Model Fitting and Optimisation in Ravl

Phil McLauchlan

CVSSP, University of Surrey

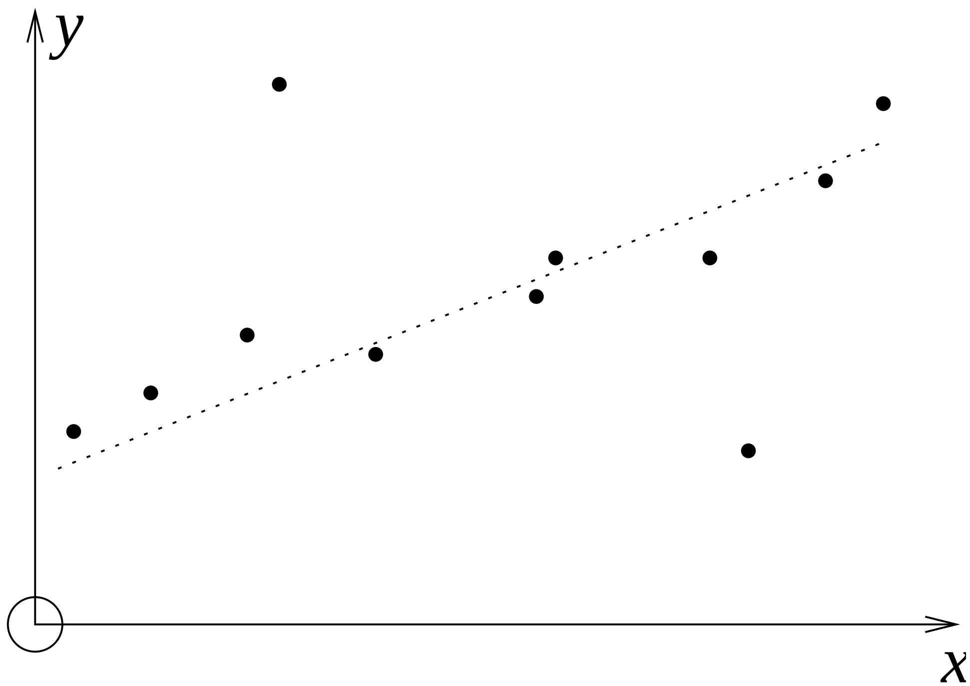
With thanks to: Charles Galambos, Bill Christmas, Josef Kittler.

Outline

1. Robust model fitting
2. RANSAC
   (a) Theory
   (b) Examples
   (c) Implementation in Ravl
3. Robust least squares
   (a) Theory
   (b) Examples
   (c) Implementation in Ravl
4. Example classes
5. Shrink-wrapped routines
6. Demonstration of mosaic building
7. Conclusions

Robust model fitting

Example: fitting a straight line $y = ax + b$ through points with outliers.



Least squares minimisation of $\sum_i (y_i - ax_i - b)^2$ will *always* fail in the presence of outliers.

Least squares: Optimal but fragile

- Least squares provides the best linear unbiased estimate of the model parameters (Gauss-Markov theorem).

- I.e. least-squares is *optimal* given that the data has unbiased error with known covariance.

- This strength of least squares is also its weakness, because of the strong assumptions.

- Two main strands in our approach:

  1. Identify and reject the outlier points (RANSAC).

  2. Construct a robust error model to incorporate both inlier and outlier points (robustified least squares).

RANSAC

- Fischler & Bolles introduced RANSAC in (CACM 1981) as a general solution to the problem of robust model fitting.

- It was widely ignored. One could speculate that:

  1. Americans invented it, but it wasn't their style.
  2. Europeans would like to have invented it.
  3. It's too simple.
  4. It's not deterministic.
  5. We waited for faster processors to make 8-dimensional Hough transforms feasible.

- The late 1990's saw a resurgence of interest as reality dawned.

