Tracking in LCTPC/MarlinTPC

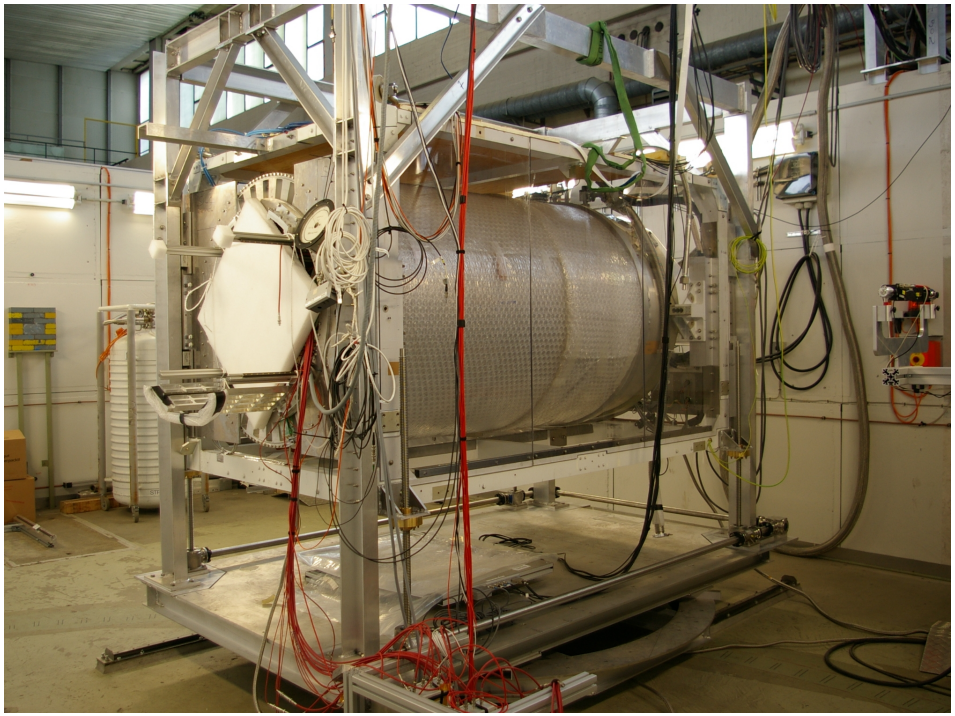
Ch. Rosemann

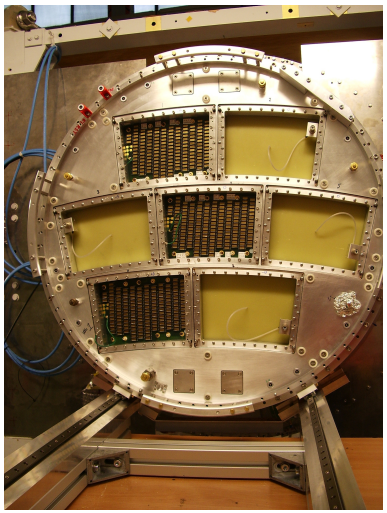
DESY
FLC-TPC

02. February 2011
ILD Optimization Meeting

LCTPC and the Large Prototype

- LCTPC (mostly) Hardware collaboration to build TPC for ILC
- Uses the *Large Prototype*:
  - $\emptyset_i = 72\,\text{cm}$, $l = 61\,\text{cm}$
  - can host up to 7 modules
  - operated in 1T field
  - DESY II $e^{+/-}$ 1-6 GeV test beam
- Already several periods of data taking
- MarlinTPC aim:
  - Common reconstruction software
  - Performance evaluation of different technologies



