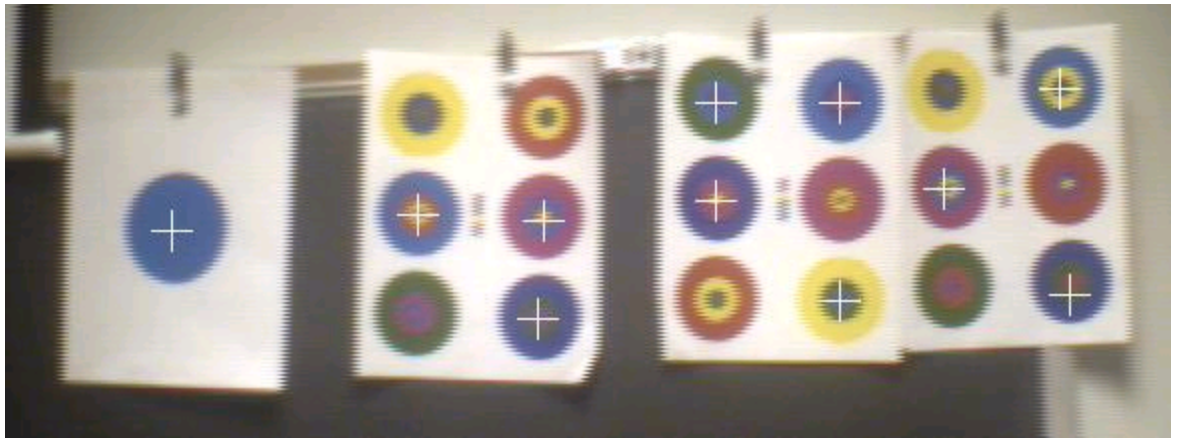
Computer Vision Projects

B.J. Guillot (Presentation Date 1/24/2001)

- May 1999 - Fiducial Tracking



- Summer 2000 - Exploration of POSIT algorithm in OpenCV

Fiducial Tracking references

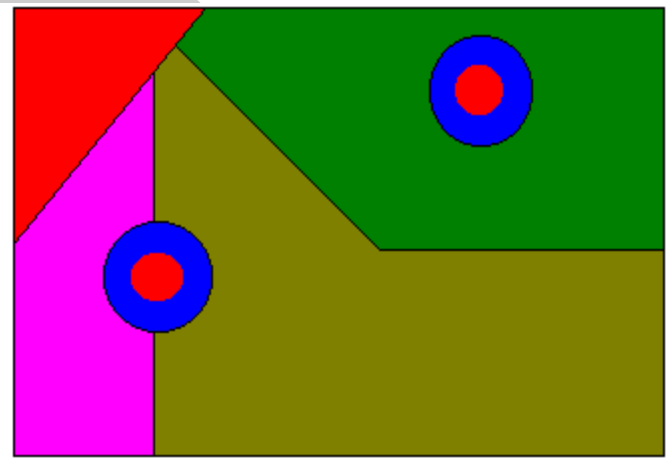
- **[CHO98]** Cho, Youngkwan, Jongweon Lee, and Ulrich Neumann. "A Multi-Ring Color Fiducial System and a Rule-Based Detection Method for Scalable Fiducial-Tracking Augmented Reality". November 1998. http://www.usc.edu/dept/CGIT/papers/cho-lee-IWAR98.pdf

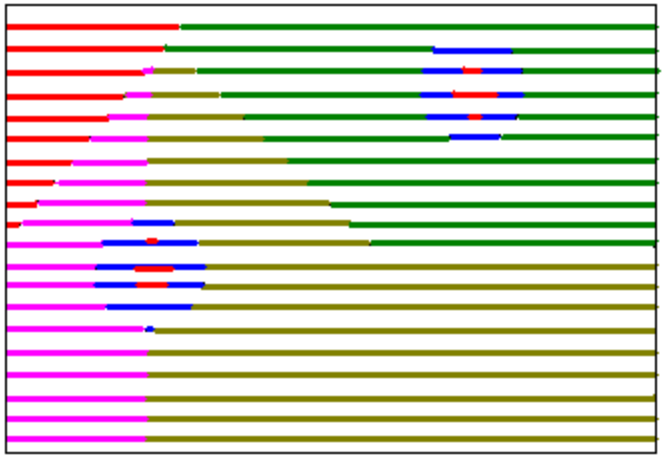
- **[CHO97]** Cho, Youngkwan, Jun Park, and Ulrich Neumann. "Fast Color Fiducial Detection and Dynamic Workspace Extension in Video See-Through Self-Tracking Augmented Reality".October 1997. http://www.usc.edu/dept/CGIT/papers/PacificGraphics98-AR.pdf