RANSAC is simple

Let's go back to our line fitting example. Follow these steps:

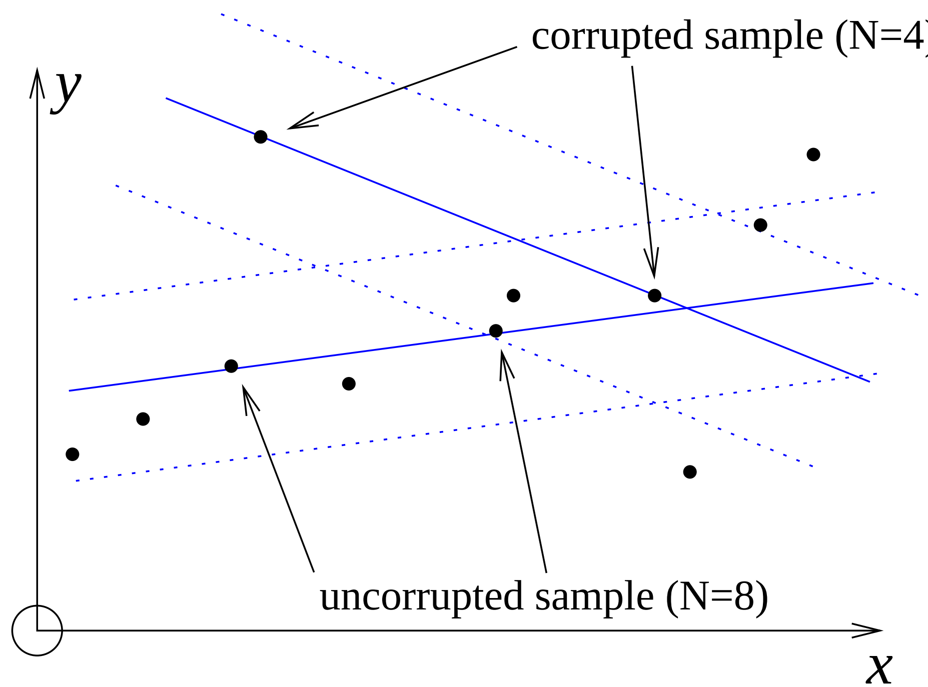
1. Initialise integer `Nmax` to zero and create line parameters `Abest, Bbest`.
2. Pick two points at random;
3. Compute the parameters `a,b` of the line $y = ax + b$ through the two points;
4. Using a distance threshold, count the number `N` of points "close enough" to the line `a,b` to be treated as inlier points.
5. Update the largest number of inlier points and the best-fit line parameters:

```
if ( N > Nmax )
{
   Abest = a;
   Bbest = b;
   Nmax = N;
}
```

6. Repeat from step 2.

RANSAC for line fitting example

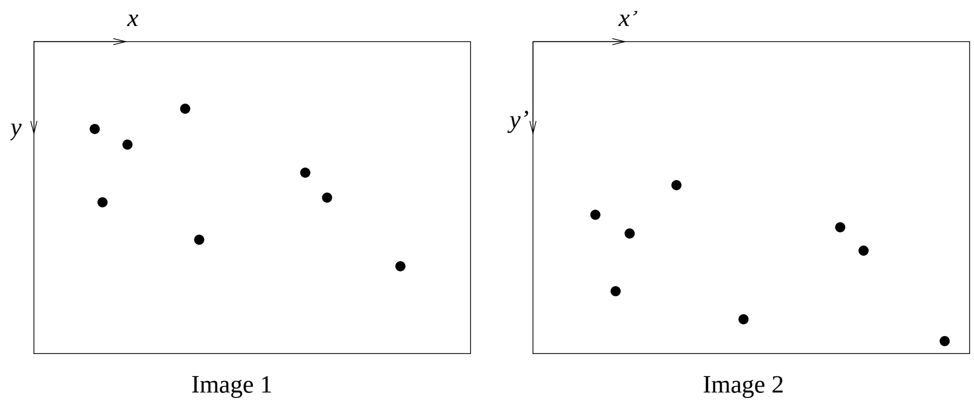
Two random samples:



Note that the uncorrupted sample still leaves inlier points labelled as outliers.