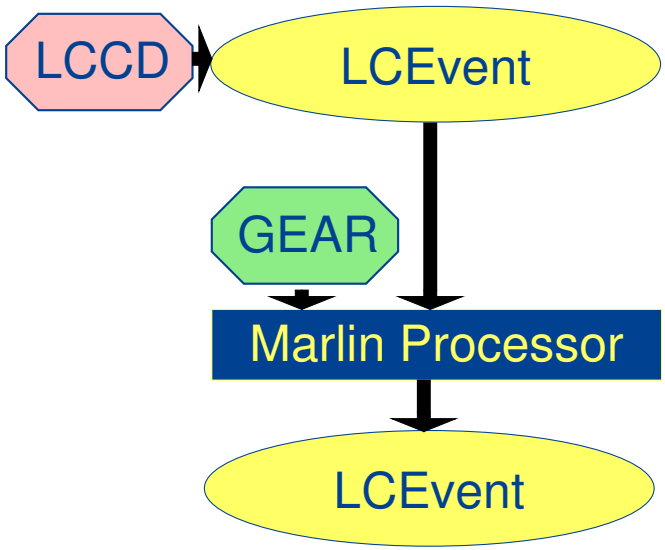


What is MarlinTPC

Current goal: a working full reconstruction chain for (large) prototype data

- Started in 2006/2007, long hibernation
- 28 total *contributors*,
  $\sim$ 12 active *developers*
- Code resides at svn repository
  - Every developer has a *private* branch
  - Latest greatest version is in the *trunk*
  - *tagged releases* from time to time
- Installation via *ilcinstall* or simple checkout
- Sub project of MARLIN, building blocks:
  - Geometry description (GEAR)
  - Data description (LCIO)
  - Conditions Data (LCCD)
    (Our own set of objects)
  - Processor code

The MARLIN structure



Two different approaches

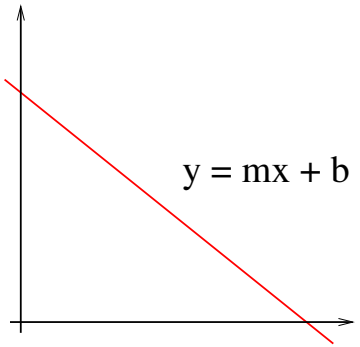
1. Track Finding and Fitting within MarlinTPC
   - Basic code exists:
     Linear Regression, *Simple*-χ^2
   - Not really used, no errors (!), not stable, no maintainer
   - Problem: hard to steer with the internal dependency
2. Finding&Fitting as external package
   - Natural way to do find&fit as unit
   - Needs interface in MarlinTPC (back & forth)
   - Two implementations exist
   - KalmanFilter Fitter by Japanese colleagues
     (also an option for general tracking)
     complemented for the Large Prototype with a simple single track finder
   - Hough Transformation finding plus χ^2 fitter

- Choice 2 is taken, with different consequences for the two options
- MarlinTPC is used for everything up to space point reconstruction

Hough Trafo: straight lines

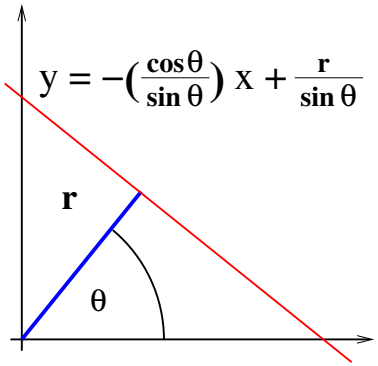
High school math: straight lines

$$y = m \cdot x + b$$



Re-parametrize to

$$y = \left(-\frac{\cos\Theta}{\sin\Theta} \right) \cdot x + \frac{r}{\sin\Theta}$$



Parameter r: closest distance to the origin
Parameter Θ: the angle between x-axis and the vector to the closest point

