CHEP 2019

Aligning the MATHUSLA Detector Test Stand with TensorFlow

M. Proffitt, E. Torro, G. Watts

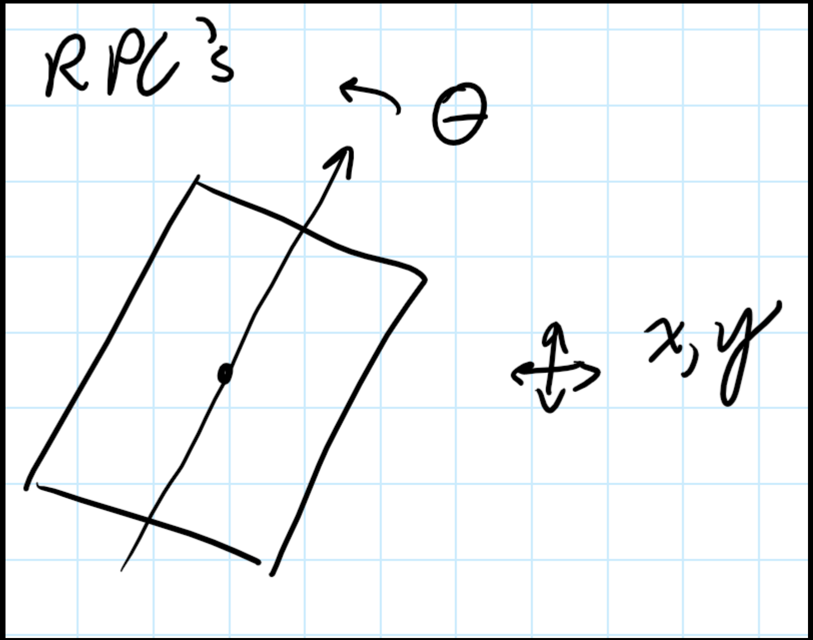
iris hep
Institute for Research & Innovation in Software for High Energy Physics

Detector Alignment

How do you know the angle at which a charge track travels through your detector?

You know where all the readout channels are located in (x,y,z)

How precisely you know this governs your resolution



1. Construction

   Accurate placement of the detector planes, the detectors within the planes, the readout channels relative to the planes, etc.

2. Measurement

   After construction of each independent piece you make accurate measurements and build a geometry model

At the very least you need a cross check…

Detector Alignment

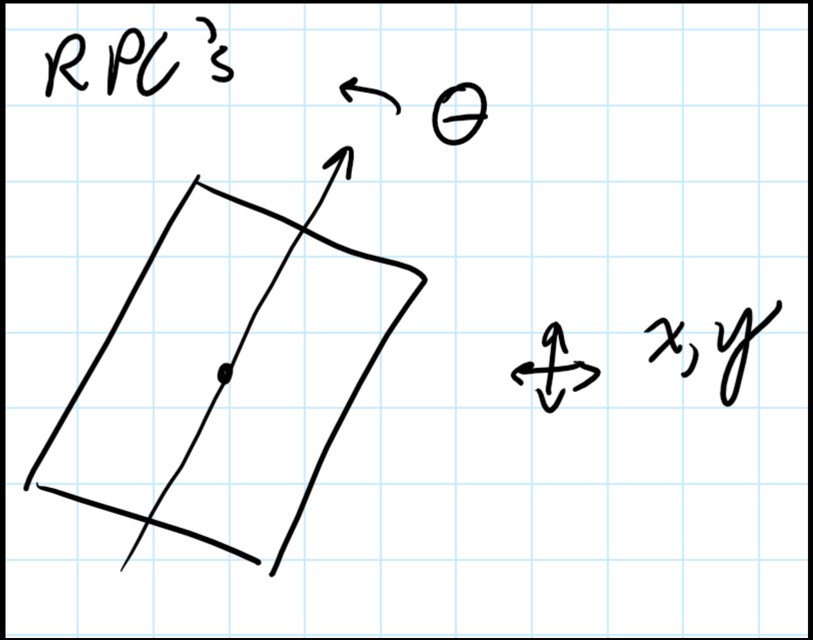
A Giant Fitting Problem

3

Aligning Using Physics

A set of tracks whose trajectory you know

Resolution means you can’t just solves this analytically



So try to adjust the detector to best represent some large numbers of trajectories as straight.