Another example: 2D projective motion estimation

We have two images of points



If the images are either

1. of the same plane, or

2. from a rotating camera,

then they are related by a 2D projective transformation:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \lambda P \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

for points x, y in the first image and x', y' in the second image. P is a 3×3 matrix.

Planar scene $\Longrightarrow$ projective transform between images

Full camera projection from 3D scene $\mathbf{X}$ into 2D image $\mathbf{p}$:

$$\mathbf{p} = \lambda K(R \,|\, \mathbf{T}) \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix}, \quad \text{or}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \lambda \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & z \end{pmatrix} \begin{pmatrix} R_{XX} & R_{XY} & R_{XZ} & T_X \\ R_{YX} & R_{YY} & R_{YZ} & T_Y \\ R_{ZX} & R_{ZY} & R_{ZZ} & T_Z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

If the scene is planar, we can w.l.o.g. set $Z = 0$, and the projection reduces to

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \lambda \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & z \end{pmatrix} \begin{pmatrix} R_{XX} & R_{XY} & T_X \\ R_{YX} & R_{YY} & T_Y \\ R_{ZX} & R_{ZY} & T_Z \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

Planar scene $\Longrightarrow$ projective transform (continued)

We have

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \lambda \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & z \end{pmatrix} \begin{pmatrix} R_{XX} & R_{XY} & T_X \\ R_{YX} & R_{YY} & T_Y \\ R_{ZX} & R_{ZY} & T_Z \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$
$$= \lambda M \mathbf{P}$$

for a 3×3 matrix M. Then given two images $\mathbf{p}$ and $\mathbf{p}'$ of the same plane $\mathbf{X}$, we have

$$\mathbf{p} = \lambda M \mathbf{P}, \quad \mathbf{p}' = \lambda' M' \mathbf{P}$$

and so finally

$$\begin{aligned} \mathbf{p}' &= \mu M' M^{-1} \mathbf{p} \\ &= \mu P \mathbf{p} \end{aligned} \tag{1}$$

We have our projective transform P.

Rotating camera $\Longrightarrow$ projective transform

If the camera is rotating then $\mathbf{T} = \mathbf{0}$ and the projections are

$$\begin{aligned} \mathbf{p} &= \lambda K R \mathbf{X} \\ \mathbf{p}' &= \lambda' K' R' \mathbf{X} \end{aligned}$$

from which we construct

$$\begin{aligned} M &= KR \\ M' &= K'R' \\ P &= M' M^{-1} \end{aligned}$$

and again we have our projective transform P relating x, y and x', y'.

2D projective motion estimation

The problem is to fit a single P projective transformation matrix to a set of point matches (x, y), (x', y').

First step: from the equation

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \lambda P \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

remove the homogeneous coordinate scale factor λ:

$$\begin{aligned} x'(P_{zx}x + P_{zy}y + P_{zz}z) &= z'(P_{xx}x + P_{xy}y + P_{xz}z) \\ y'(P_{zx}x + P_{zy}y + P_{zz}z) &= z'(P_{yx}x + P_{yy}y + P_{yz}z) \end{aligned}$$

Now we can solve these equations for P given four or more point matches.

Some of the corner matches are incorrect, so there are outliers.

RANSAC is applied with a sample size of four point matches.

Optimisation in Ravl

Classes to remember:

`StateVectorC` encapsulates the model parameters being computed, e.g. the line parameters `a, b`.

`ObsVectorC` encapsulates a data point/item, e.g. a point x_i, y_i.

`ObservationC` encapsulates an `ObsVectorC` plus its relationship to a `StateVectorC`, e.g. the equation $y = ax + b$.

RANSAC in Ravl

Three elements to the Ravl RANSAC implementation class `RansacC`:–

1. `ObservationManagerC` Provides random samples from the data points, as lists of `ObservationC`'s

2. `FitToSampleC` Fits the model parameters to a sample, producing a `StateVectorC` result.

