
  - look up (model, basis-pair) in table and tally votes



Affine combinations of points

- Let P_1 and P_2 be points
- Consider the expression $P = P_1 + t(P_2 - P_1)$
  - P represents a point on the line joining P_1 and P_2.
  - if 0 <= t <= 1 then P lies between P_1 and P_2.
  - We can rewrite the expression as $P = (1-t)P_1 + tP_2$
- Define an **affine combination** of two points to be
  - $a_1P_1 + a_2P_2$
  - where $a_1 + a_2 = 1$
  - $P = (1-t)P_1 + tP_2$ is an affine combination with $a_2 = t$.



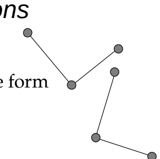
Affine combinations

- Generally,
  - if $P_1, \ldots P_n$ is a set of points, and $a_1 + \bullet\bullet\bullet a_n = 1$, then
  - $a_1 P_1 + \bullet\bullet\bullet\ a_n P_n$ is the point $P_1 + a_2(P_2-P_1) + \bullet\bullet\bullet\ a_n(P_n-P_1)$
- Let's look at affine combinations of three points. These are points
  - $P = a_1P_1 + a_2P_2 + a_3P_3 = P_1 + a_2(P_2-P_1) + a_3(P_3-P_1)$
  - where $a_1 + a_2 + a_3 = 1$
  - if 0 <= a_1, a_2, a_3, <=1 then P falls in the triangle, otherwise outside
  - (a_2, a_3) are the affine coordinates of P
    - "homogeneous" representation is $(1,a_2,a_3)$
  - P_1, P_2, P_3 is called the affine basis

Affine combinations

- Given any two points, $P = (a_1, a_2)$ and $Q = (a'_1, a'_2)$
  - Q-P is a vector
  - its affine coordinates are (a'_1-a_1, a'_2-a_2)
  - Note that the affine coordinates of a point sum to 1, while the affine coordinates of a vector sum to 0.

Affine transformations

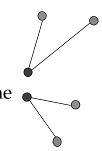


- Rigid transformations are of the form

$$[x', y'] = \begin{bmatrix} a & b \\ -b & a \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

where $a^2 + b^2 = 1$

- An affine transformation is of the form



$$[x', y'] = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

for arbitrary a,b,c,d and is determined by the transformation of three points

Representation in homogeneous coordinates

- In homogeneous coordinates, this transformation is represented as:

$$[x', y', 1] = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine coordinates and affine transformations

- The affine coordinates of a point are unchanged if the point and the affine basis are subjected to the same affine transformation
- Based on simple properties of affine transformations
- Let T be an affine transformation
  - $T(P_1 - P_2) = TP_1 - TP_2$
  - $T(aP) = aTP$, for any scalar a.

Proof

- Let $P_1 = (x,y,1)$ and $P_2 = (u,v,1)$
  - Note that $P_1 - P_2$ is a vector
- $TP_1 = (ax + by + t_x, cx + dy + t_y, 1)$
- $TP_2 = (au + bv + t_x, cu + dv + t_y, 1)$
  - $TP_1 - TP_2 = (a(x-u) + b(y-v), c(x-u) + d(y-v), 0)$
- $P_1 - P_2 = (x-u, y-v, 0)$
- $T(P_1-P_2) = (a(x-u) + b(y-v), c(x-u)+d(y-v), 0)$

Geometric hashing

- Let $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$ be an ordered affine basis triplet in the plane.
- Then the affine coordinates (α,β) of a point P are:
  - $P = \alpha(\mathbf{P_2} - \mathbf{P_1}) + \beta(\mathbf{P_3} - \mathbf{P_1}) + \mathbf{P_1}$
- Applying any affine transformation T will transform it to
  - $TP = \alpha(T\mathbf{P_2} - T\mathbf{P_1}) + \beta(T\mathbf{P_3} - T\mathbf{P_1}) + T\mathbf{P}_1$
- So, TP has the same coordinates (α,β) in the basis triplet as it did originally.

