# Computer Vision: Project Description

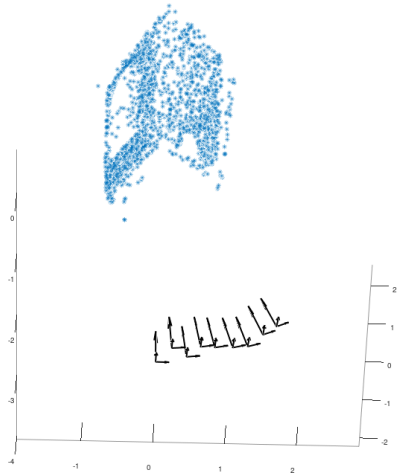
2023-11-30

# Overview

You should implement a 3D reconstruction software.

In short: from

to



# Algorithm: Nonsequential Approach

1. Calculate relative orientations $(\mathtt{R}_{i,i+1} \mid \mathbf{T}_{i,i+1})$ between images $i$ and $i+1$
2. Upgrade to absolute rotations $\mathtt{R}_i$
3. Reconstruct initial 3D points from an initial image pair $i_1$ & $i_2$
4. For each image $i$ *robustly* calculate the camera center $\mathbf{C}_i$ / translation $\mathbf{T}_i$
   Reduced camera resectioning problem (analogous to point triangulation)
5. Refine camera centers (or translation vectors) using Levenberg-Marquardt method
6. Triangulate points for all pairs $(i, i+1)$ and visualize 3D points + cameras

# Algorithm: Nonsequential Approach

1. Calculate relative orientations $(\mathtt{R}_{i,i+1} \mid \mathbf{T}_{i,i+1})$ between images $i$ and $i+1$
   - $\mathtt{P}_{i,1} = (\mathtt{I} \mid \mathbf{0})$ & $\mathtt{P}_{i,2} = (\mathtt{R}_{i,i+1} \mid \mathbf{T}_{i,i+1})$
   - Since some datasets have a dominant planar structure:
     Implement robust estimation of $(\mathtt{R} \mid \mathbf{T})$ that works for (nearly) planar and general non-planar scenes (following slides) and keep the inliers!
2. Upgrade to absolute rotations $\mathtt{R}_i$
3. Reconstruct initial 3D points from an initial image pair $i_1$ & $i_2$
4. For each image $i$ *robustly* calculate the camera center $\mathbf{C}_i$ / translation $\mathbf{T}_i$
   Reduced camera resectioning problem (analogous to point triangulation)
5. Refine camera centers (or translation vectors) using Levenberg-Marquardt method
6. Triangulate points for all pairs $(i, i+1)$ and visualize 3D points + cameras

# Algorithm: Nonsequential Approach

1. Calculate relative orientations $(\mathtt{R}_{i,i+1} \mid \mathbf{T}_{i,i+1})$ between images $i$ and $i+1$
2. Upgrade to absolute rotations $\mathtt{R}_i$
   - Set $\mathtt{R}_1 = \mathtt{I}$ and chain rotation matrices $\mathtt{R}_i = \mathtt{R}_{i-1,i}\mathtt{R}_{i-1}$
   - You now know the rotations matrices for all cameras!
3. Reconstruct initial 3D points from an initial image pair $i_1$ & $i_2$
4. For each image $i$ *robustly* calculate the camera center $\mathbf{C}_i$ / translation $\mathbf{T}_i$
   Reduced camera resectioning problem (analogous to point triangulation)
5. Refine camera centers (or translation vectors) using Levenberg-Marquardt method
6. Triangulate points for all pairs $(i, i+1)$ and visualize 3D points + cameras