Processing Flow

- Grab a frame from the camera
- Create horizontal line segments every N horizontal scan lines (“low resolution pass”)
- Grow regions around “blue” line segments
- Form bounding rectangles around these coarse regions
- Refine the region size and position using full pixel count inside the bounding rectangles (“high resolution pass”)
- Square regions are the targets
- Loop back to the top

Horizontal Line Segments





- This function scans every n'th horizontal scan line looking to break the lines up into segments of "like" colors
- The segments are broken with a form of edge detection using a color distance metric based upon the square of the color angle cosine

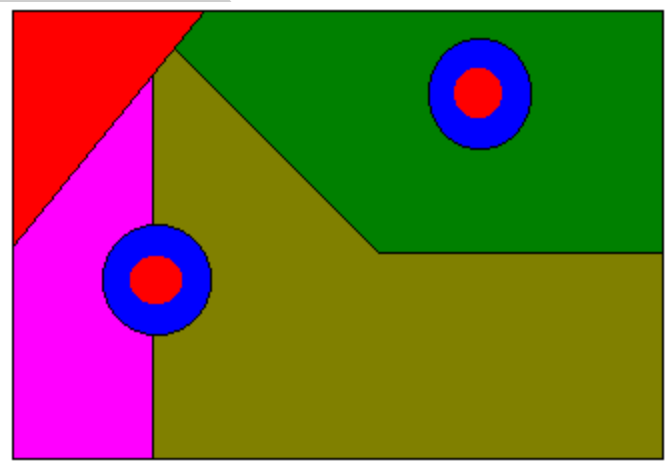
$$\cos^2(\theta) = \frac{(color_1 \bullet color_2)^2}{\|color_1\|^2 \|color_2\|^2}$$

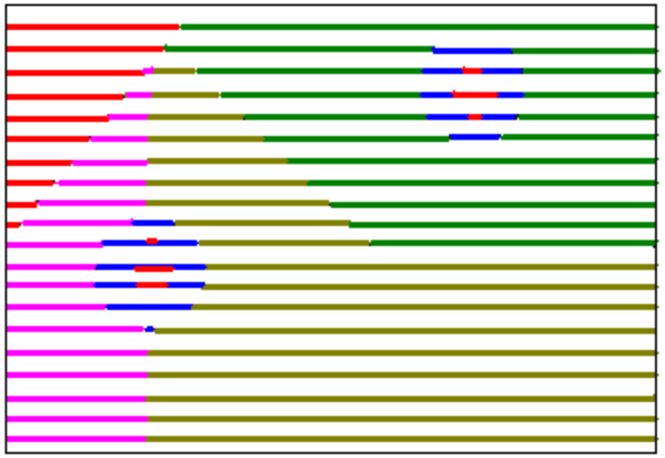
Square of the color angle cosine pseudo-code implementation

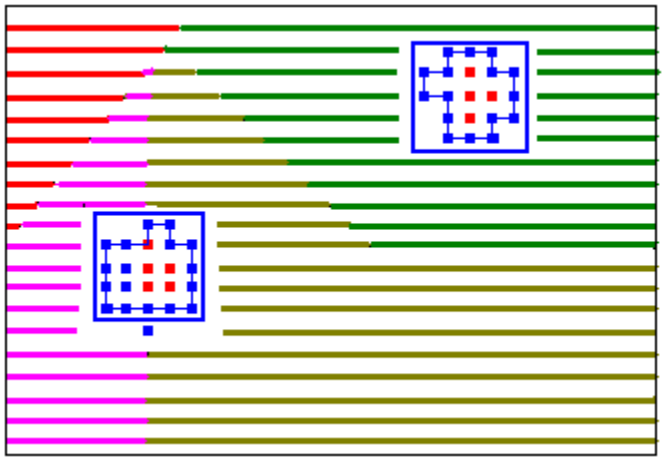
```
Boolean hitEdge(inputs: left and right color components)
{
  numerator =
    (leftRed*rightRed + leftGreen*rightGreen + leftBlue*rightBlue)^2

  denominator =
    (leftRed^2 + leftGreen^2 + leftBlue^2) *
    (rightRed^2 + rightGreen^2 + rightBlue^2)

  if (denominator == 0) return(FALSE)

  distance = numerator / denominator

  if (distance < DISTANCE_THRESHOLD) return(TRUE) else return(FALSE)
}
```