What do affine transformations have to do with 3-D recognition

- Suppose our point pattern is a planar pattern - i.e., all of the points lie on the same plane.
  - we construct out hash table using these coordinates, choosing three at a time as a basis
- We position and orient this planar point pattern in space <u>far from the camera</u> and take its image.
  - So, the t_z component of the model to world rigid transformation is large - I.e., the Z coordinates of the 3D object points are large.
  - the transformation of the model to the image is an affine transformation
  - so, the affine coordinates of the points in any given basis are the same in the original 3-D planar model as they are in the image.

Why is this true?

- If our model, M, is a planar point pattern, then in the object coordinate system the points are represented as
  - $P_i = (X_i, Y_i, 0)$
  - Just create the model in the Z=0 plane
- M is then placed into the camera 3-D coordinate system by some rigid transformation with its rotation matrix and translation vector.
- Let $p_i = (u_i, v_i)$ be the image of the rigidly transformed P_i. Then

$$u_i = f\frac{r_{11}X_i + r_{12}Y_i + t_x}{r_{31}X_i + r_{32}Y_i + t_z} \qquad v_i = f\frac{r_{21}X_i + r_{22}Y_i + t_y}{r_{31}X_i + r_{32}Y_i + t_z}$$

Remember

$$P_w = RP_o + T$$

$$[X_w, Y_w, Z_w] = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

$$X_w = r_{11}X_o + r_{12}Y_o + r_{13}Z_0 + t_x$$

And the image coordinates of (X_w, Y_w, Z_w) are

$$u = fX_w / Z_w = f\frac{r_{11}X_o + r_{12}Y_o + r_{13}Z_o + t_x}{r_{31}X_o + r_{32}Y_o + r_{33}Z_o + t_z}$$

Why is this true?

- Placing M "far" from the camera means that in the denominator of these expressions, t_z dominates. So we rewrite them as:

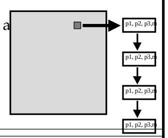
$$u_i = f\frac{r_{11}x_i + r_{12}y_i + t_x}{t_z} = \underbrace{[fr_{11}/t_z]}_{a}x_i + \underbrace{[fr_{12}/t_z]}_{b}y_i + \underbrace{t_x/t_z}_{t_1}$$

$$v_i = f\frac{r_{21}x_i + r_{22}y_i + ty}{t_z} = \underbrace{[fr_{21}/t_z]}_{c}x_i + \underbrace{[fr_{22}/t_z]}_{d}y_i + \underbrace{t_y/t_z}_{t_2}$$

- This is an affine transformation

Preprocessing

- Suppose we have a model containing m feature points
- For each ordered noncollinear triplet of model points
  - compute the affine coordinates of the remaining m-3 points using that triple as a basis
  - each such coordinate is used as an entry into a hash table where we record the (**base triplet,model)** at which this coordinate was obtained.
- Complexity is m^4 per model.



Recognition

- Scene with n interest points
- Choose an ordered triplet from the scene
  - compute the affine coordinates of remaining n-3 points in this basis
  - for each coordinate, check the hash table and for each entry found, tally a vote for the (basis triplet, model)
  - if the triplet scores high enough, we verify the match
- If the image does not contain an instance of any model, then we will only discover this after looking at all n^3 triples.

View Based Recognition

- Recognize a 3-D object by comparing an image to typical “pictures” of the object
  - will need many pictures of each 3-D object that cover different poses and scales
  - each such picture needs to be compared against every image window to determine if any match sufficiently well
  - correlation-like measures are used to compute similarity of a specific view to an image window

View Based Recognition

- Canonical problem: Face recognition
  - We are provided with a gallery of frontal images of faces we want to recognize
    - images in gallery have been normalized to fixed size
    - images have been normalized so that the centers of the left and right eyes are in standard positions
    - there is no background texture against which the faces are viewed.
  - Now, given an image of an unknown face
    - normalize it by finding the eyes and scaling/rotating the unknown image so that they are in standard positions
    - compare against each face in the gallery

Representing the gallery

- Typical gallery contains O(10,000) faces
  - this makes image correlation impractical
- Find a low dimensional representation for images
  - Character recognition - reduced a 50x50 character (2500 bits of information) to 7-8 features, each of which required ~32 bits of information (10:1 reduction)
  - Features were chosen in an ad hoc manner - no way to judge their quality other than to experiment with classification
- Image coding model
  - An image representation is good if it can be used to reconstruct a close approximation to the image it codes
  - Given two representations that reconstruct an image with the same accuracy, the one requiring fewer bits is preferable

Coding-based representations - Fourier transforms

- Fourier’s theorem:

Given any (well-behaved) one dimensional function, f(x), it is possible to represent the function as a weighted sum of sine and cosine terms of increasing frequency. The function, F(u), is the Fourier transform and describes the weights.

