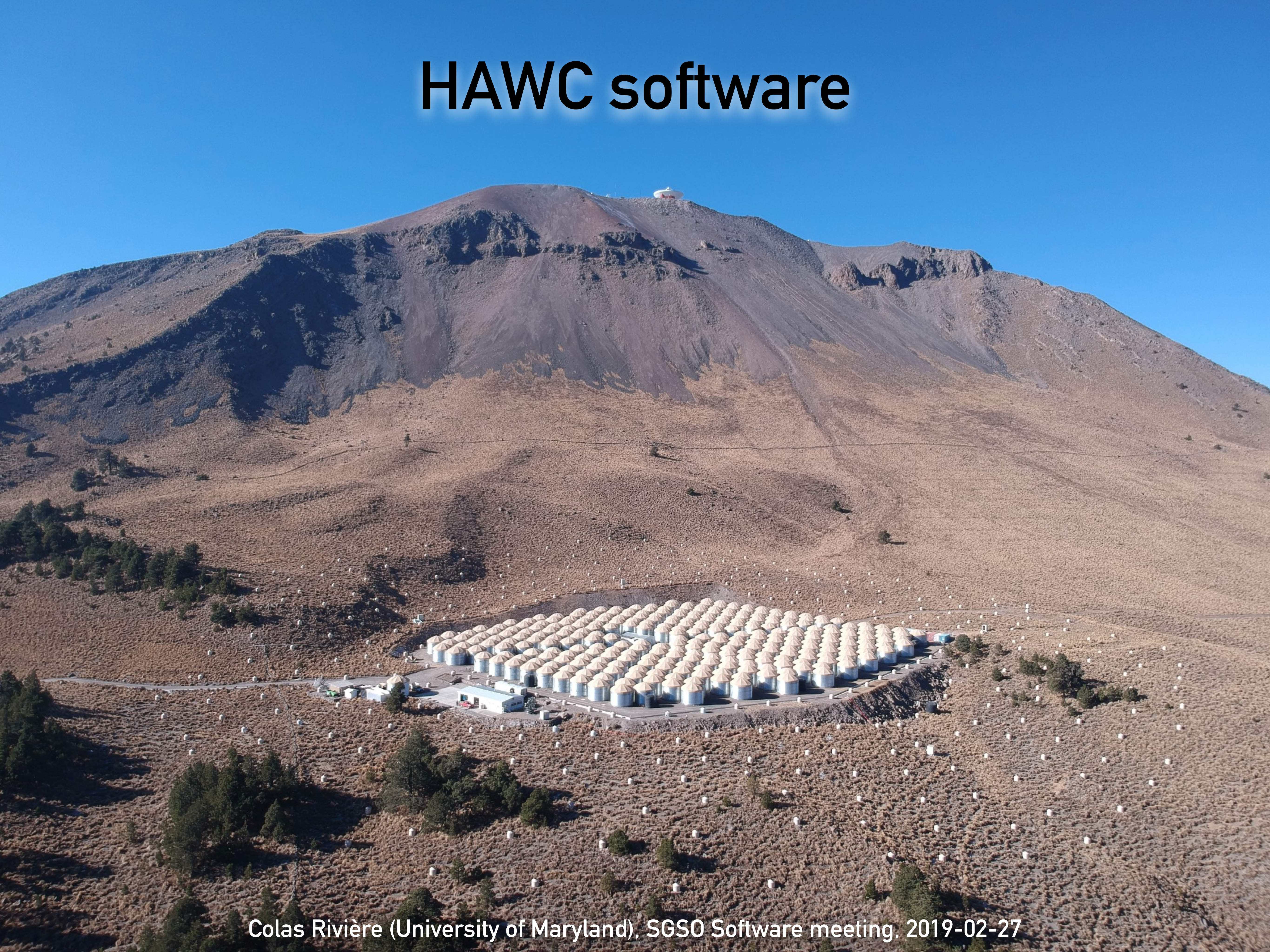
HAWC software



Colas Rivière (University of Maryland), SGSO Software meeting, 2019-02-27

What is HAWC software? (simplified)