Till your track residuals are as small as possible

This is a giant fitting problem: adjust the detector until as many tracks fit well as possible

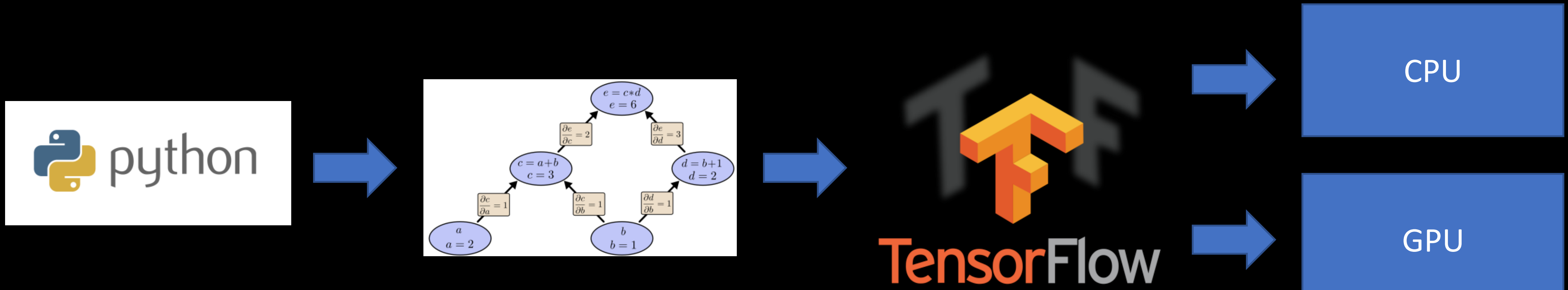
Why use TensorFlow?

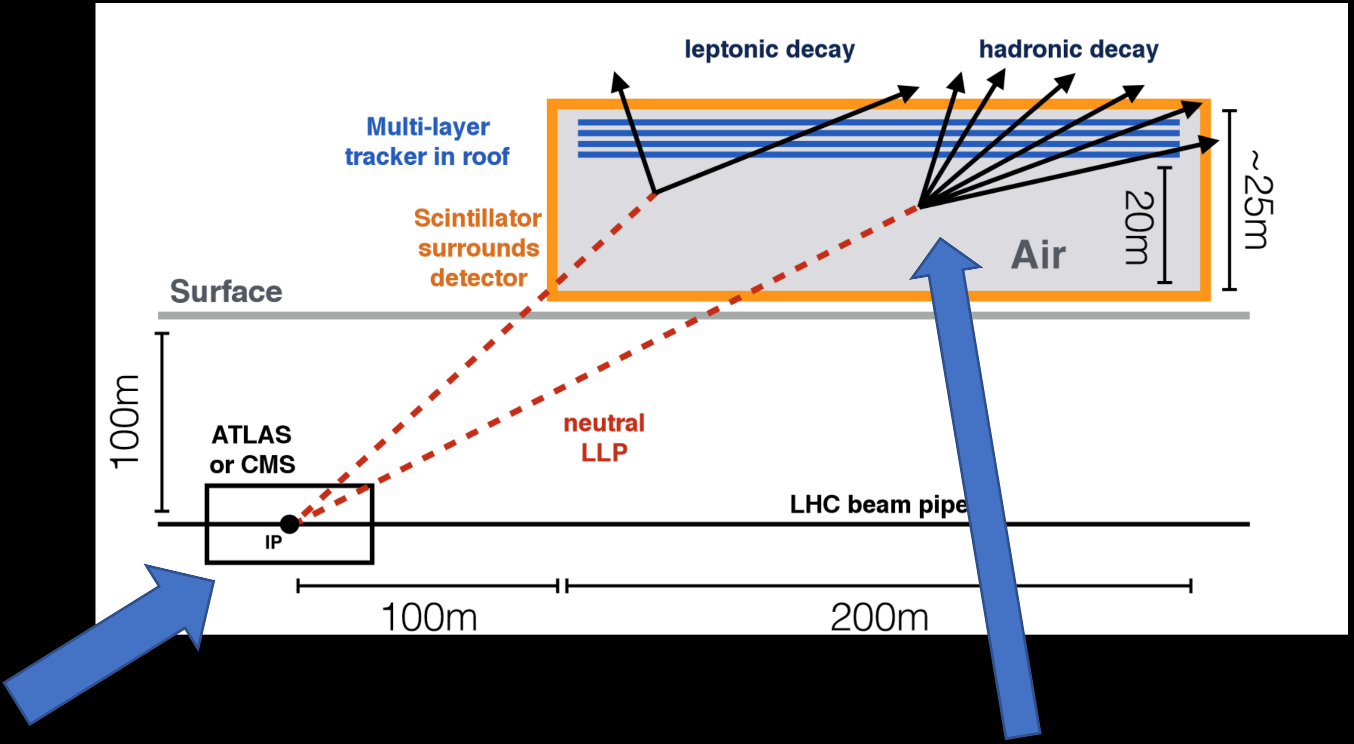
1. Alignment is a giant minimization problem
2. NN's are a giant minimization problem