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i u x} dx$$

$$e^{-2\pi i u x} = \cos(2\pi i u x) - i \sin(2\pi i u x)$$

Fourier transforms

- Given F(u), it is possible to reconstruct f(x) using the formula:

$$f(x) = \int_{-\infty}^{\infty} F(u)e^{2\pi iux}\,du$$

- Typically, F(u) decreases with u
  - small u correspond to sines/cosines with low frequencies - they grossly encode large "objects" in the signal
  - large u correspond to high frequency sines/cosines - they encode fine detail in the signal
- For digital functions, the integrals are replaced with summations, and only discrete values of u are used
  - an approximation to f can be computed by "truncating" the reconstruction - using only a finite range of u

Fourier transforms

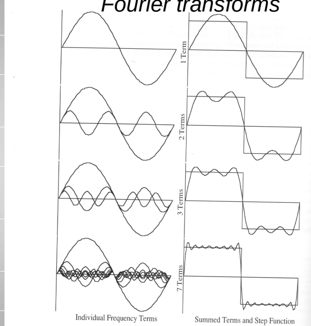


Figure 1. Summation of Fourier frequency terms to fit a simple step function.

Fourier transforms of 2-D images

- Sine and cosine functions are replaced by sinusoidal gratings. Each grating has
  - spatial frequency
  - orientation
- Image is then represented as a weighted sum of these "basis" functions
  - By truncating the summation, we get an approximation of the original image

Problems and solutions

- The inverse Fourier transform converges slowly to the original function
  - this means we need a lot of coefficients to obtain a good representation of the image
- There is nothing magical about the use of sines and cosines as a "basis" for image representation
  - any orthonormal and "complete" set of functions will do
  - natural basis of the m-dimensional images
- So, maybe we can find a set of basis functions that are good for representing a particular class of images
  - problem is solved by principal component analysis

Fourier transforms of 2-D images

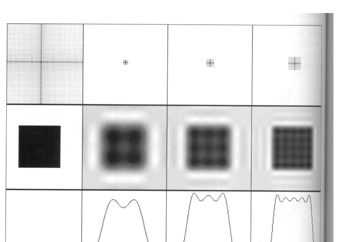
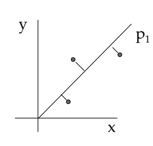


Figure 13. A two-dimensional step function and its frequency transform (left), and reconstructions with different numbers of terms (shown as a portion of the frequency transform). Bottom row shows horizontal line profiles through the center of the reconstructed spatial image.

Principal components analysis

- We are given:
  - a set of n "objects" (images)
  - each is represented, initially, by a set of m features (512^2) pixels
  - This data is organized as a (very large!) n x m matrix
- Let's look at a small example



- points are 2-D points
- we find the axis that most closely passes through these points
- if the axis passed exactly through these points, then we would need only one coordinate to represent each point.

Principal components analysis

- PC seeks the axis which the cloud of points are closest to
  - this is mathematically identical to finding the axis on which the variance of the point projections is greatest (that is, on which the projections are most spread out).
  - for high dimensional objects, like pictures, it is unlikely that there will be a single axis that passes close to all of the objects.
    - So, in this case, after we find the best axis (u_1), we then find the next best one (orthogonal to the first - u_2), and then the third best (u_3), etc.
    - Images are then represented by their projections on these axes: $v_i = I \bullet u_i$. This is exactly analogous to the Fourier transform, with the u_i replacing the sinusoids.