$$\cos^2(\theta) = \frac{(color_1 \bullet color_2)^2}{\|color_1\|^2 \|color_2\|^2}$$

Region growing & Bounding Rectangles

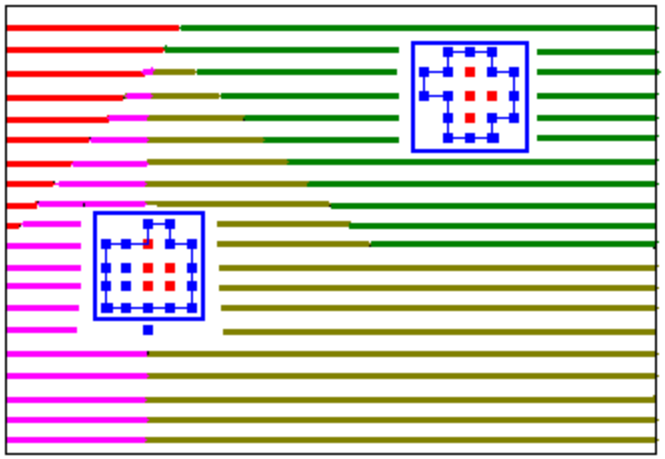


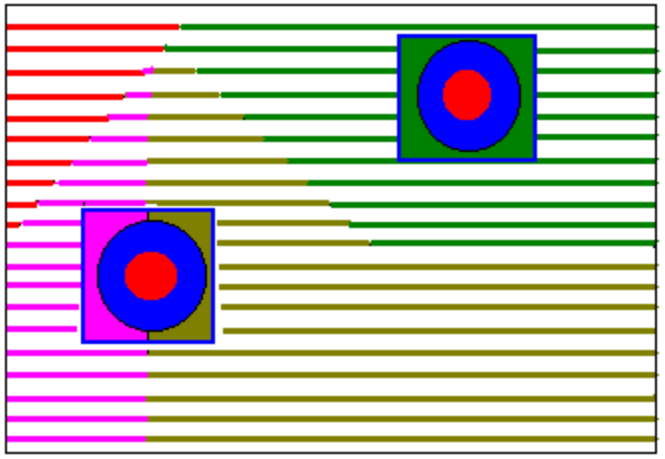




- The system was designed to look for regions that had a large blue intensity component

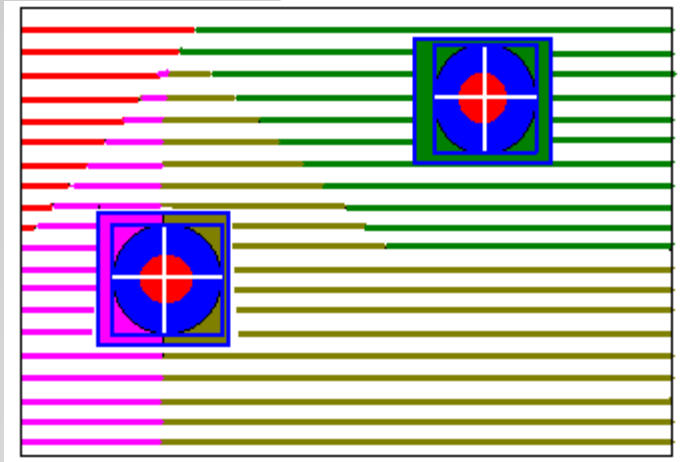
Refinement of Bounding Rectangles





- All pixels inside the bounding rectangle are used (“high res” pass) to refine the size and position of the rectangle by determining separating the background color from the “blue”-colored target