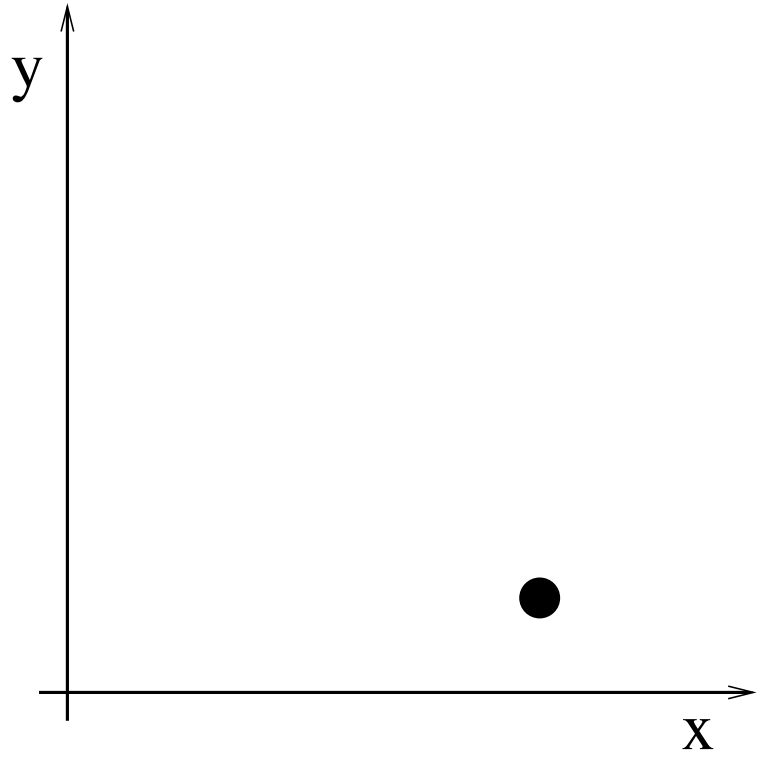
Different way to write it:

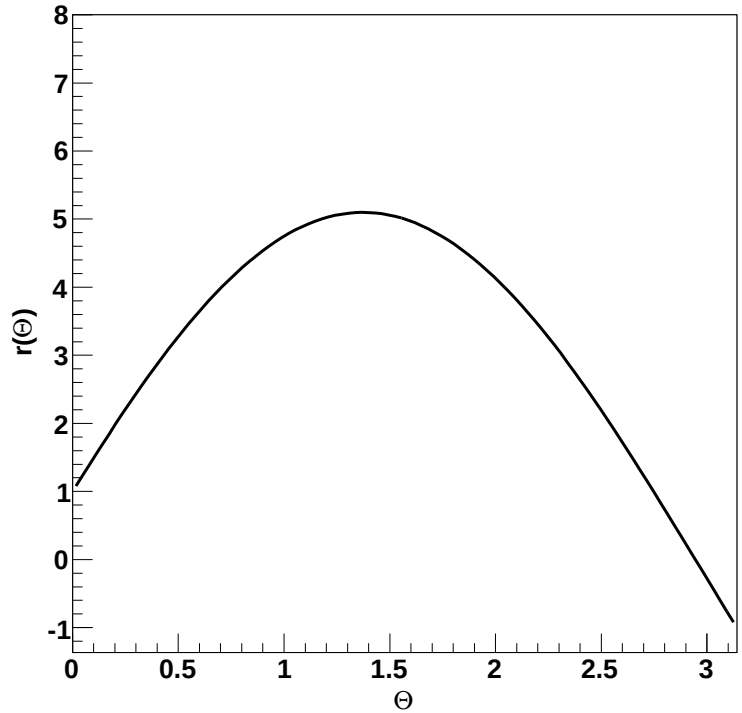
$$r(\Theta) = x \cdot \cos\Theta + y \cdot \sin\Theta$$