# Algorithm: Nonsequential Approach

1. Calculate relative orientations $(\mathtt{R}_{i,i+1} \mid \mathbf{T}_{i,i+1})$ between images $i$ and $i+1$
2. Upgrade to absolute rotations $\mathtt{R}_i$
3. Reconstruct initial 3D points from an initial image pair $i_1$ & $i_2$
   - For better accuracy: $i_1$ and $i_2$ should not be adjacent (have enough baseline) Suggested values provided by us
   - Calculate $(\mathtt{R}_{i_2,i_1} \mid \mathbf{T}_{i_2,i_1})$ and triangulate inliers: 3D points $\mathcal{X}_0$
   - Center $\mathcal{X}_0$ and bring to world coordinates (via $\mathtt{R}_{i_1}^\top$)
   - Save the SIFT descriptors (coming from $i_1$ or $i_2$) for reconstructed 3D points e.g. in `desc_X`
4. For each image $i$ *robustly* calculate the camera center $\mathbf{C}_i$ / translation $\mathbf{T}_i$ Reduced camera resectioning problem (analogous to point triangulation)
5. Refine camera centers (or translation vectors) using Levenberg-Marquardt method
6. Triangulate points for all pairs $(i, i+1)$ and visualize 3D points + cameras

# Algorithm: Nonsequential Approach

1. Calculate relative orientations $(\mathtt{R}_{i,i+1} \mid \mathbf{T}_{i,i+1})$ between images $i$ and $i+1$
2. Upgrade to absolute rotations $\mathtt{R}_i$
3. Reconstruct initial 3D points from an initial image pair $i_1$ & $i_2$
4. For each image $i$ *robustly* calculate the camera center $\mathbf{C}_i$ / translation $\mathbf{T}_i$
   Reduced camera resectioning problem (analogous to point triangulation)
   1. Establish correspondences between $i$ and 3d points (using `desc_X`), e.g.
      `[matches_2d_3d, scores] = vl_ubcmatch(descriptors{i}, desc_X);`
   2. Estimate $\mathbf{C}_i/\mathbf{T}_i$ robustly from these correspondences using a 2-point method
5. Refine camera centers (or translation vectors) using Levenberg-Marquardt method
   - Modify CE in assignment 4
6. Triangulate points for all pairs $(i, i+1)$ and visualize 3D points + cameras

# Things you need to implement

In order to complete the mandatory parts

1. A RANSAC algorithm robustly estimates $\mathtt{R}$ and $\mathbf{T}$ from correspondences
   - For scenes with and w/o a dominant plane
   - Suggested approach: use a "parallel" version of RANSAC
   - Alternative: implement a direct 5-point solver (optional points)
     - In this case a standard RANSAC loop is sufficient
2. Extract $\mathtt{R}$ and $\mathbf{T}$ from either $\mathtt{E}$ or $\mathtt{H}$
   - $(\mathtt{R}|\mathbf{T})$ from $\mathtt{E}$: CE3 in assignment 3
   - $(\mathtt{R}|\mathbf{T})$ from $\mathtt{H}$: provided (`homography_to_RT`)
   - Use cheirality to determine the correct configuration
3. A method to estimate the translation $\mathbf{T}$ robustly from 2D-3D correspondences $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$
   - E.g. `P = estimate_T_robust(xs, Xs, inlier_threshold)`
   - Simplify `estimate_camera_DLT` from CE2 in assignment 2 to estimate $\mathbf{T}$ for a minimal sample
4. High-level wrapper code ("glue code")

# RANSAC: Searching for $\mathtt{E}$ and $\mathtt{H}$ in parallel

Modify the vanilla RANSAC loop

- Maintain (and return) $(\mathtt{R}^*|\mathbf{T}^*)$
- Maintain inlier fractions $\varepsilon_\mathtt{E}$ and $\varepsilon_\mathtt{H}$ and corresponding number of iterations
- Return best $(\mathtt{R}^*|\mathbf{T}^*)$ found after sufficient iterations for a good $\mathtt{E}$ or $\mathtt{H}$