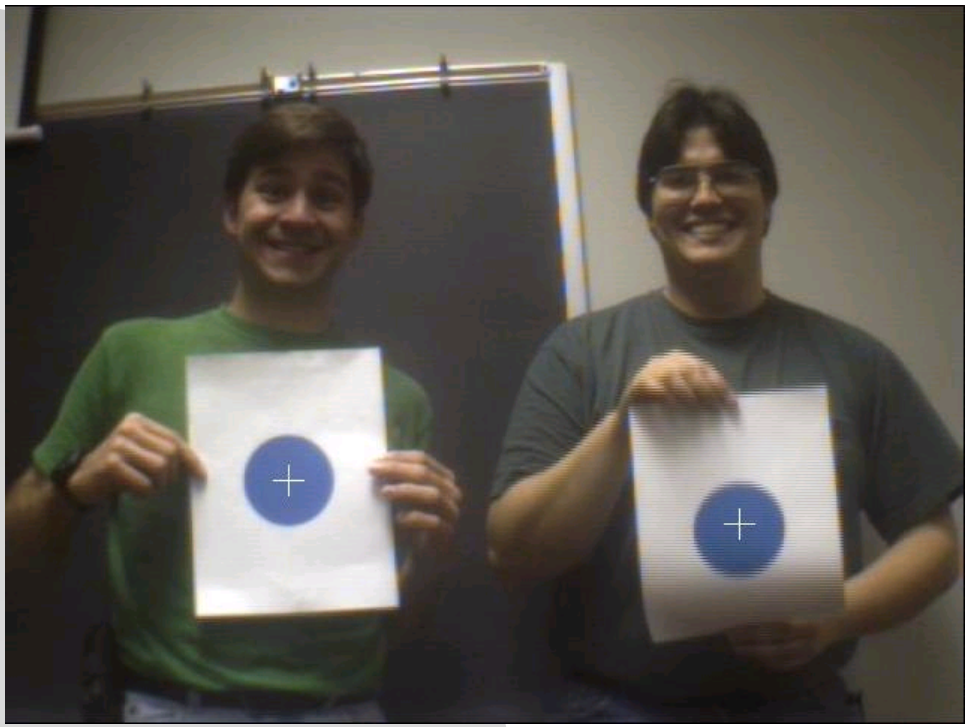
Identification of centroid

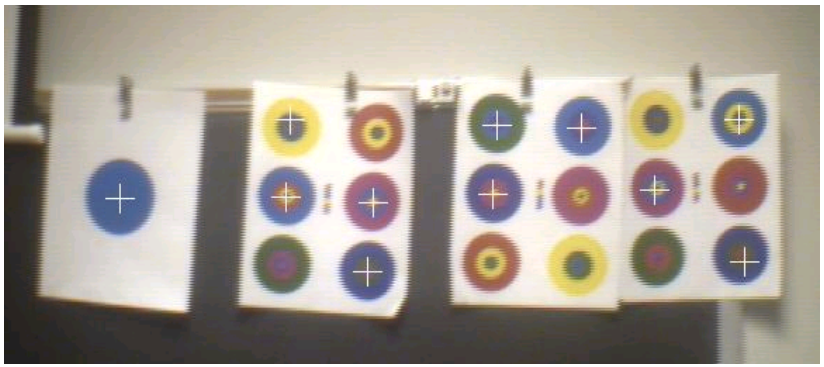


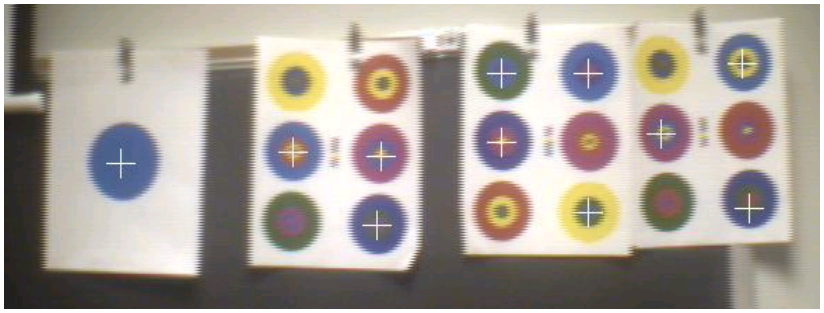
After bounding box refinement, we can “guess” we are looking at a blue-colored target if the new bounding rectangle is approximately “square”. (The program will not handle circles not directly perpendicular to the camera--they will appear an ellipses and therefore will not be detected.)