Principal components analysis

- If we compute m principal axes, then we can reconstruct any image exactly from its principal components representation:

$$I = \sum_{i=1}^{m} v_i u_i$$

- This is just another basis for the m-vector that represents the image
  - the original basis is the natural one - (1,0,0,...0), (0,1,0,...) ...
  - the principal axes represent just a rotation of the original high dimensional coordinate system

Principal components analysis

- However, we don't need to use all of the principal axes to obtain good reconstructions of the image.
- The mathematical procedure that determines the principal axes uses an eigenvector analysis, and associates a "score" with each axis
  - these scores correspond to the amount of variation in the image set that the axis corresponds to and are the eigenvalues of the procedure
  - The scores generally go to zero "quickly". For a face database, we can generally reconstruct a 512x512 face using only 80-100 principal axes with very small error.

Recognition using principal component analysis

- Given your gallery of images
  - compute its principal components
    - this is just a set of other images that are used as a basis for representing the images in the gallery
  - determine a k<<m such that the first k principal components are a "good" representation for the gallery
    - can be chosen based on the scores (eigenvalues) computed by the PCA
  - represent each image in the gallery by its projection on these k principal components
    - this is just the dot product of the image and the principal components.
    - each image now represented by k numbers

Recognition using principal component analysis

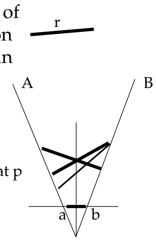
- Given an unknown image
  - compute its projection onto the principal component basis
    - this is a set of k numbers representing the unknown image
  - compare this k-tuple against each of the database image k-tuples
    - simple L^2 norm
    - sometimes each component is weighted by the associated eigenvalue

Challenges to appearance-based vision

- Variations in lighting
- Occlusion
  - addressed by the use of robust estimation for computing projections onto principal axes
- Normalization
  - for size, position and orientation within the image
- Large number of images in gallery for viewpoint independence
- Modeling within-class variations
- Rejection criteria

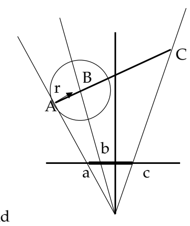
A 2-D object to 1-D image example

- How many points do we need to see in the 1-D image of a 2-D line segment of **known length** to recover the position and orientation of the line segment in the plane?
  - 1 - clearly not enough
  - 2 - no
    - choose a point, p_A on ray A
    - draw the circle of radius r centered at p
    - if P is not too far away from O, then this circle will intersect ray B at two points, p_{B1} and p_{B2}
    - ab is the image of both $p_A p_{B1}$ and $p_A p_{B2}$



A 2-D object to 1-D image example

- How many points?
  - 3
    - $B = A + k_1 r$
    - $C = A + k_2 r$
    - r is a unit vector of unknown direction defining the pose of the line ABC
    - since ABC are collinear, the circles of radii AB and BC centered at B must be tangent to the rays aA and cC



Hough transform for object recognition

- Hypothesize-and-test approach to recognition
- Hypotheses generated by clustering pose estimates from 3-point perspective solutions
- Brute force algorithm:
  - set up 6-D pose Hough array
  - For all n^3m^3 pairings of triples of image features to triples of object features compute possible poses using triangle pose algorithm and increment counters in Hough array
  - Possible correct poses will correspond to "clusters" of high votes in the 6-D pose Hough array, obtained by searching through Hough array and finding 6-D neighborhoods with high total counts

Hough transforms for pose estimation

- Key subproblems:
  1) Representing the parameter space
     - impractical to use a 6-D array to represent all possible poses
  2) Employing geometric constraints to filter clusters
     - in this more complex problem there is no guarantee that the image triangle/object triangle pairings that vote for a specific pose will be consistent

Representing the parameter space

- Impractical to cluster directly in a six dimensional clustering array
  - too much storage is required, even with variable resolution techniques
  - too much computation associated with clustering
  - clusters too spread out due to various sources of error
- Proposed solution - represent only a lower dimensional projection of the 6 dimensional space

A 3-D projection

- Two parameters correspond to the line of sight to the object centroid
  - solving triangle pose can be used to compute (x_c, y_c, z_c) - the location of object centroid in world coordinate system
  - its projection onto the image is then $(-fx_c/z, -fy_c/z)$, where f is the focal length of the camera
    - can regard this as the line of sight to the "center" of the object

A 3-d projection

- Third coordinate is the apparent size of the object in the image
  - if actual size of object is h, then its apparant size is fh/z
  - h corresponds to largest distance between 3-D model vertices