→ Force me to learn the basics of how TF works

Felt a bit like RooStats and RooFit

Code I started from was in FORTRAN, translated to C++!!





An Ultra-Long-Lived-Particle (ULLP) produced at the LHC interaction point…

… Decays on the surface in MATHUSLA, a 5 or 6 layer tracking chamber with veto scintillator around the edges

MATHUSLA Test Stand

Run above the ATLAS IP:

- Took data 2017-2018
- 3 double layers of Resistive Plate Chambers (RPC's) for track reconstruction
  - From the ARGO experiment
- A top and bottom layer of Scintillator paddles
  - From the DZERO experiment's forward muon chambers

Goals:

- Can we reconstruct tracks from Cosmic Rays
- Can we reconstruct upward going tracks?
- Can we tell the difference between them?

Status:

- We have millions of tracks
- GEANT4 based simulation of the test stand
- Geometry measured by **hand.**
- Finalizing data for a paper.

Workflow

Detector Model

Track Fits

χ^2

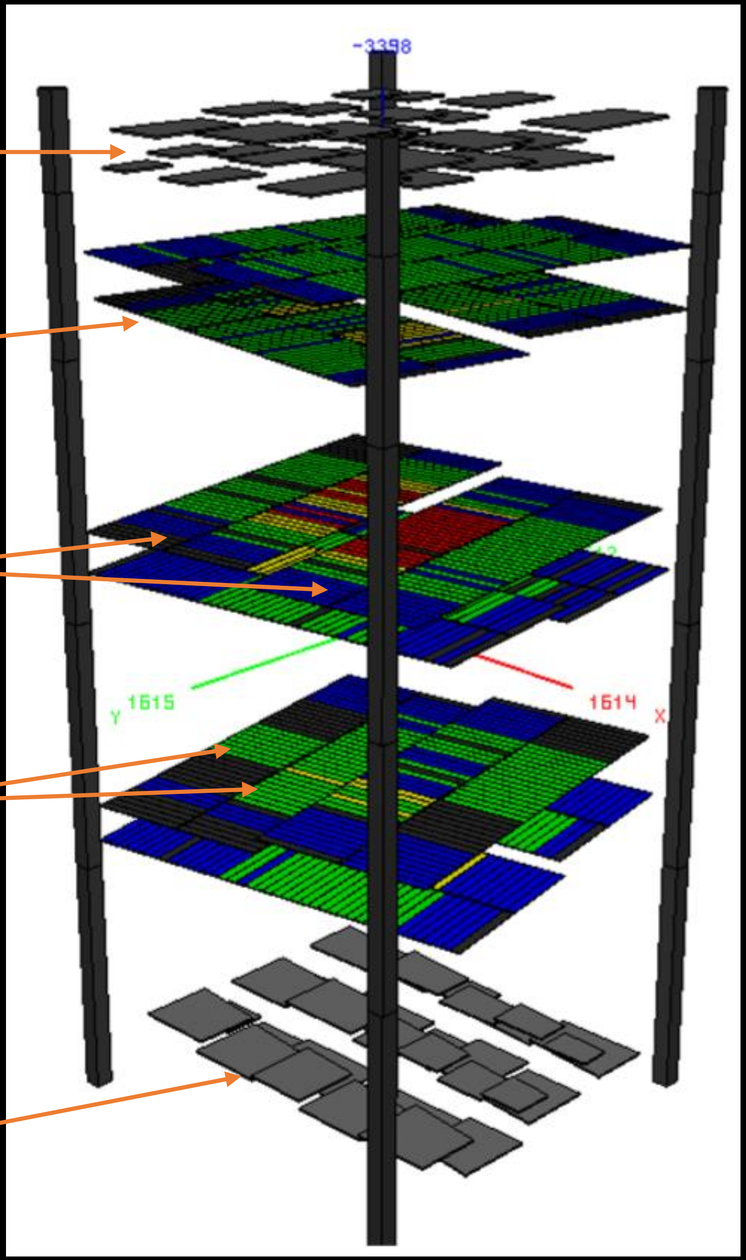
Scintillator Paddles
(not used in track fit)