- **AERIE**: *Analysis and Event Reconstruction Integrated Environment* (initially by Jim Braun, John Pretz, Segev BenZvi?), implemented in C++, used e.g. for
  - DAQ
  - Geant4 simulation
  - Online and offline event reconstructions
  - Simulation event weighting
  - Map making (event and background map)
- Data format: **XCDF**, *eXplicitly Compacted Data Format*, (initially by Jim Braun, Segev BenZvi) simple and efficient storage of data at desired precision, implemented in C++.
- Source analyses use **3ML**, *Multi Mission Maximum Likelihood* (initially by Giacomo Vianello, J. Michael Burgess) python framework.
- Package manager **APE**, *Auger Package Environment* (initially Lukas Nellen, Segev BenZvi)

What is HAWC software? (simplified)

- **AERIE**: *Analysis and Event Reconstruction Integrated Environment* (initially by Jim Braun, John Pretz, Segev BenZvi?), implemented in C++, used e.g. for
  - DAQ
  - Geant4 simulation
  - Online and offline event reconstructions
  - Simulation event weighting
  - Map making (event and background map)
- Data format: **XCDF**, *eXplicitly Compacted Data Format*, (initially by Jim Braun, Segev BenZvi) simple and efficient storage of data at desired precision, implemented in C++.
- Source analyses use **3ML**, *Multi Mission Maximum Likelihood* (initially by Giacomo Vianello, J. Michael Burgess) python framework.
- Package manager **APE**, *Auger Package Environment* (initially Lukas Nellen, Segev BenZvi)

- Useful to simulate future detector:
  - Simple point source sensitivity
  - More complex cases (extended sources, nearby sources, etc.)

AERIE intro

Introduction

The HAWC software, called the Analysis and Event Reconstruction Integrated Environment (AERIE) provides a framework intended for processing of HAWC events and for subsequent analysis.

The software is structured as a set of interdependent C++ projects glued together by a central core or “framework.” The core provides a run loop for analyzing batches of data, hooks for physicists to plug in their own algorithms, data classes to store simulated and reconstructed data, and libraries to handle common tasks such as geometry, astronomical coordinates, or time conversions. Other projects are provided to handle specialized tasks like disk I/O, track reconstruction, and map making.

The AERIE run loop can be driven with C++ “main” programs or python scripts. Templates for running popular applications can be found in the examples folders inside various projects.

Software Components

There are several major components to AERIE:

1. **The HAWCNest Framework** – A central object that registers and initializes services. It does not edit data.

2. **The Data Structures** – An in-memory representation of the data that can be edited by services.

3. **Services** – User code which can be used to edit data in a processing loop (“modules”) or provide stand-alone calculations, like random number generation or astronomical transformations.

4. **MainLoop** – A special service that defines the flow of control for data processing.

5. **Applications** – A suite of programs used for basic analysis of HAWC data, like map-making or estimation of energy spectra.

Beginning users will typically be most interested in applications. Intermediate users will be interested in data structures and data I/O. Advanced users write their own services and, for specialized tasks, edit the HAWCNest framework.

HAWC
High Altitude Water Cherenkov
Gamma-Ray Observatory

Table Of Contents

Previous topic

AERIE Offline/Online Software

Next topic

Downloading AERIE

This Page

Show Source

Quick search

Go

HAWCNest illustration

- Illustrated here for (a simplified view of) the offline reconstruction.
  - We do use all these algorithms, and more.
  - Modular: Can add/remove/swap services.
- Modules include:
  - DAQ simulation, to make MC events look like real data
  - Shower fitters (code, direction)
  - Gamma/Hadron separation
  - Energy estimators

offline-reconstructor-short.cc

```
HAWCNest nest;

// A few general services
nest.Service<StdRNGService>("random")
nest.Service<StdAstroService>("astroService"); // Astro transfo

// Readers
if (!isMC) { // reading data files
  nest.Service<ConfigDirDetectorService>("det")
  nest.Service<ChargeCalibrationService>("chargeCalibrator")
  nest.Service<TimingCalibrationService>("timingCalibrator")
  nest.Service<Reader>("reader")
  nest.Service<TriggeredInputSelector>("selector")
  nest.Service<Calibrator>("calibrator")
} else { // reading MC files
  nest.Service<StdDetectorService>("det")
  nest.Service<Reader>("reader")
  nest.Service<HAWCSimInputSelector>("eventSource")
  NestIniConfig(nest,mc_params); // DAQSim
}

// Reconstruction
nest.Service<SFCF>("coreFitGuess")
nest.Service<GaussPlaneFit>("planeFitGuess")
nest.Service<PropagationPlaneCut>("propagationPlaneCutGuess")
nest.Service<SFCF>("coreFit")
nest.Service<GaussPlaneFit>("gaussPlaneFit")
nest.Service<PINCCalculator>("pincCalc")
nest.Service<NeuralNetEnergyEstimator>("nnEneCalc")
nest.Service<LatDist>("LatDist")
nest.Service<ZenithAlignment>("zenithAlign")

// Writer
nest.Service<BinaryWriter>("writer")
nest.Service<DynamicSerializer>(serializerName)

// Configure and execute
nest.Service<SequentialMainLoop>("mainloop")
nest.Configure();
MainLoop& main = GetService<MainLoop>("mainloop");
main.Execute();
nest.Finish();
```