In each iteration

- Estimate $\mathtt{E}$ using the 8/7-point method from a minimal sample set of size 8/7
  - If $\mathtt{E}$ is the new best essential matrix:
    Adjust $\varepsilon_\mathtt{E}$, number of $\mathtt{E}$-iterations, and extract new $(\mathtt{R}^*|\mathbf{T}^*)$ from $\mathtt{E}$
- Estimate $\mathtt{H}$ using the DLT from a minimal sample set of size 4
  - If $\mathtt{H}$ is new best homography:
    Extract two solutions $(\mathtt{R}_a|\mathbf{T}_a)$ and $(\mathtt{R}_b|\mathbf{T}_b)$ from $\mathtt{H}$ (`homography_to_RT`)
    Test both essential matrices $\mathtt{E} \in \{[\mathbf{T}_a]_\times \mathtt{R}_a, [\mathbf{T}_b]_\times \mathtt{R}_b\}$ for validity
      If $\mathtt{E}$ is the new best essential matrix *and* is "acceptable"
        Assign correct $(\mathtt{R}|\mathbf{T})$ (extracted from $\mathtt{E}$) to $(\mathtt{R}^*|\mathbf{T}^*)$
        Adjust $\varepsilon_\mathtt{H}$ & number of $\mathtt{H}$-iterations
        Adjust $\varepsilon_\mathtt{E}$ & number of $\mathtt{E}$-iterations

An essential matrix $\mathtt{E}$ is "acceptable" if exactly one of the 4 cheirality configurations has all inliers (w.r.t. the epipolar constraint) in front of both cameras

# RANSAC: Searching for $\mathtt{E}$ and $\mathtt{H}$ in parallel

Modify the vanilla RANSAC loop

- Maintain (and return) $(\mathtt{R}^*|\mathbf{T}^*)$
- Maintain inlier fractions $\varepsilon_\mathtt{E}$ and $\varepsilon_\mathtt{H}$ and corresponding number of iterations
- Return best $(\mathtt{R}^*|\mathbf{T}^*)$ found after sufficient iterations for a good $\mathtt{E}$ or $\mathtt{H}$

In each iteration

- Estimate $\mathtt{E}$ using the 8/7-point method from a minimal sample set of size 8/7
  - If $\mathtt{E}$ is the new best essential matrix:
    Adjust $\varepsilon_\mathtt{E}$, number of $\mathtt{E}$-iterations, and extract new $(\mathtt{R}^*|\mathbf{T}^*)$ from $\mathtt{E}$
- Estimate $\mathtt{H}$ using the DLT from a minimal sample set of size 4
  - If $\mathtt{H}$ is new best homography:
    Extract two solutions $(\mathtt{R}_a|\mathbf{T}_a)$ and $(\mathtt{R}_b|\mathbf{T}_b)$ from $\mathtt{H}$ (`homography_to_RT`)
    Test both essential matrices $\mathtt{E} \in \{[\mathbf{T}_a]_\times \mathtt{R}_a, [\mathbf{T}_b]_\times \mathtt{R}_b\}$ for validity
    If $\mathtt{E}$ is the new best essential matrix *and* is "acceptable"
    Assign correct $(\mathtt{R}|\mathbf{T})$ (extracted from $\mathtt{E}$) to $(\mathtt{R}^*|\mathbf{T}^*)$
    Adjust $\varepsilon_\mathtt{H}$ & number of $\mathtt{H}$-iterations
    Adjust $\varepsilon_\mathtt{E}$ & number of $\mathtt{E}$-iterations

An essential matrix $\mathtt{E}$ is "acceptable" if exactly one of the 4 cheirality configurations has all inliers (w.r.t. the epipolar constraint) in front of both cameras

# How to run your software

- Once your implementation is complete, you should run it (in Matlab) via

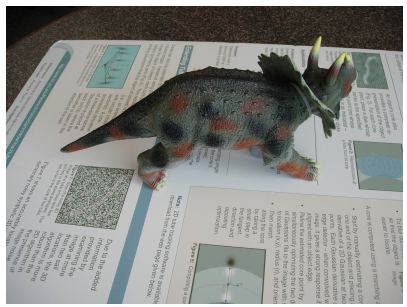
  ```
  > run_sfm(<dataset number>)
  ```