Use this as a transformation rule from $(x, y) \rightarrow (r, \Theta)$

Step 1

A point in the xy plane becomes a sine function in $r\Theta$ space

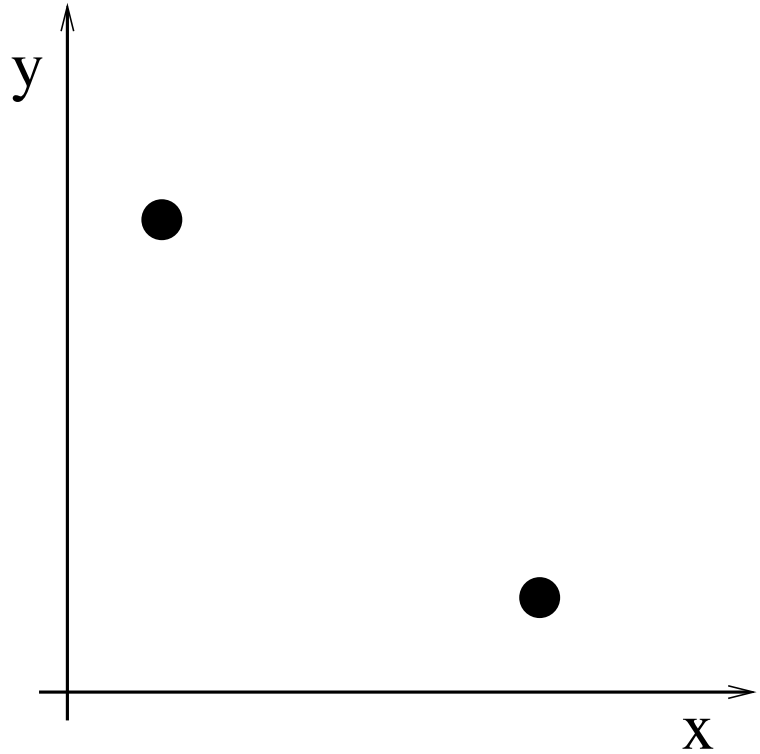


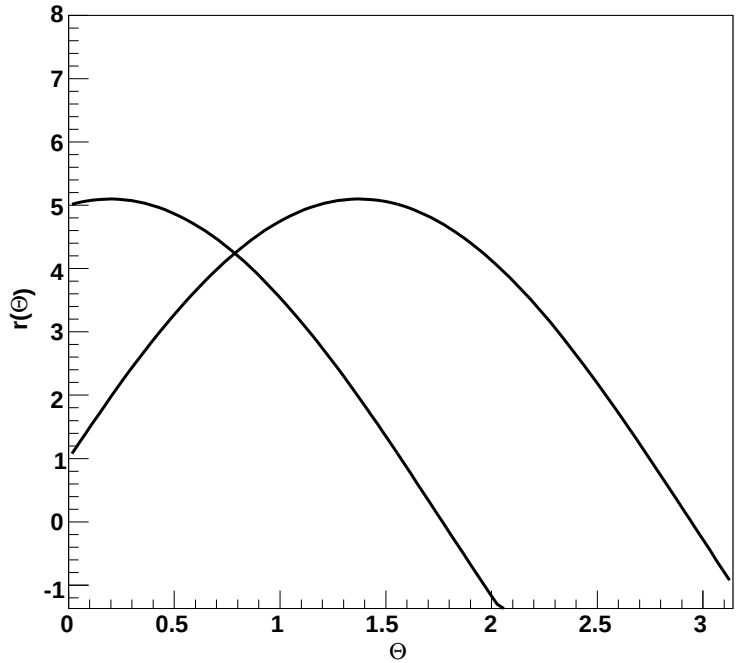


$$r(\Theta) = x \cdot \cos\Theta + y \cdot \sin\Theta$$

Step 2