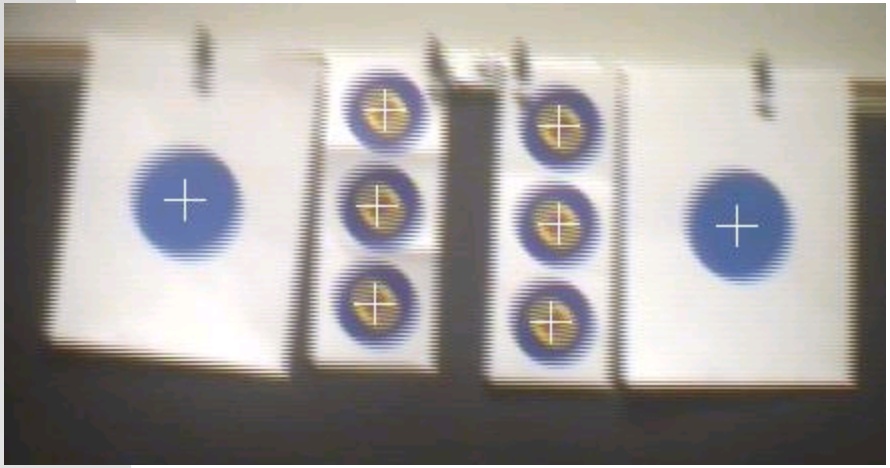
- A crosshair can be drawn over the original video image’s targets to produce an augmented reality image

Real examples









POSIT reference

- DeMenthon, Daniel F. and Davis, Larry S. “Model-Based Object Pose in 25 Lines of Code.” http://www.cfar.umd.edu/~daniel/daniel_papersfordownload/Pose25Lines.pdf

- POSIT = POS + IT
- POS = Pose from Orthography and Scaling
- IT = with Iterations

POSIT description

- Method for finding the pose of an object from single image
- Assumption: We can detect and match in the image four (or more) noncoplanar feature points of the object, and that we know the relative 3D geometry of the object
- POS: Approximates the perspective projection with a scaled orthographic projection and finds the rotation matrix and the translation vector of the object by solving a linear system
- POSIT: Uses in its iteration loop the approximate pose found by POS in order to compute better scaled orthographic projections of the feature points, then applies POS to these projections instead of the original image projections.