3. `EvaluateSolutionC` Evaluates the model parameters computed from sample, producing a "vote" `N` to be compared with the current best vote `Nmax`.

The `RansacC` class provides the basic RANSAC functionality.

Extra trick: Select inliers from RANSAC solution using a larger threshold, to feed into robust least squares.

RANSAC in Ravl, continued

Subclasses allow more specialised approaches:

1. Subclasses of `ObservationManagerC` allow variations on:–
   - Sampling methods
   - Storage of data points
2. A specific subclass of `FitToSampleC` is necessary to fit the specific model parameters to a sample.
3. Subclasses of `EvaluateSolutionC` allow:–
   - Different voting methods, e.g. MLESAC (Torr & Zisserman CVIU'00).
   - Efficient evaluation methods, e.g. Randomised RANSAC (Chum & Matas BMVC'02).

Robust least squares

Assume we have k noisy measurements (data points) $\mathbf{z}(j)$ on the vector $\mathbf{x}$ of model parameters:–

$$\mathbf{z}(j) = \mathbf{h}(j; \mathbf{x}) + \mathbf{w}(j), \quad j = 1, \ldots, k$$

For inlier measurements $\mathbf{w}(j)$ can be modelled as zero mean Gaussians with covariances $N(j)$.

We maximise the likelihood of the $\mathbf{z}(j)$ given $\mathbf{x}$:

$$\mathbf{x} = \arg\min \left(\sum_{j=1}^{k} (\mathbf{z}(j) - \mathbf{h}(j; \mathbf{x}))^\top N(j)^{-1} (\mathbf{z}(j) - \mathbf{h}(j; \mathbf{x})) \right)$$

The Levenberg-Marquardt algorithm is a good iterative solver for $\mathbf{x}$.

The Levenberg-Marquardt algorithm

- Start with an estimate $\mathbf{x}^-$ of $\mathbf{x}$.

- Iteratively update the estimate to $\mathbf{x}^+$:

$$\mathbf{x}^+ = \mathbf{x}^- + A^{-1}\mathbf{a}, \quad \text{where}$$

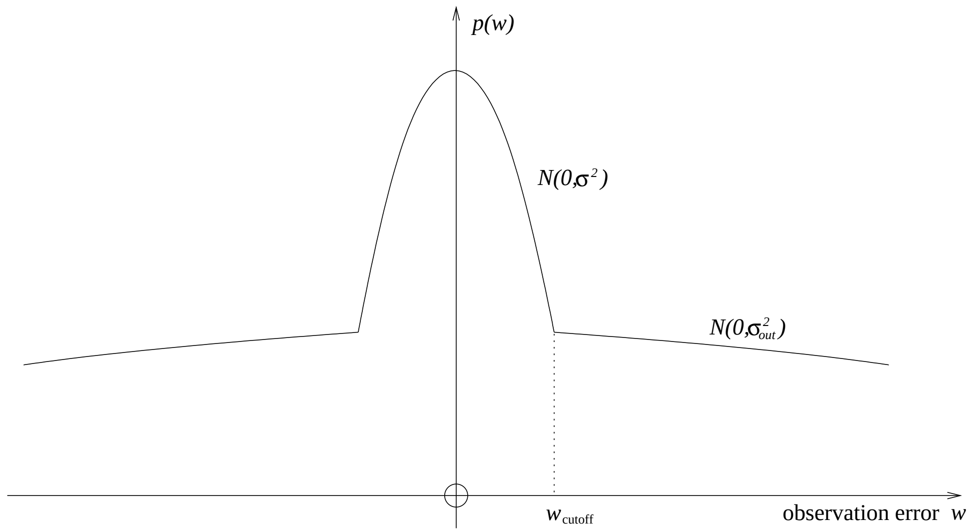
$$\begin{aligned} A &= \sum_j H(j)^\top N(j)^{-1} H(j) + \lambda I, \\ \mathbf{a} &= \sum_j H(j)^\top N(j)^{-1} (\mathbf{z}(j) - \mathbf{h}(j\,\mathbf{x}^-)) \quad \text{and} \\ H(j) &= \left.\frac{\mathrm{d}h(j)}{\mathrm{d}\mathbf{x}}\right|_{\mathbf{x}^-}, \quad \text{the Jacobian matrix of } \mathbf{h}(j). \end{aligned}$$

- λ is a damping parameter.

