Aligning the MATHUSLA Detector Test Stand with TensorFlow

M. Proffitt, E. Torro, G. Watts
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

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

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

Felt a bit like RooStats and RooFit

```
Python
```

```
TensorFlow
```

```
CPU
```

```
GPU
```
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

$\chi^2$

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

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

1

Track Fits

Build a $\chi^2$ that involves the solutions:
  • Slope and offset in both transverse planes
  • Minimize the points to slope distance

$\chi^2$

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

Build a $\chi^2$ 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 $\chi^2$ is calculated as a function of multiplying the two matrices

Memory efficiency?
Workflow

Detector Model

Track Fits

Terms in the $\chi^2$:
• Each point in the x-line fit
• Each point in the y-line fit

Contains the locations of each strip

Contains the $(x,y,\theta)$ of the planes

$\chi^2$
\( \chi^2 \)

Calculation is straightforward
- No need to change shapes as everything is about tracks
- Use methods to hide ugliness

\[
\begin{align*}
\text{strip} \cos_y &= \text{np.sin} (\text{strip} \text{rz}) \text{ if } \text{type} (\text{strip} \text{rz}) \text{ is np.ndarray} \text{ else } \text{tf.sin} (\text{strip} \text{rz}) \\
\text{strip} \cos_x &= \text{np.cos} (\text{strip} \text{rz}) \text{ if } \text{type} (\text{strip} \text{rz}) \text{ is np.ndarray} \text{ else } \text{tf.cos} (\text{strip} \text{rz}) \\
\text{Lx} &= \text{strip} \cos_x \\
\text{Ly} &= \text{strip} \cos_y \\
\text{Wx} &= -\text{strip} \cos_y \\
\text{Wy} &= \text{strip} \cos_x \\
\end{align*}
\]

# And the width ratios.
\[
\begin{align*}
\text{del}_L &= \text{strip} \text{widths}[0] / \text{math.sqrt}(12) \\
\text{del}_W &= \text{strip} \text{widths}[1] / \text{math.sqrt}(12) \\
\text{strip} \text{ratio} \text{Lx} &= \text{Lx} / \text{del}_L \\
\text{strip} \text{ratio} \text{Ly} &= \text{Ly} / \text{del}_L \\
\text{strip} \text{ratio} \text{Wx} &= \text{Wx} / \text{del}_W \\
\text{strip} \text{ratio} \text{Wy} &= \text{Wy} / \text{del}_W \\
\end{align*}
\]

# Calculate the ratio for the length and width in (x,y)
\[
\begin{align*}
\text{ratio} \text{Lx} &= \text{expand} \text{to} \text{tracks} (\text{strip} \text{ratio} \text{Lx}, \text{len}(\text{hits} \text{used})) \\
\text{ratio} \text{Ly} &= \text{expand} \text{to} \text{tracks} (\text{strip} \text{ratio} \text{Ly}, \text{len}(\text{hits} \text{used})) \\
\text{ratio} \text{Wx} &= \text{expand} \text{to} \text{tracks} (\text{strip} \text{ratio} \text{Wx}, \text{len}(\text{hits} \text{used})) \\
\text{ratio} \text{Wy} &= \text{expand} \text{to} \text{tracks} (\text{strip} \text{ratio} \text{Wy}, \text{len}(\text{hits} \text{used})) \\
\end{align*}
\]

# Calc the delta between the strip position and track for each hit
\[
\begin{align*}
\text{delta} \text{x} &= \text{calc} \text{delta} (\text{strip} \text{location}[:,0], \text{strip} \text{location}[:,2], x0[0], m[0], \text{hits} \text{used}) \\
\text{delta} \text{y} &= \text{calc} \text{delta} (\text{strip} \text{location}[:,1], \text{strip} \text{location}[:,2], x0[1], m[1], \text{hits} \text{used}) \\
\end{align*}
\]

# Calculate the contributions to the strip chi2
\[
\begin{align*}
\text{length} \text{error} &= \text{delta} \text{x} \times \text{ratio} \text{Lx} + \text{delta} \text{y} \times \text{ratio} \text{Ly} \\
\text{width} \text{error} &= \text{delta} \text{x} \times \text{ratio} \text{Wx} + \text{delta} \text{y} \times \text{ratio} \text{Wy} \\
\end{align*}
\]

# 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
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

Fit Iteration

[Graph showing Fit Iteration progression]

[Graph comparing Original Chi2 and Fit Chi2]
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

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