Double Layer of RPC's

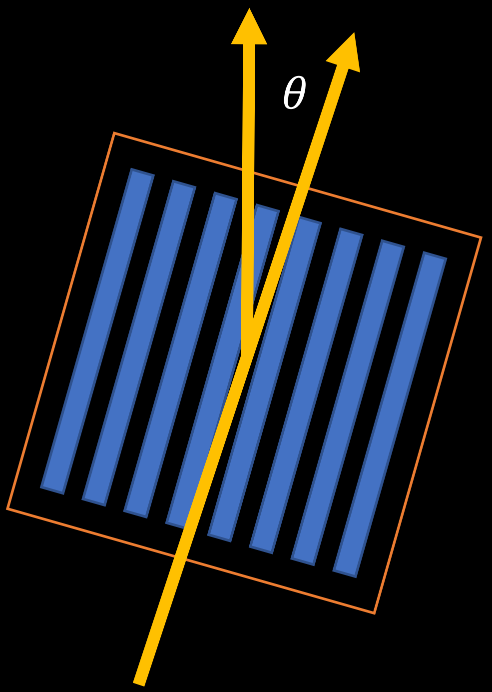
Each layer consists of two RPC Planes

Each RPC Plane consists of 8 RPC detectors

Scintillator Paddles
(not used in track fit)



Detector Model



Each plane:

- Shift center by (x,y)
- Can rotate in the (x,y) plane
- RCP #0 is fixed in place

Rotations and displacement happen per detector

→ But must be translated into per-strip

```
def strip_locations_xy(pos_x, pos_y, rot_rz, offset_xy=None, offset_rot=None, leave_centered=False):
    # For each RPC find the center as a (x,y) tuple. This is in global coordinates
    rpc_centers = [rpc_center(pos_x, pos_y, i) for i in range(0,rpcg.n_rpcs)]

    # Transform all the x,y positions to be relative to the centers.
    center_offset = np.zeros((rpcg.n_total_strips,2), dtype=np.float32)
    for rpc_id,xy_center in enumerate(rpc_centers):
        start = rpcg.rpc_first_index(rpc_id)
        end = rpcg.rpc_first_index(rpc_id+1)
        center_offset[start:end] = xy_center
    raw_strip_locations = np.concatenate((np.reshape(pos_x, (-1,1)), np.reshape(pos_y, (-1,1))), axis=1)
    strip_centers = raw_strip_locations-center_offset

    # Apply a rotation to the location for each RPC if sliding is allowed.
    strip_rotations = rot_rz
    rotated_center = strip_centers
    if offset_rot is not None:
        strip_rot_thetas = extend_to_strips(offset_rot)
        strip_rotations += tf.reshape(strip_rot_thetas, (-1,))
        rotated_center = calc_point_rotation (strip_centers, strip_rot_thetas)

    # The center of the RPC can slide, so put that in.
    if offset_xy is not None:
        offset_xy = extend_to_strips(offset_xy)

    rotated_slide_center = rotated_center if offset_xy is None else rotated_center + offset_xy

    # And finally add those together and return the actual position of the RPC.
    final_strip_location = rotated_slide_center if leave_centered else rotated_slide_center+center_offset
    return (final_strip_location, strip_rotations)
```

Code is complex because:

- Each plane is rotated
- Which must be translated to each strip
- And there are 8 detectors per plane, and 10 strips in each detector
- Need to use tf.stack, tf.reshape a lot!