- Datasets 1–9 are test datasets provided by us
  - Datasets 1 & 2: for debugging, do not include in the report
  - Datasets 3–7: increasing difficulty, you should include those in the report
  - Datasets 8 & 9: optional
- You may include your own datasets (optional, see later)







# For the report...

- Provide a high-level explanation of your algorithmic design choices
  - Especially what you implemented beyond the routines from the computer exercises
  - Also if you modified/improved your implementation for the computer exercises
- Describe unexpected problems you encountered (if any)
- Document changes to the provided code
  - E.g. modifying inlier threshold or selecting a different initial pair
- Briefly describe extra datasets (if you have any)
  - If you picked something from the internet: link to the dataset(s)
  - Describe unexpected issues with the additional datasets
- For optional points: describe what you implemented

*Deadline: Thursday, January 4nd, 2024, 23:59*

# Extracting R and T from H

- Bill Triggs, "Autocalibration from Planar Scenes", ECCV 1998 (Appendix 1)
  - http://perception.inrialpes.fr/Publications/1998/Tri98/Triggs-eccv98-long.pdf
  - Adapted to our conventions
    `function [R1, t1, R2, t2] = homography_to_RT(H, x1, x2)`
  - Python: convert Matlab routine, or
    `num, Rs, Ts, Ns = cv2.decomposeHomographyMat(H, K)` (with $K = I$)

```
function [R1,t1,n1, R2,t2,n2, zeta] = homog_to_Rt(H)
  [U,S,V] = svd(H);
  s1 = S(1,1)/S(2,2); s3 = S(3,3)/S(2,2);
  zeta = s1-s3;
  a1 = sqrt(1-s3^2); b1 = sqrt(s1^2-1);
  [a,b] = unitize(a1,b1);
  [c,d] = unitize( 1+s1*s3, a1*b1 );
  [e,f] = unitize( -b/s1, -a/s3 );
  v1 = V(:,1); v3 = V(:,3);
  n1 = b*v1-a*v3; n2 = b*v1+a*v3;
  R1 = U*[c,0,d; 0,1,0; -d,0,c]*V';
  R2 = U*[c,0,-d; 0,1,0; d,0,c]*V';
  t1 = e*v1+f*v3; t2 = e*v1-f*v3;
  if (n1(3)<0) t1 = -t1; n1 = -n1; end;
  if (n2(3)<0) t2 = -t2; n2 = -n2; end;
end;
```

# Importing dataset information

- You may create a function `get_dataset_info(<dataset number>)`, e.g.

```
function [K, image_names, init_pair] = get_dataset_info(dataset)
   if dataset == 1
       image_names = {"dataset1/kronan1.JPG", "dataset1/kronan2.JPG"};
       im_width = 1936; im_height = 1296;
       focal_length_35mm = 45.0; % from the EXIF data
       init_pair = [1, 2];
       pixel_threshold = 1.0;
   elseif
      ...
   end
   focal_length = max(im_width, im_height) * focal_length_35mm / 35.0
   K = [focal_length 0 im_width/2; 0 focal_length im_height/2; 0 0 1]
end
```

- This is probably one of the first routines to call in your `run_sfm` script
  - Another one is seting up `vl_feat`
- *Provided together with the project datasets in the* `data` *folder*

# Practical Suggestions

- Work with normalized image points $\tilde{\mathbf{x}}_i = \mathtt{K}^{-1}\mathbf{x}_i$
  - Camera matrices are $\mathtt{P}_i = (\mathtt{R}_i \mid \mathbf{T}_i)$
  - Do not forget to adjust inlier thresholds from pixels to normalized units (`pixel_threshold` is a suggested value in `get_dataset_info`)

```
epipolar_threshold    =     pixel_threshold / focal_length
homography_threshold  = 3 * pixel_threshold / focal_length
translation_threshold = 3 * pixel_threshold / focal_length
```