Take a second point in the xy plane – another sinusoidal in $r\Theta$ space

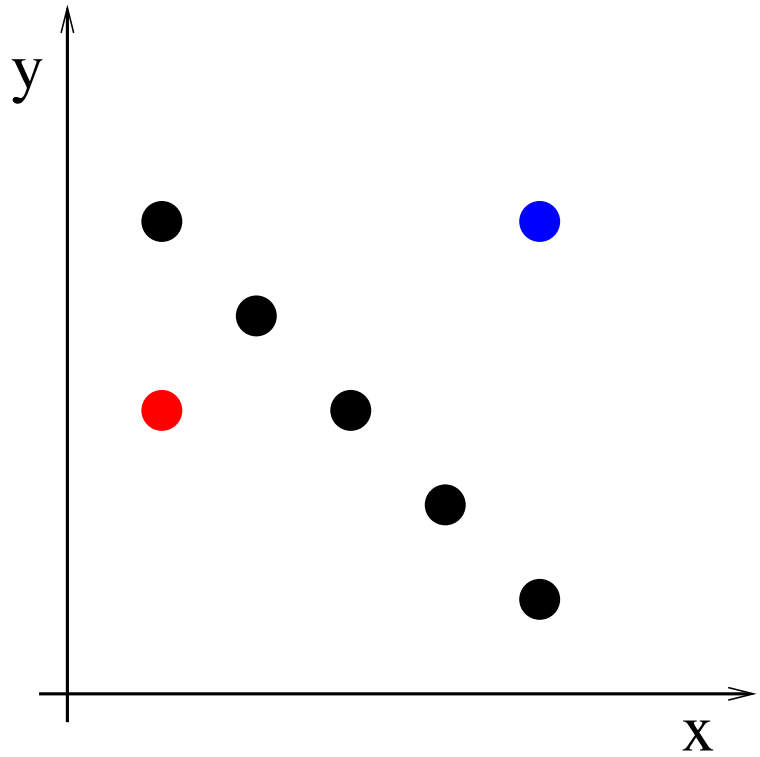


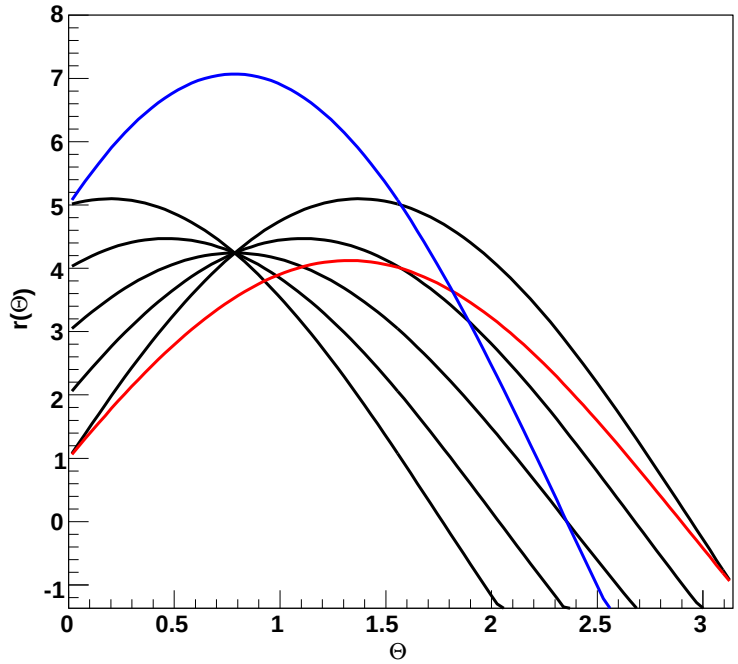


They intersect at one point in $r\Theta$!
This is the right value for the two parameters that determine the straight line connecting the points!