POSIT advantages

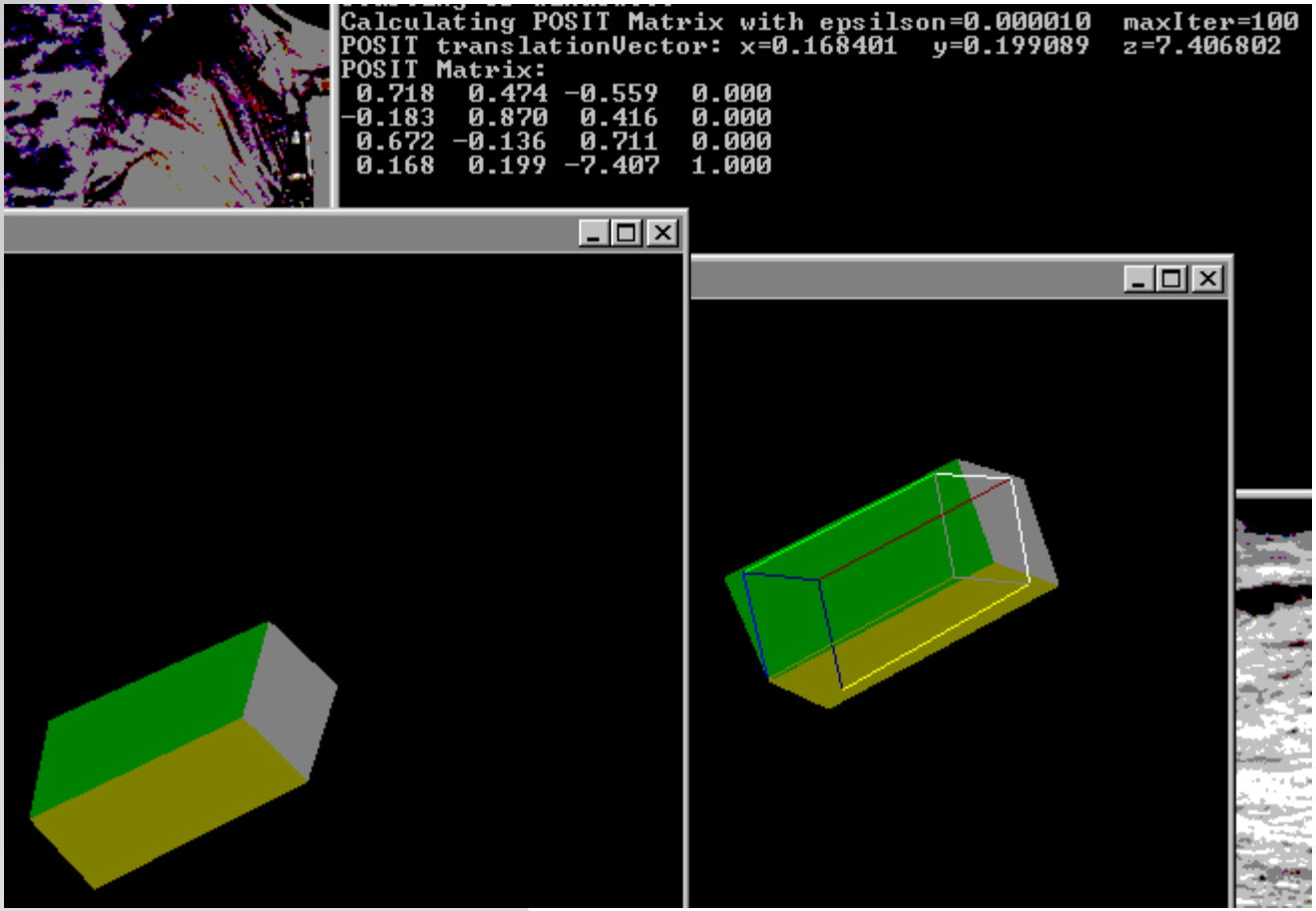
- More than 4 points can be used for added insensitivity to measurement errors and image noise
- Classic approach uses Newton's method and requires an initial pose estimate; POSIT does not require this and uses an order of magnitude fewer floating point operations (10x faster)
- Suitable for real-time operation
- Algorithm can be written in 25 lines of code (Mathematica)

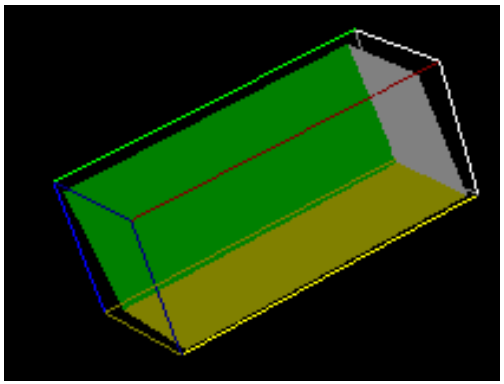
Review: Orthographic vs. Perspective Projections

- In orthogonal views, the projectors are perpendicular to the projection plane. Orthographic views preserves both distances and angles, and because there is no distortion of either distance or shape, orthographic projections are well suited for working drawings.

- Perspective views are characterized by diminution of size. When objects are moved farther from the viewer, their images become smaller. The size change gives perspective views their natural appearance. Because the amount by which the line is foreshortened depends on how far the line is from the viewer, we cannot make measurements from a perspective view. The major use of perspective views is in applications like architecture and animation, where it is important to achieve real-looking images.