  - Use maybe `xn` for normalized and `xu` for unnormalized image points
- Filter 3D points excessively far away from the center of gravity, e.g.

$$\|\mathbf{X}_j - \bar{\mathbf{X}}\| \leq 5 \times 90\% \text{ quantile of } \{\|\mathbf{X}_j - \bar{\mathbf{X}}\|\}_j$$

- Check (visualize) triangulated points and cameras for each relative pose
  - $\mathtt{P}_{i,1} = (\mathtt{I} \mid \mathbf{0})$ & $\mathtt{P}_{i,2} = (\mathtt{R}_{i+1,i} \mid \mathbf{T}_{i+1,i})$
  - Verify correctness of relative pose and cheirality
- While in the development/debugging phase
  - Precompute and store keypoints and matches
  - Load those for faster time to reach your own code

# Practical Suggestions

- Work first with the non-planar datasets 1-5 and start the RANSAC implementation from CE2 in assignment 4
- RANSAC is a stochastic method, so it may *occasionally* fail
  - But there is likely a bug if your software frequently produces bad 3D models
- Use `randperm(N, K)` to draw $K$ indices from $\{1, \ldots, N\}$ (w/o replacement)
- Reproducability while debugging: early in your code set the random seed via
  - Matlab: `rng(42);`
  - Octave: `rand("state", 42);`
- Accelerating model evaluation in RANSAC
  - Use `.^2`, `sum(X, dim)` and `sqrt.` instead of loops
    E.g. `sqrt.(sum((x-y).^2, 1))`

# Optional Points for Additional Functionality

1. Run your software on datasets 8 & 9 (5 points each, max. 10 points)
2. Run your software on more datasets (10 points/dataset, max. 20 points)
   - Describe dataset (and its origin) and your experiences briefly
   - Recall lecture 3 for how to extract `K`
   - Images should have $\approx$ HD ($1920 \times 1080$) resolution
     - Tradeoff between run-time and image details
   - Use dataset ids $\geq 10$
3. Use MSAC model scoring (max. 10 points)
   - Reference: `https://www.robots.ox.ac.uk/~vgg/publications/2000/Torr00/torr00.pdf`
4. Faster RANSAC using early bailout: $T_{d,d}$ test (max. 10 points)
   - Reference: Chum & Matas, "Randomized RANSAC with $T_{d,d}$ test"
5. Nonlinear refinement of camera pose (max. 20 points)
   - Optimize over camera translation/center and rotation matrix
6. Implement the 7-point method (cf. lecture 8, max. 20 points)
   - Document how you used a CAS
7. Implement the 5-point method (max. 60 points)
   - Cf. lecture 8 and the following hints
   - No need for the parallel search for `E` and `H`
   - Document how you used a CAS (if you did)

# The 5-Point Method

Finding an essential matrix can be done with five correspondences by solving

$$\begin{aligned} \bar{\mathbf{x}}_i^T \mathtt{E} \mathbf{x}_i &= 0 & i &= 1, \ldots, 5 \\ \det(\mathtt{E}) &= 0 \\ 2\mathtt{E}\mathtt{E}^\top \mathtt{E} - \operatorname{trace}(\mathtt{E}\mathtt{E}^\top)\mathtt{E} &= 0 \end{aligned}$$

Form the system matrix $\mathtt{M}$ from the five correspondences. $\mathtt{M}$ has a 4 dimensional null-space:

$$\mathtt{M}(x\mathbf{n}_1 + y\mathbf{n}_2 + z\mathbf{n}_3 + w\mathbf{n}_4) = 0$$

Reshape into matrices $\mathtt{E}_i$ such that $\mathbf{n}_i = \operatorname{vec}(\mathtt{E}_i)$

$$\bar{\mathbf{x}}_i^T (x\mathtt{E}_1 + y\mathtt{E}_2 + z\mathtt{E}_3 + w\mathtt{E}_4)\mathbf{x}_i = 0 \quad i = 1, \ldots, 5$$