- $H(j)$ can be computed symbolically or numerically.

Robustified Levenberg-Marquardt

Modify the Gaussian error distribution to a combination of two Gaussians (bi-Gaussian):



- Other error PDF's are common, but less suited to Levenberg-Marquardt.
- The bi-Gaussian model works well for "close" outliers.
- Only simple changes required to the basic Levenberg-Marquardt algorithm.

Robustified Levenberg-Marquardt in Ravl

- The `StateVectorC` class encapsulates the parameter vector $\mathbf{x}$.
- The `ObsVectorC` subclass encapsulates a measurement vector $\mathbf{z}$ together with its error covariance N.
- The `ObservationExplicitC` subclass of `ObservationC` encapsulates a single measurement (data point)

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{w}$$

  It contains an `ObsVectorC` representing $\mathbf{z}$ and N, plus a method for evaluating $\mathbf{h}(.)$ on a particular subclass of `StateVectorC`.
- The `ObservationImplicitC` subclass of `ObservationC` encapsulates a single measurement (data point) of the implicit form

$$\mathbf{F}(\mathbf{x}, \mathbf{z} - \mathbf{w}) = \mathbf{0}$$

- The `ObsVectorBiGaussianC` subclass of `ObsVectorC` encapsulates a measurement $\mathbf{z}$ with a bi-Gaussian error N/N_{out}.

Relationships to other algorithms

- RANSAC's closest competitor is the Hough transform:
  - Hough transform applies exhaustive search.
  - For high dimensional spaces RANSAC is faster.
- Robustified Levenberg-Marquardt is a special case of an M-estimator.
  - M-estimators normally implemented using reweighted least-squares. Yuck!
- Block-vector version of Levenberg-Marquardt is the best way to implement:–
  - Bundle adjustment.
  - Recursive parameter estimation.

  Throw away the Kalman filter!

Projective 2D motion example

Rearrange the projective motion equation:

$$\begin{aligned} x' &= z' \frac{P_{xx}x + P_{xy}y + P_{xz}z}{P_{zx}x + P_{zy}y + P_{zz}z} \\ y' &= z' \frac{P_{yx}x + P_{yy}y + P_{yz}z}{P_{zx}x + P_{zy}y + P_{zz}z} \end{aligned}$$

This can be written as

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{w}$$

where

- $\mathbf{x}$ contains the elements of P.
- $\mathbf{z}$ is identified as $\begin{pmatrix} x' \\ y' \end{pmatrix}$.
- x, y are treated as error-free independent variables.

Noise in x, y can be modelled using the implicit form

$$\mathbf{F}(\mathbf{x}, \mathbf{z} - \mathbf{w}) = \mathbf{0}, \quad \text{where} \quad \mathbf{z} = \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix}$$

Example classes

- For robust line fitting (orthogonal regression): classes `StateVectorLine2dC`, `Point2dObsC`, `ObservationLine2dPoint`, `FitLine2dPointsC`.

- For robust projective 2D motion estimation: classes `StateVectorHomog2dC`, `ObservationHomog2dPoint`, `ObservationImpHomog2dPoint`, `FitHomog2dPointsC`.

- For robust affine 2D motion estimation: classes `StateVectorAffine2dC`, `FitAffine2dPointsC`, `ObservationAffine2dPoint`.

- For robust quadratic curve fitting: classes `StateVectorQuadraticC`, `FitQuadraticPointsC`, `ObservationQuadraticPoint`, `ObservationImpQuadraticPoint`.