Examples using OpenCV





- User picks cooresponding points from a 3D model and a 2D image; program then overlays 3D wireframe model on top of 2D image

POSIT usage example

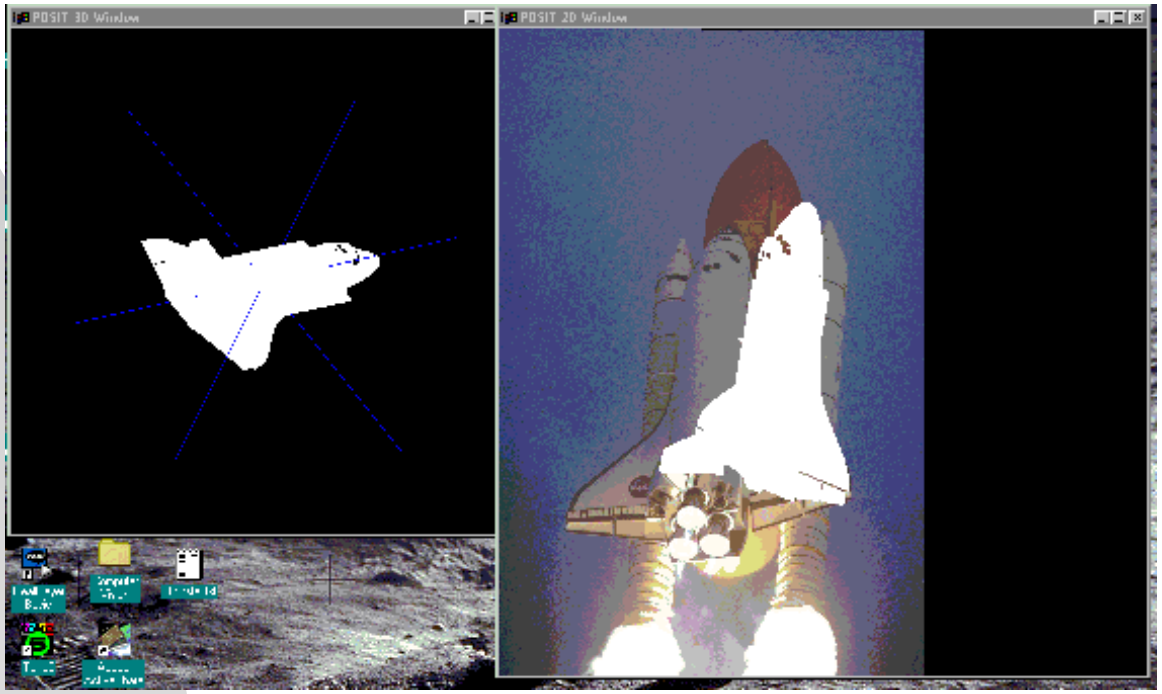
```
CvPoint2D32f* points2D;          //must be populated before calling POSIT
CvPoint3D32f* points3D;          //must be populated before calling POSIT
…                                //should be a 1:1 binding bwtn 2D & 3D points
CvPOSITObject* pObject;          //posit object
CvMatrix3 rotationMatrix;        //posit-returned rotation matrix
CvPoint3D32f translationVector;  //posit-returned translation vector

pObject = cvCreatePOSITObject(points3D, num3dPoints+1); //make sure not null

//set posit termination criteria: 100 max iterations, convergence epsilon 1.0e-5
CvTermCriteria criteria = cvTermCriteria(CV_TERMCRIT_EPS, 100, 1.0e-5);

cvPOSIT(points2D, pObject, FOCAL_LENGTH, criteria, &rotationMatrix,
    &translationVector);
cvReleasePOSITObject(&pObject)
```

- In order to make use of the returned matrix and vector with OpenGL, we must do some quick adjustments...

POSIT usage example, continued...