Step 3

Further points in the xy plane – lots of sundials in $r\Theta$ space



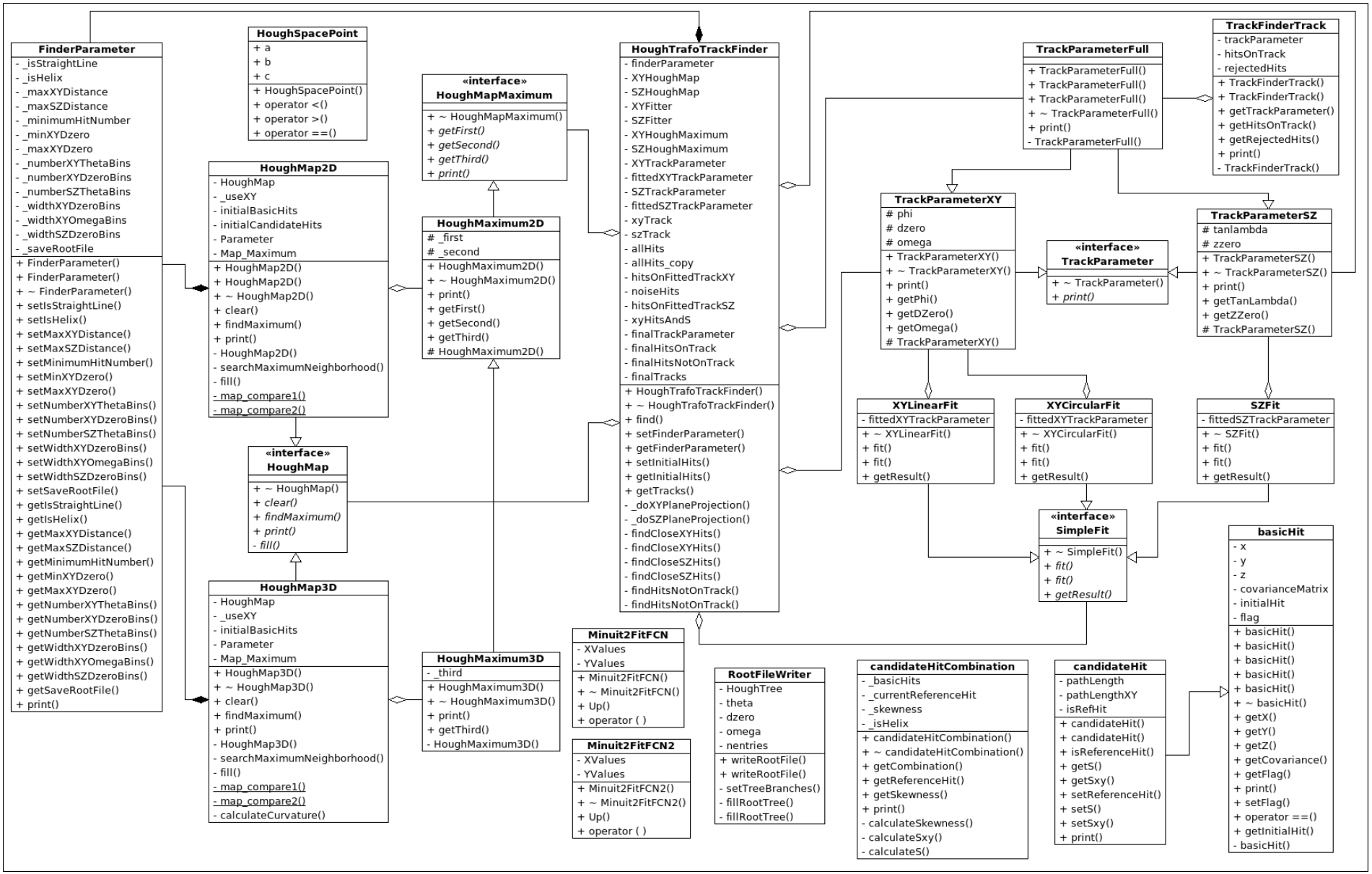


Each set of points belonging to one straight line intersect at one point!

Status

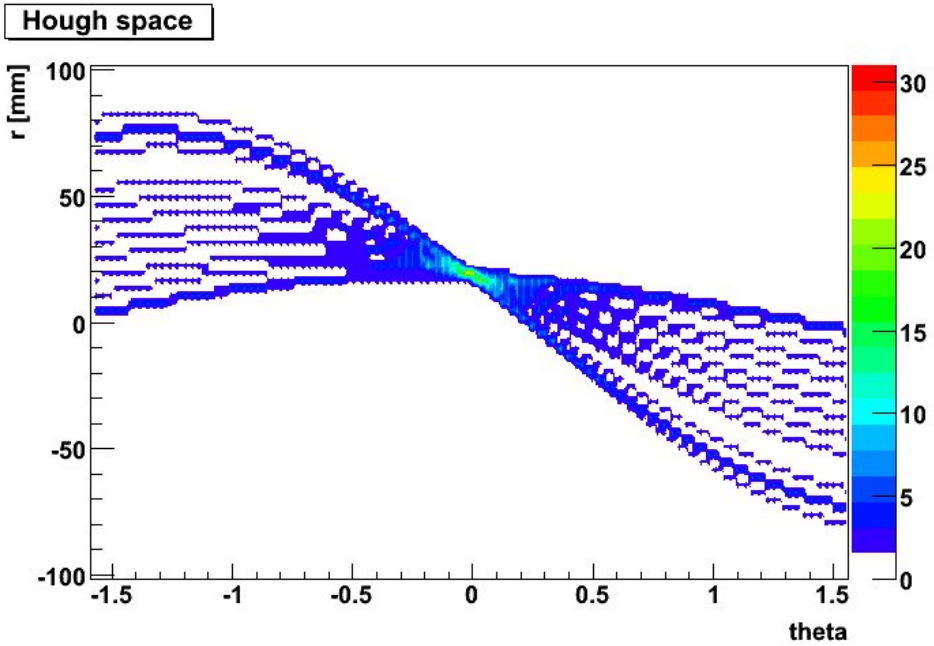
- Hough Transformation in general possible for any shape that can be parametrized
- Computationally expensive for higher dimensions
- Algorithm here:
  1. Search for circle/straight line in *xy*-projection
  2. Find maximum in Hough space
  3. Search for the corresponding hits
  4. Calculate arc length *s*
  5. Find straight line in *sz* projection
- Code basis is built
- Works for straight lines, helices and curlers
- No external constraints
- Extension with χ^2 fitter currently in progress
- Currently benchmarking the basic code