# The 5-Point Method

- Use the remaining equations to determine $x, y, z, w$
  - Fix $w = 1$: $\mathbf{E} = x\mathbf{E}_1 + y\mathbf{E}_2 + z\mathbf{E}_3 + \mathbf{E}_4$
- Identify $x \to \alpha_1$, $y \to \alpha_2$, $z \to \alpha_3$, $1 \to \alpha_4$

  Trace constraint, 9 equations

$$2\mathbf{E}\mathbf{E}^\top\mathbf{E} - \operatorname{trace}(\mathbf{E}\mathbf{E}^\top)\mathbf{E} = \sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4}\alpha_i\alpha_j\alpha_k\left(2\mathbf{E}_i\mathbf{E}_j^\top\mathbf{E}_k - \operatorname{trace}(\mathbf{E}_i\mathbf{E}_j^\top)\mathbf{E}_k\right) \stackrel{!}{=} 0$$

  Determinant, one equation:

$$\det(\mathbf{E}) = \sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4}\alpha_i\alpha_j\alpha_k\big(e_{11}^i e_{22}^j e_{33}^k + e_{12}^i e_{23}^j e_{31}^k + e_{13}^i e_{21}^j e_{32}^k - e_{11}^i e_{23}^j e_{32}^k - e_{12}^i e_{21}^j e_{33}^k - e_{13}^i e_{22}^j e_{31}^k\big) \stackrel{!}{=} 0$$

- Map 64 expressions $\alpha_i\alpha_j\alpha_k$ to the 20 degree $\leq 3$ monomials in $x$, $y$, $z$
  - E.g. $\alpha_1\alpha_2\alpha_1 \to x^2y$ (index 2) and $\alpha_3\alpha_4\alpha_4 \to z$ (index 19)
    - You can fill a 3-tensor `T(i,j,k)` to map $(i, j, k)$ to its monomial index
- Or use a CAS to extract the coefficients for the 10 multi-variate polynomials
  - Analogous to the use of a CAS for the 7-point method (lecture 8)
- You obtain $10 \times 20$ coefficients!

# The 5-Point Method

- Use the remaining equations to determine $x, y, z, w$
  - Fix $w = 1$: $\mathbf{E} = x\mathbf{E}_1 + y\mathbf{E}_2 + z\mathbf{E}_3 + \mathbf{E}_4$
- Identify $x \to \alpha_1$, $y \to \alpha_2$, $z \to \alpha_3$, $1 \to \alpha_4$

  Trace constraint, 9 equations

$$2\mathbf{E}\mathbf{E}^\top\mathbf{E} - \operatorname{trace}(\mathbf{E}\mathbf{E}^\top)\mathbf{E} = \sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4}\alpha_i\alpha_j\alpha_k\left(2\mathbf{E}_i\mathbf{E}_j^\top\mathbf{E}_k - \operatorname{trace}(\mathbf{E}_i\mathbf{E}_j^\top)\mathbf{E}_k\right) \stackrel{!}{=} 0$$

  Determinant, one equation:

$$\det(\mathbf{E}) = \sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4}\alpha_i\alpha_j\alpha_k\big(e_{11}^i e_{22}^j e_{33}^k + e_{12}^i e_{23}^j e_{31}^k + e_{13}^i e_{21}^j e_{32}^k - e_{11}^i e_{23}^j e_{32}^k - e_{12}^i e_{21}^j e_{33}^k - e_{13}^i e_{22}^j e_{31}^k\big) \stackrel{!}{=} 0$$