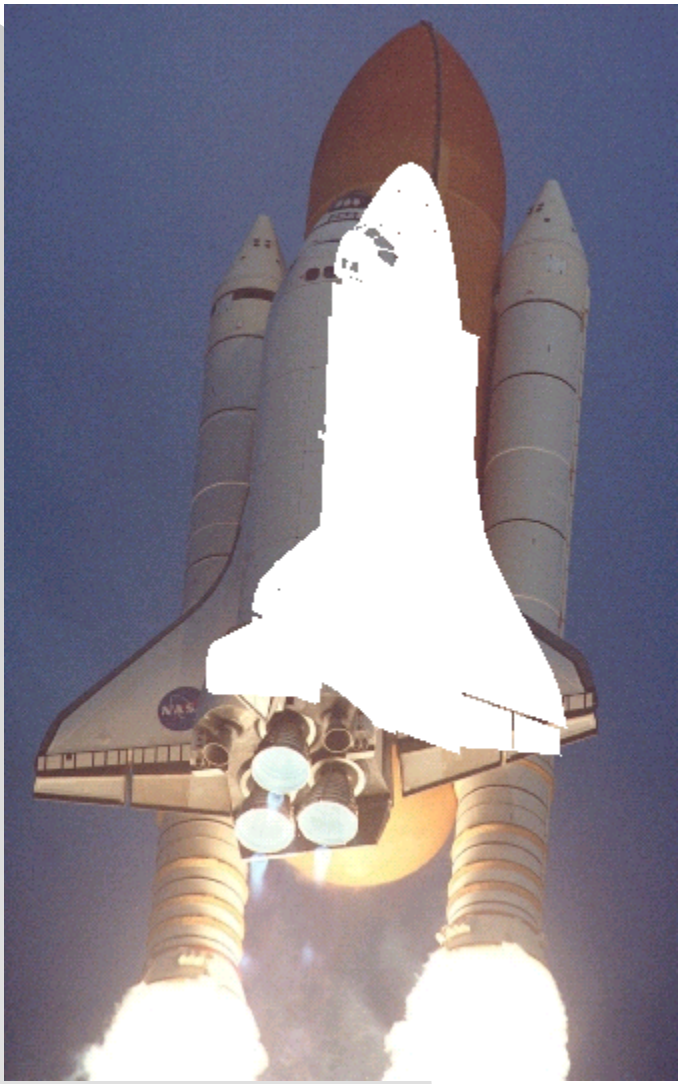
```
//populate opengl-compatible matrix from posit rotation matrix and translation vector
GLfloat positMatrix[16];

positMatrix[0] = rotationMatrix.m[0][0];
positMatrix[1] = rotationMatrix.m[1][0];
positMatrix[2] = rotationMatrix.m[2][0];
positMatrix[3] = 0.0; //homogeneous
positMatrix[4] = rotationMatrix.m[0][1];
positMatrix[5] = rotationMatrix.m[1][1];
positMatrix[6] = rotationMatrix.m[2][1];
positMatrix[7] = 0.0; //homogeneous
positMatrix[8] = rotationMatrix.m[0][2];
positMatrix[9] = rotationMatrix.m[1][2];
positMatrix[10] = rotationMatrix.m[2][2];
positMatrix[11] = 0.0; //homeogenous
positMatrix[12] = translationVector.x;
positMatrix[13] = translationVector.y;
positMatrix[14] = -translationVector.z; //negative
positMatrix[15] = 1.0; //homogeneous

for (int ii = 0; ii <= NUM_POINTS; ii++) { //coordinate system returned is relative to the first 3D input point
        pointArrayAdjusted[ii][0] = pointArray[ii][0] - points3D[0].x; //pointArray[] are 3D geometric coordinates of vertices in
        pointArrayAdjusted[ii][1] = pointArray[ii][1] - points3D[0].y; //the object's internal coordinate system (normalized bwtn 0.0 and 1.0)
        pointArrayAdjusted[ii][2] = pointArray[ii][2] - points3D[0].z;
}

printf("POSIT Matrix (OpenGL form):\n");
for (int j=0; j <= 15; j++) {
        printf("%6.3f ", positMatrix[j]);
        if ((j+1) % 4 == 0) printf("\n");
}
```

Example using VRML model of space shuttle



- Overlay did not line up; points chosen were left and right wing tips, tail, and nose.
- Overlay lines up with the tail and right wing tip, but not with the left wing tip and nose!

Second try...



- We still have the problem here; the same points were selected along with two additional (the two overhead windows), but in this case, the alignment with the tail and right wing is not as good as the previous attempt

Other OpenCV experiments

- Study of the calibration filter; attempting to modify is more of an exercise in MFC/Visual C++ than Computer Vision