Shrink-wrapped routines

Example: fitting 2D projective motion to pairs of points in two images.

```
const StateVectorHomog2dC
Optimise2dHomography ( DListC<Point2dPairObsC> &matchList,
                       RealT zh1=1.0, RealT zh2=1.0,
                       RealT varScale=10.0,
                       RealT chi2Thres=5.0,
                       UIntT noRansacIterations=100,
                       RealT ransacChi2Thres=3.0,
                       RealT compatChi2Thres=5.0,
                       UIntT noLevMarqIterations=10,
                       RealT lambdaStart=0.1,
                       RealT lambdaFactor=0.1 );
```

This invokes RANSAC and robustified Levenberg-Marquardt in one tidy routine.

Example application: Mosaicing

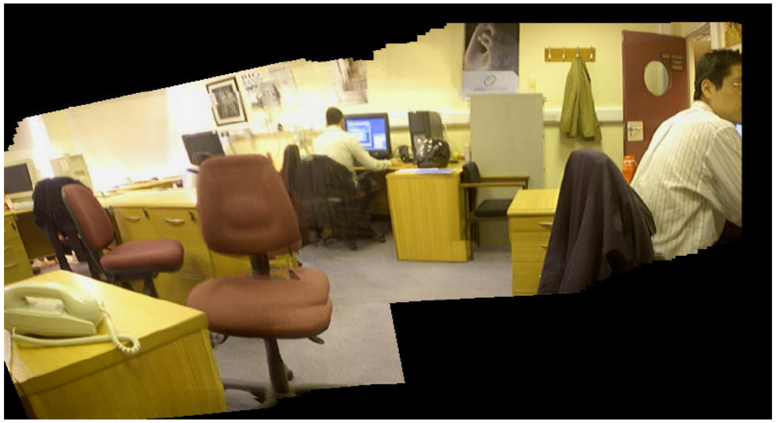
You can register images of a plane or from a rotating camera using 2D methods.

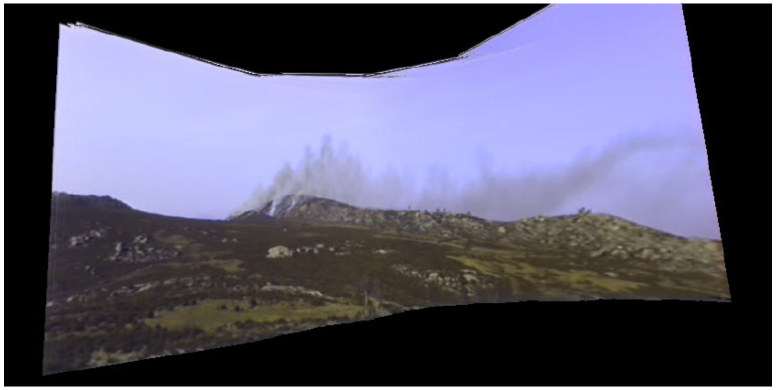
Mosaicing (continued)

Mosaicing algorithm:

1. Select a reference coordinate frame, usually the first image.
2. Track corner features independently (thanks Charles).
3. Compute 2D projective motion between consecutive images using RANSAC and robust Levenberg-Marquardt.
4. Using the accumulated product of the P matrices, warp new images back to the reference coordinate frame.
5. Insert the warped image into the mosaic.
6. Median filter the pixels to remove moving objects.

More mosaicing results





Foreground segmentation

Now you can register images to the mosaic rather than each other

This allows you to do a bit of difference keying:

Conclusions

- RANSAC and robustified Levenberg-Marquardt make a good combination.
- You need to have a reasonable estimate of the inlier errors and the outlier rate.
- Just plug in a few subclasses and you're away.
- Ravl is great!!
- It just needs a few bits and bobs to make it... cosmic!

TODO list

1. Camera classes including distortion models, all nicely templated.

2. Some matrix routines, e.g. Cholesky factorisation.

3. Make the GUI nicer & integrate the foreground separator.

Any volunteers?