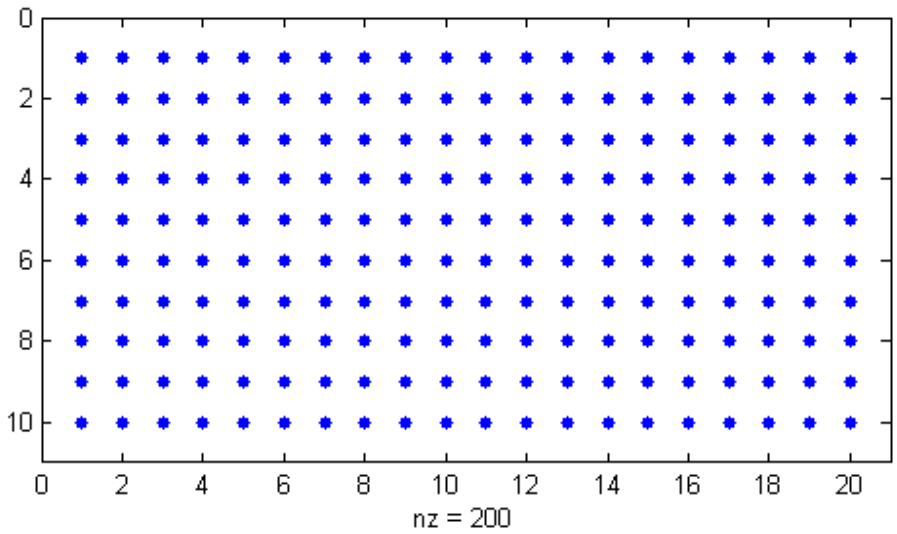
- Map 64 expressions $\alpha_i\alpha_j\alpha_k$ to the 20 degree $\leq 3$ monomials in $x$, $y$, $z$
  - E.g. $\alpha_1\alpha_2\alpha_1 \to x^2y$ (index 2) and $\alpha_3\alpha_4\alpha_4 \to z$ (index 19)
    - You can fill a 3-tensor `T(i,j,k)` to map $(i, j, k)$ to its monomial index
- Or use a CAS to extract the coefficients for the 10 multi-variate polynomials
  - Analogous to the use of a CAS for the 7-point method (lecture 8)
- You obtain $10 \times 20$ coefficients!

# The 5-Point Method

- Order for monomials: all 10 degree-3 monomials come first, e.g.
  - Correspond to columns in your $10 \times 20$ coefficient matrix $\mathtt{C}$

$$(x^3, x^2y, x^2z, xy^2, xyz, xz^2, y^3, y^2z, yz^2, z^3, \quad x^2, xy, xz, y^2, yz, z^2, x, y, z, 1)$$

- Gaussian elimination gives reductions for all third order terms
  - Lucky coincidence: 10 degree-3 monomials and 10 equations
  - Gaussian elimination: $\mathtt{C}' = (\mathtt{C}_{1:10,1:10})^{-1}\mathtt{C}$

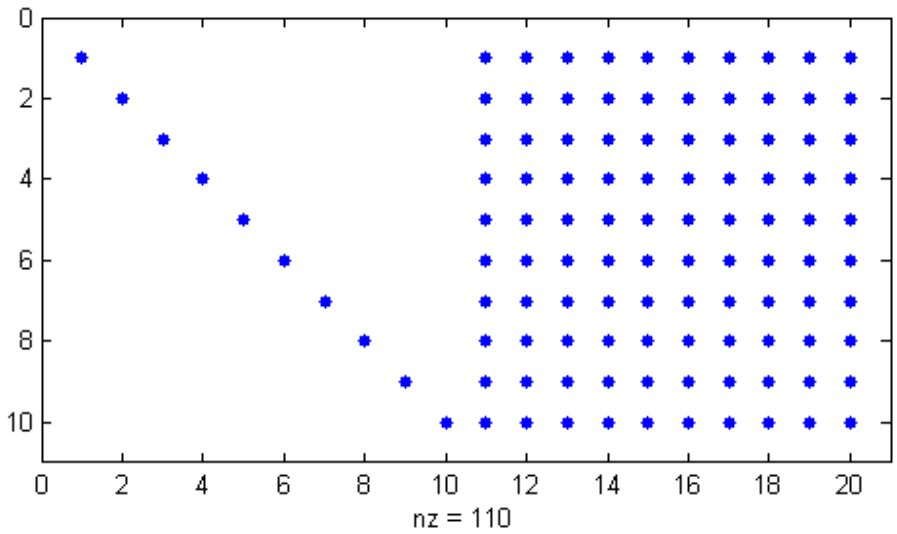


# The 5-Point Method

- Order for monomials: all 10 degree-3 monomials come first, e.g.
  - Correspond to columns in your $10 \times 20$ coefficient matrix $\mathtt{C}$