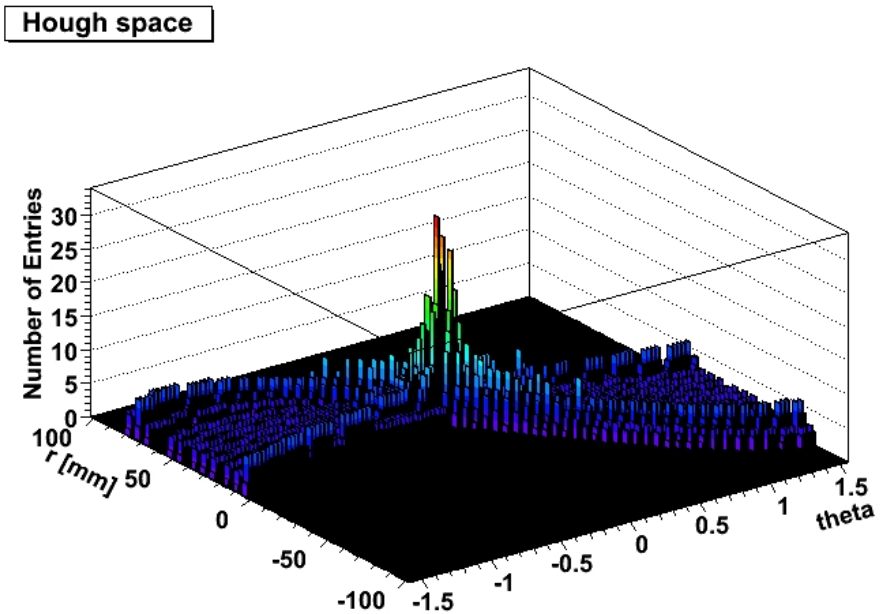
UML diagram for Hough Transformation code



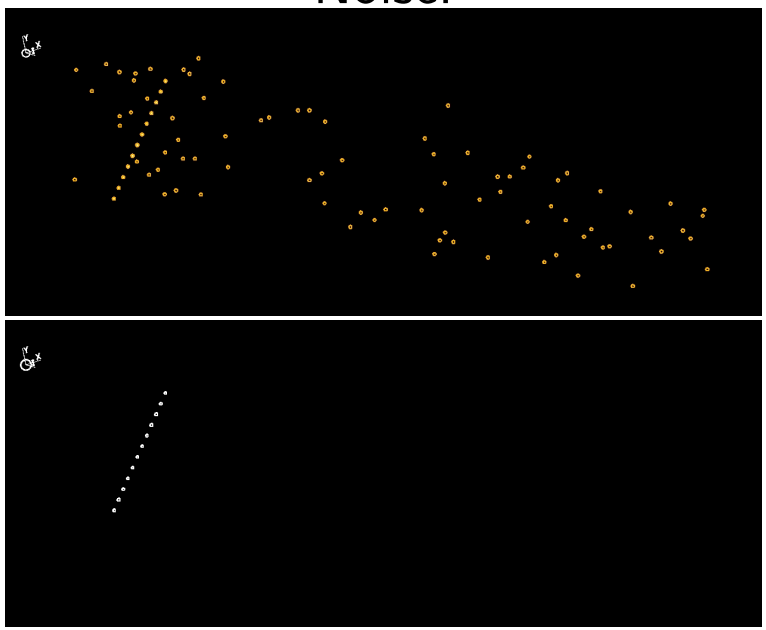
Some example/performance plots

Hough Space dimensions:





Noise:



High multiplicity: