Roman Pots Matrix R&D Meeting – DD4HEP Implementation

Friday, March 10th, 2023

Alex Jentsch

Preliminaries

- The EICrecon issue was worse than I thought, and took some time to fix.
    - When the algorithm was “ported” from Juggler/Gaudi, a “factory” was never written to actually use it.
    - Additionally, the way in which Roman Pots reconstruction must be carried-out is fundamentally at-odds with how EICrecon is intended to function.
- **Good news:** The static matrix implementation is now fully-functional in DD4HEP.
    - https://github.com/eic/EICrecon/pull/536
- Next few slides are to go through what is actually there, and how to use it.

Setting up the environment (assuming Mac + Docker)

Very helpful tutorial page: https://eic.github.io/tutorial-jana2/aio/index.html

- Start eic-shell, and make sure to upgrade the container.

```
./eic-shell --upgrade
```

- Once in eic-shell, setup the environment.

```
source /opt/detectors/setup.sh
```

- Now, let's go ahead and make a test sample with the particle gun.

```
npsim --compactFile ${DETECTOR_PATH}/epic_ip6_arches.xml  --enableGun --gun.particle proton --
gun.energy 275*GeV --crossingAngleBoost -0.025 --gun.thetaMin 0.002 --gun.thetaMax 0.005 --
gun.distribution uniform --runType run --numberOfEvents 1000 --outputFile test.edm4hep.root
```

The filename is intentional – naming it [name].edm4hep.root puts the output into the PODIO format which EICrecon **requires.**

- **This will take a little more than an hour to run, and produce a ~100MB output file.**
- You can change the thetaMin angle, I set it to 2mrad to just ensure the particle hit the detector for now.

Now for the EICrecon part

Very helpful tutorial page: https://eic.github.io/tutorial-jana2/aio/index.html

- **EICrecon part:**
  - Clone the EICrecon repo.
  - Now, perform the compilation.
  - Then, source your installation (assuming you're already in the EICrecon directory).
  - Now, you can run EICrecon with your test input file.
    - It will produce an output file with the name you specify, but if you don't supply an output file, it will produce a file with the default name of "podio_output.root".

```
git clone https://github.com/eic/EICrecon
```

```
cd EICrecon
cmake -S . -B build
cmake --build build --target install -- -j4
```

```
source ./bin/eicrecon-this.sh
```

```
eicrecon test.edm4hep.root -Ppodio:output_file=out.root
```

At this point, you will have a full installation of the current build of EICrecon.

BUT, the container HAS the nightly build, so why do we want our own installation? → So we can make our changes and upgrades!

So what exists for the Roman Pots now?

Starting in EICrecon/src/detectors/RPOTS/

General code for steering the pieces together.

RPOTS.cc
RomanPotsReconstruction_factory.cc
RomanPotsReconstruction_factory.h

Where the input collection comes in, and the hits are process through the reco algorithm, and then stored to a vector for analysis.

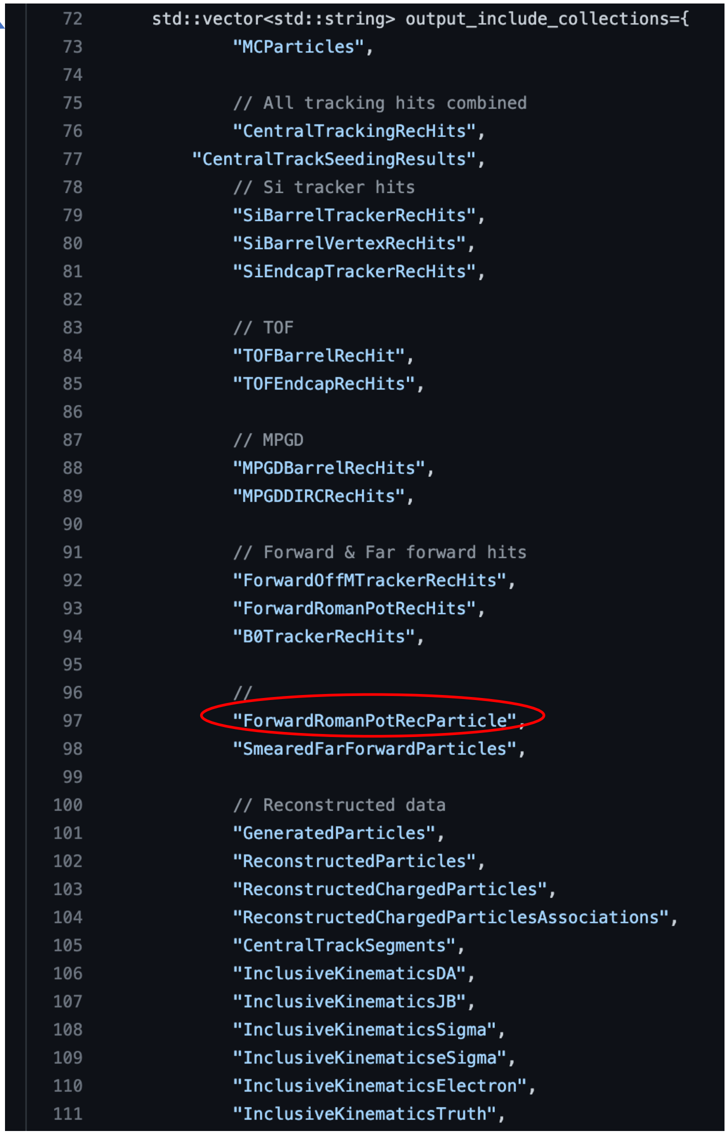
Notes:

- The "algorithm" is *supposed* to be called by the factory, but currently we cannot pass collections to algorithms, which does not work for us (we need a collection of hits to do the reco).
- The input hits are truly at generator level – they are not properly digitized (that's our next task ☺).
- The calculations are done in global coordinates → need to do things in local coordinate system (second task ☺).
- This is using the "static" matrix describing the 275 GeV proton orbit (really, describing all three main hadron orbits down to a few microns.
    - Will update with dynamic approach (third task).