Workflow

Detector Model

↓

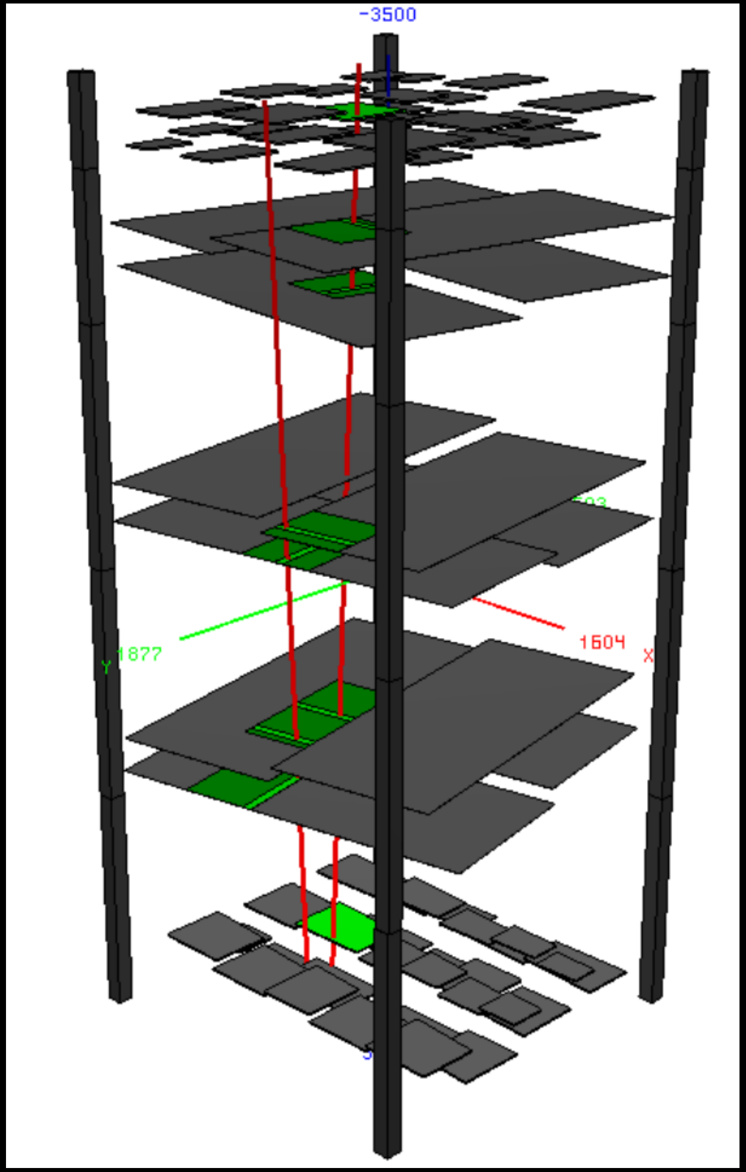
Track Fits

↓

χ^2

1. Straight line fit:
   - Analytical solution
   - Matrix inversion
   - TF should be great at this
   - Hard (for me) because 3D tensors

2. Build a χ^2 that involves the solutions:
   - Slope and offset in both transverse planes
   - Minimize the points to slope distance