$$(x^3, x^2y, x^2z, xy^2, xyz, xz^2, y^3, y^2z, yz^2, z^3, \quad x^2, xy, xz, y^2, yz, z^2, x, y, z, 1)$$

- Gaussian elimination gives reductions for all third order terms
  - Lucky coincidence: 10 degree-3 monomials and 10 equations
  - Gaussian elimination: $\mathtt{C}' = (\mathtt{C}_{1:10,1:10})^{-1}\mathtt{C}$



# The 5-Point Method

- We can express all degree-3 monomials via degree-$\leq 2$ ones
- Action matrix $\mathtt{M}_x$ (or $\mathtt{M}_y$ etc) is $10 \times 10$ matrix
  - Only 6 equations / rows of $\mathtt{C}'$ will be used

$$x \begin{pmatrix} x^2 \\ xy \\ xz \\ y^2 \\ yz \\ z^2 \\ x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x^3 \\ x^2y \\ x^2z \\ xy^2 \\ xyz \\ xz^2 \\ x^2 \\ xy \\ xz \\ x \end{pmatrix} = \mathtt{M}_x \begin{pmatrix} x^2 \\ xy \\ xz \\ y^2 \\ yz \\ z^2 \\ x \\ y \\ z \\ 1 \end{pmatrix}$$

- Up to 10 real eigenvalues for $x$
- How do we get $y$ and $z$?
- Each eigenvector $\mathbf{v} \sim (x^2, xy, xz, y^2, yz, z^2, x, y, z, 1)^\top$
- Extract $(x, y, z)$ as $\mathbf{v}_{7:9}/\mathrm{v}_{10}$
  - Discard complex solutions (what about "almost" real ones?)
  - Or try the real parts of all $(x, y, z)$
  - $\mathbf{E} = x\mathbf{E}_1 + y\mathbf{E}_2 + z\mathbf{E}_3 + \mathbf{E}_4$
- Evaluate all 10 essential matrices for their inliers

# The 5-Point Method

- We can express all degree-3 monomials via degree-$\leq 2$ ones
- Action matrix $\mathtt{M}_x$ (or $\mathtt{M}_y$ etc) is $10 \times 10$ matrix
  - Only 6 equations / rows of $\mathtt{C}'$ will be used

$$x \begin{pmatrix} x^2 \\ xy \\ xz \\ y^2 \\ yz \\ z^2 \\ x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x^3 \\ x^2y \\ x^2z \\ xy^2 \\ xyz \\ xz^2 \\ x^2 \\ xy \\ xz \\ x \end{pmatrix} = \mathtt{M}_x \begin{pmatrix} x^2 \\ xy \\ xz \\ y^2 \\ yz \\ z^2 \\ x \\ y \\ z \\ 1 \end{pmatrix}$$

- Up to 10 real eigenvalues for $x$
- How do we get $y$ and $z$?
- Each eigenvector $\mathbf{v} \sim (x^2, xy, xz, y^2, yz, z^2, x, y, z, 1)^\top$
- Extract $(x, y, z)$ as $\mathbf{v}_{7:9}/\mathrm{v}_{10}$
  - Discard complex solutions (what about "almost" real ones?)
  - Or try the real parts of all $(x, y, z)$
  - $\mathbf{E} = x\mathbf{E}_1 + y\mathbf{E}_2 + z\mathbf{E}_3 + \mathbf{E}_4$
- Evaluate all 10 essential matrices for their inliers