Pruning false triangle matches using geometric constraints

- Store list of matching triangle pairs at each cell of clustering array
- **Constraint 1** : eliminate duplicate matches of the same triangle pair
  - occurs because of nonuniqueness of triangle pose
- **Constraint 2**: eliminate pairs in which model triangle could not be visible
  - face normal might point away from image
  - both triangles meeting at a concave edge must be visible for the edge to be visible
  - visibility analysis eliminates about 50% of the false triangle pairs

Uniqueness of image feature to object vertex mapping

- Let T be the set of triangle pairs at a point in the clustering array
  - each model vertex can be matched to only one image feature
  - each image feature is the image of a unique model vertex
- f_{ij} - the frequency of pairing model vertex i to image feature j.
  - correct matches should have large f_{ij} because image features ordinarily belong to many triangles

Pruning triangles using uniqueness

- For each image feature i, choose the model vertex with highest f_{ij}
- Let P be the set of resulting image feature - model vertex pairings
  - eliminate from T any triangle pair with a pairing inconsistent with P
  - eliminate from T any triangle pair that does not contain at least one pairing from P
- Finally, eliminate from T any pair of triangles pairs with mutually inconsistent pairings

Experimental results

- Example- single polyhedral object with 12 vertices, noncluttered background
- Clusters correspond to maximal 3x3x3 neighborhoods of clustering array

Cluster	Triangles	Point Pairs
Orig	Final	Final
60	22	12
50	1	3
44	0	0
38	2	4
36	0	0

Scaled orthographic projection - definitions and pose estimation

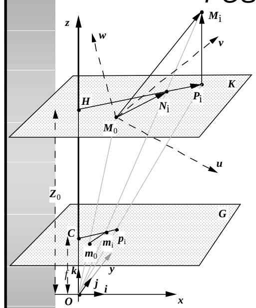
- $M_0, M_1, ..., M_i, ...M_n$ are the object feature points
  - M_0 is called the reference point
  - object frame of reference is $M_0\mathbf{u}$, $M_0\mathbf{v}$, $M_0\mathbf{w}$
  - $(\mathbf{U}_i, \mathbf{V}_i, \mathbf{W}_i)$ are the known coordinates of M_i in the object frame of reference.
- In Scaled orthographic projection (SOP) we assume the range to all object points is Z_0, the range to M_0.
  - SOP coordinates of p_i, the SOP image of M_i are
    - $x'_i = fX_i/Z_0$
    - $y'_i = fY_i/Z_0$

Notation

- The perspective image, m_i, of this point is
  - $x_i = fX_i/Z_i$
  - $y_i = fY_i/Z_i$
- The ratio $s = f/Z_0$ is called the **scaling factor** of the SOP
  - $x'_i = fX_i/Z_0 + f(X_i-X_0)/Z_0 = x_0 + s(X_i-X_0)$
  - $y'_i = y_0 + s(Y_i - Y_0)$
- The rotation matrix R for the object is the matrix whose rows are the coordinates of the unit vectors **i,j,k** from the camera coordinate system expressed in the object coordinate system (M_0**u**, M_0**v**, M_0**w**).

$$R = \begin{bmatrix} i_u & i_v & i_w \\ j_u & j_v & j_w \\ k_u & k_v & k_w \end{bmatrix}$$

POSE from SOP



- SOP coordinates of M_i are:

$$x_i - x_0 = I \bullet M_0M_i = [U_i V_i W_i] \times \begin{bmatrix} I_u \\ I_v \\ I_w \end{bmatrix}$$

$$y_i - y_0 = J \bullet M_0M_i = [U_i V_i W_i] \times \begin{bmatrix} J_u \\ J_v \\ J_w \end{bmatrix}$$

- I = si, J = sj, $s = f/Z_0$
- I and J can be recovered by solving system of linear equations from n point correspondences
- Unscaling I and J gives rotation matrix and distance, Z_0, to M_0.

Geometric Hashing

- Recognition of flat objects
  - depth variation within object small compared to distance of object from camera and focal length of camera
- Perspective is then well approximated by parallel projection with a scale factor
- Two different images of the same flat object are in **affine 2-D correspondence**
  - there is a nonsingular 2x2 matrix **A** and a 2-D translation vector b such that each point x in the first image is transformed to **A**x + **b**