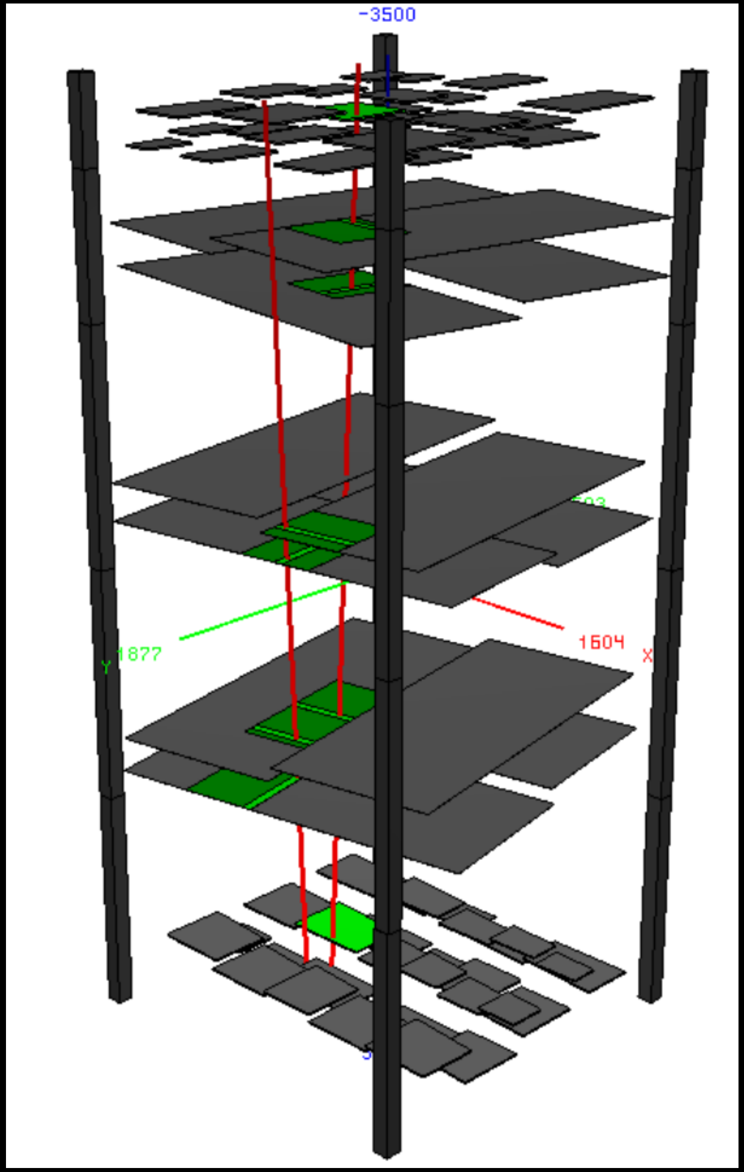
Workflow

2

Build a χ^2 that involves the solutions:

- Slope and offset in both transverse planes
- Minimize the points to slope distance

Do hit-track association once!



How do we represent the tracks?

In TensorFlow everything is by matrices!

1. Each strip is a column
2. Tracks have 1's in the strips they hit
3. Second set of matrices are the strip locations
4. Track χ^2 is calculated as a a function of multiplying the two matrices

Memory efficiency?

Workflow

Detector Model

↓

Track Fits

↓

χ^2

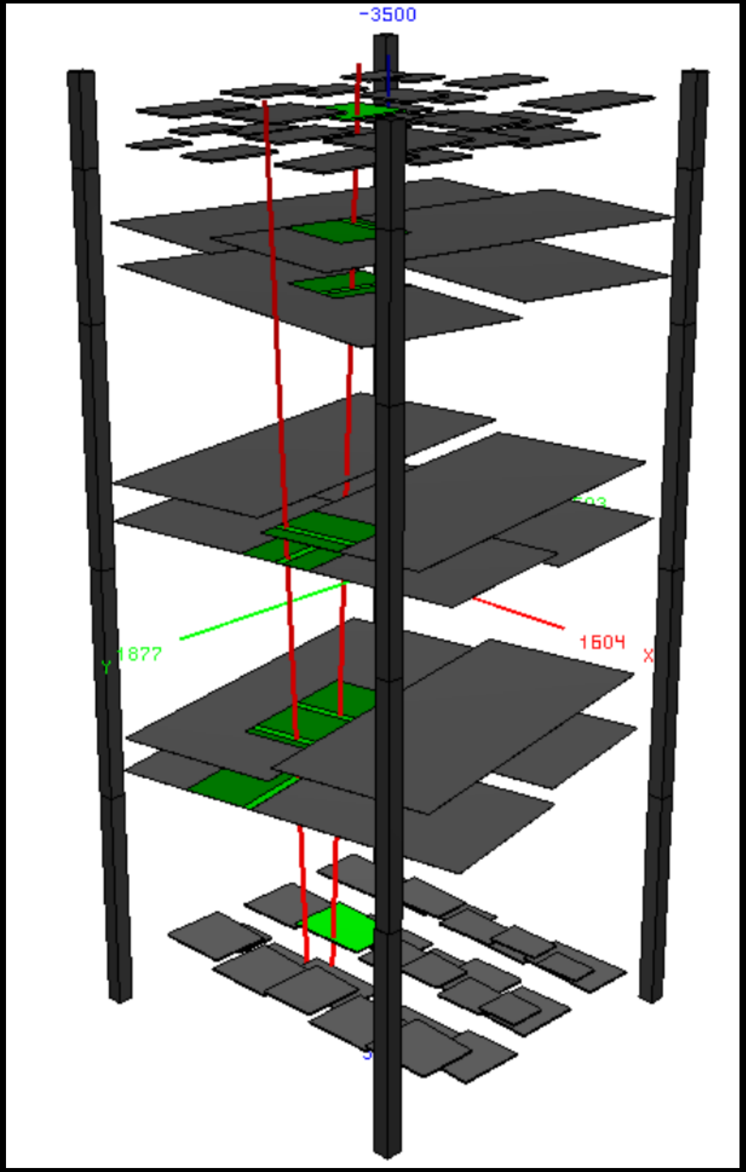
Terms in the χ^2:
- Each point in the x-line fit
- Each point in the y-line fit