Pitfalls

EICrecon/src/services/io/podio/JEventProcessorPODIO.cc

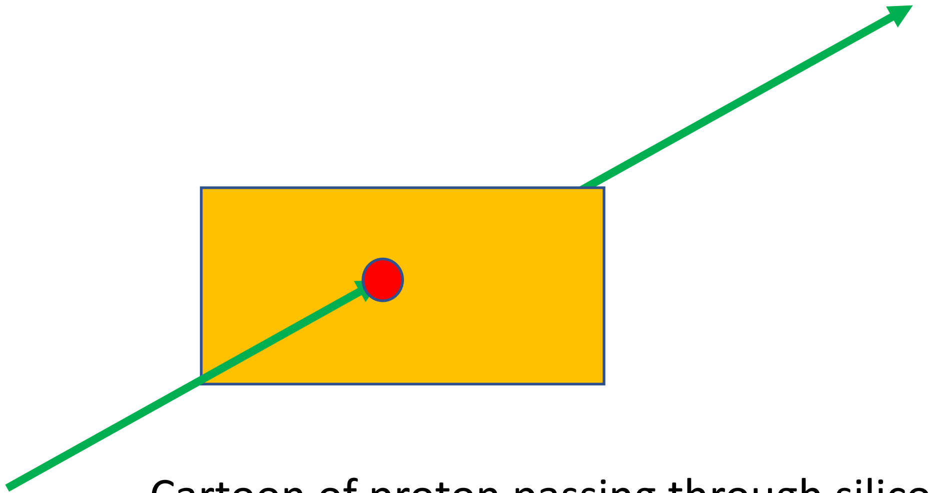
- When you make a new factory in EICrecon, the PODIO guys need to know you have a new ”collection” collection for them to handle.
  - You cannot just make a new factory without updating the PODIO code.
  - This is not remotely obvious, and I could not find it documented anywhere (thanks to Dmitry Romanov for helping me solve this).
- Your input collection also needs to be correctly associated to the “tag”.
  - “ForwardRomanPotHits” are “edm4hep::SimTrackerHit” objects.
  - These will need to be properly digitized as part of the next step.

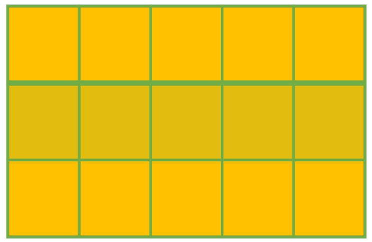


```
    std::vector<std::string> output_include_collections={
            "MCParticles",

            // All tracking hits combined
            "CentralTrackingRecHits",
        "CentralTrackSeedingResults",
            // Si tracker hits
            "SiBarrelTrackerRecHits",
            "SiBarrelVertexRecHits",
            "SiEndcapTrackerRecHits",

            // TOF
            "TOFBarrelRecHit",
            "TOFEndcapRecHits",

            // MPGD
            "MPGDBarrelRecHits",
            "MPGDDIRCRecHits",

            // Forward & Far forward hits
            "ForwardOffMTrackerRecHits",
            "ForwardRomanPotRecHits",
            "B0TrackerRecHits",

            //
            "ForwardRomanPotRecParticle",
            "SmearedFarForwardParticles",

            // Reconstructed data
            "GeneratedParticles",
            "ReconstructedParticles",
            "ReconstructedChargedParticles",
            "ReconstructedChargedParticlesAssociations",
            "CentralTrackSegments",
            "InclusiveKinematicsDA",
            "InclusiveKinematicsJB",
            "InclusiveKinematicsSigma",
            "InclusiveKinematicseSigma",
            "InclusiveKinematicsElectron",
            "InclusiveKinematicsTruth",
```

What do I mean by digitization?

- Digitization takes the information the GEANT produces, and turns it into a mimicked signal in you simulated detector.
- In DD4HEP, we draw a rectangle of silicon, make it “active”, and provide it with some segmentation (e.g. 500um pixels).
  - What this *means* is that DD4HEP takes your rectangle, and chops it up into 500um pixels “on paper”, and does nothing else with it.
  - Each pixel is simply assigned a “CellID”.
  - Our job is to take the hit information with the CellIDs and use that to properly account for the fact that we don’t know (in real life) *where* the hit occurred on the pixel.
  - We then make a new collection of hits reflecting that uncertainty.


Cartoon of proton passing through silicon plane, and depositing a bit of energy.



Next Steps

1) General cleanup of static reco code + conversion to local coordinates (easy task).
2) Digitization of hits, and definition of digitized hit collection (medium task).
3) In parallel, I will work to get the dynamic matrix code staged and figure out its implementation in the DD4HEP setup.
   - Need to investigate what linear algebra package we have in DD4HEP.
4) Begin looking at ML algorithms which match our needs, and discuss with software group.

Some additional resources:
https://indico.bnl.gov/event/18359/
https://indico.bnl.gov/event/18373/

Discussion