↓

Contain the locations of each strip

↓

Contains the (x,y,θ) of the planes



χ^2

Calculation is straightforward

- No need to change shapes as everything is about tracks
- Use methods to hide uglyness

```
strip_cos_y = np.sin(strip_rz) if type(strip_rz) is np.ndarray else tf.sin(strip_rz)
strip_cos_x = np.cos(strip_rz) if type(strip_rz) is np.ndarray else tf.cos(strip_rz)

Lx = strip_cos_x
Ly = strip_cos_y
Wx = -strip_cos_y
Wy = strip_cos_x

# And the width ratios.
del_L = strip_widths[0] / math.sqrt(12)
del_W = strip_widths[1] / math.sqrt(12)
strip_ratio_Lx = Lx / del_L
strip_ratio_Ly = Ly / del_L
strip_ratio_Wx = Wx / del_W
strip_ratio_Wy = Wy / del_W

# Calculate the ratio for the length and width in (x,y)
ratio_Lx = expand_to_tracks(strip_ratio_Lx, len(hits_used))
ratio_Ly = expand_to_tracks(strip_ratio_Ly, len(hits_used))
ratio_Wx = expand_to_tracks(strip_ratio_Wx, len(hits_used))
ratio_Wy = expand_to_tracks(strip_ratio_Wy, len(hits_used))

# Calc the delta between the strip position and track for each hit
delta_x = calc_delta (strip_location[:,0], strip_location[:,2], x0[0], m[0], hits_used)
delta_y = calc_delta (strip_location[:,1], strip_location[:,2], x0[1], m[1], hits_used)

# Calculate the contributions to the strip chi2
length_error = delta_x * ratio_Lx + delta_y * ratio_Ly
width_error = delta_x * ratio_Wx + delta_y * ratio_Wy

# Do the per-element squaring (this is common def of tensor operations in numpy and tf)
# NOTE: As an experiment, we remove the length error as it is causing the detectors
# to slide. We need to better document this.
# TODO
return width_error**2 + length_error**2
```

Rotation from slight stereo angle


THINK
DIFFERENT

All Tracks at the same time

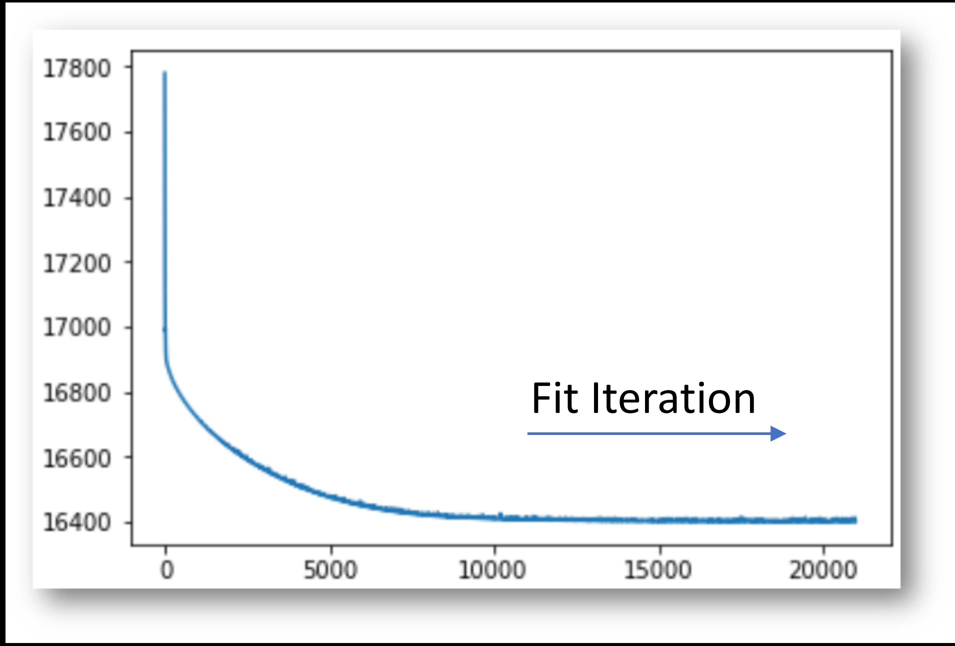
Treating information of different dimensionalities

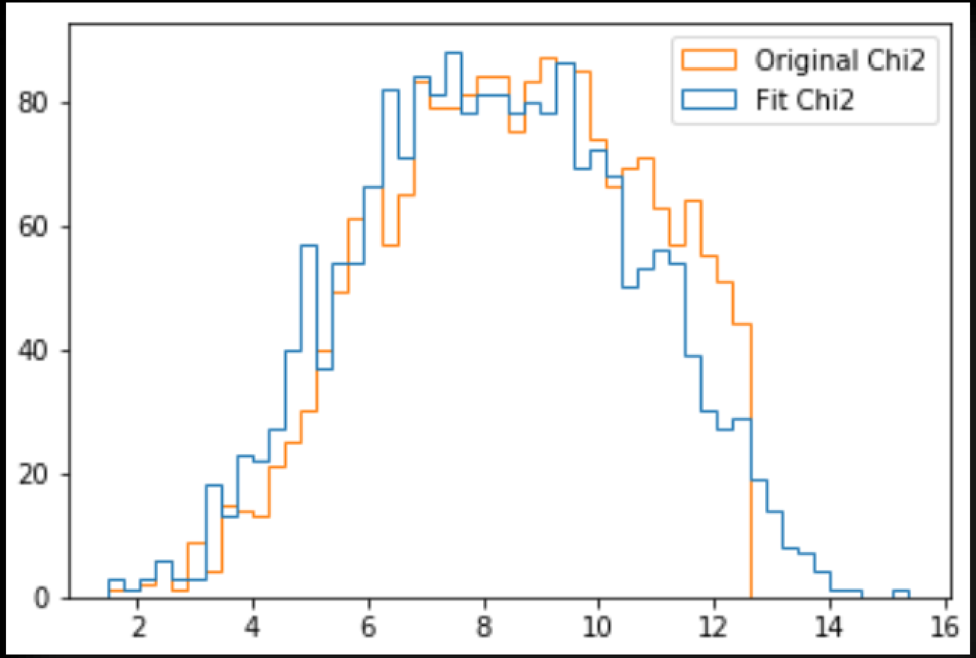
Treating information of different dimensionalities

e.g. 12 angles have to propagate to 960 strips

Could not figure out Unit Tests

Conclusions





Conclusions

- Technique Works!
- Large slews are due to geometry!
- Required very different thinking to fit TF's programming model
  - All tracks simultaneously
  - Hits aren't indices into an array, but are 1's and 0's in a matrix
- Code is concise
  - Gitlab
  - Not a generic toolkit however
- GPU is slower… optimization in progress
- Future work
  - Some FORTRAN matrix operations are still unwound
  - Understand GPU inefficiencies
  - Increase number of tracks
  - Other HEP packages
  - TF 2.0, and immediate mode for unit tests (?)
  - Address length-wise sliding

	dx	dy	theta
0	0.000000	0.000000	0.000000
1	62.149452	8.070271	-0.005316
2	20.052612	-17.888035	0.027767
3	38.217182	3.451782	0.008647
4	-23.514259	12.853182	-0.002025
5	40.880676	-5.684084	-0.001107
6	6.830047	-16.709791	0.024888
7	31.081161	8.822522	0.010519
8	12.655775	3.564259	-0.008935
9	-42.947567	12.582623	0.013111
10	-9.767097	-37.386765	0.035398
11	24.999971	29.194012	0.007217