Example of reconstructed data format

- Compact version (used for production)

Rec data, MC truth, Weight, Comments

Field	Type	Resolution
rec.status	Unsigned Integer	1
rec.version	Unsigned Integer	1
rec.eventID	Unsigned Integer	1
rec.runID	Unsigned Integer	1
rec.timeSliceID	Unsigned Integer	1
rec.trigger_flags	Unsigned Integer	1
rec.event_flags	Unsigned Integer	1
rec.gtc_flags	Unsigned Integer	1
rec.gpsSec	Unsigned Integer	1
rec.gpsNanosec	Unsigned Integer	1
rec.nChTot	Unsigned Integer	1
rec.nChAvail	Unsigned Integer	1
rec.nHitTot	Unsigned Integer	1
rec.nHit	Unsigned Integer	1
rec.nHitSP10	Unsigned Integer	1
rec.nHitSP20	Unsigned Integer	1
rec.nTankTot	Unsigned Integer	1
rec.nTankAvail	Unsigned Integer	1
rec.nTankHitTot	Unsigned Integer	1
rec.nTankHit	Unsigned Integer	1
rec.windowHits	Unsigned Integer	1
rec.angleFitStatus	Unsigned Integer	1
rec.planeNDOF	Unsigned Integer	1
rec.SFCFNDOF	Unsigned Integer	1
rec.coreFitStatus	Unsigned Integer	1
rec.CxPE40PMT	Unsigned Integer	1
rec.CxPE40XnCh	Unsigned Integer	1
rec.coreFiduScale	Unsigned Integer	1
rec.LHLatDistFitNHitTanksMA	Unsigned Integer	1
rec.LHLatDistFitNHitTanksOR	Unsigned Integer	1
rec.LHLatDistFitNGoodTanksMA	Unsigned Integer	1
rec.LHLatDistFitNZeroTanksMA	Unsigned Integer	1
rec.LHLatDistFitNZeroTanksOR	Unsigned Integer	1
mc.corsikaParticleId	Unsigned Integer	1
mc.coreFiduScale	Unsigned Integer	1
mc.status	Unsigned Integer	1
mc.prescale	Signed Integer	1
sweets.oneWgt	Floating Point	0
sweets.IWgt	Floating Point	0
sweets.TWgt	Floating Point	0
sweets.BWgt	Floating Point	0
rec.logNNEnergyV2	Floating Point	0.001
rec.logGPV2	Floating Point	0.001
rec.zenithAngle	Floating Point	0.0001
rec.azimuthAngle	Floating Point	0.0001
rec.dec	Floating Point	0.0001
rec.ra	Floating Point	0.0001
rec.planeChi2	Floating Point	0.01
rec.coreX	Floating Point	0.1
rec.coreY	Floating Point	0.1
rec.logCoreAmplitude	Floating Point	0.1
rec.coreFitUnc	Floating Point	0.1
rec.SFCFChi2	Floating Point	0.01
rec.logNNEnergy	Floating Point	0.01
rec.fAnnulusCharge0	Floating Point	0.01
rec.fAnnulusCharge1	Floating Point	0.01
rec.fAnnulusCharge2	Floating Point	0.01
rec.fAnnulusCharge3	Floating Point	0.01
rec.fAnnulusCharge4	Floating Point	0.01
rec.fAnnulusCharge5	Floating Point	0.01
rec.fAnnulusCharge6	Floating Point	0.01
rec.fAnnulusCharge7	Floating Point	0.01
rec.fAnnulusCharge8	Floating Point	0.01
rec.protonlheEnergy	Floating Point	0.01
rec.protonlheLLH	Floating Point	0.01
rec.gammalheEnergy	Floating Point	0.01
rec.gammalheLLH	Floating Point	0.01
rec.chargeFiduScale50	Floating Point	0.01
rec.chargeFiduScale70	Floating Point	0.01
rec.chargeFiduScale90	Floating Point	0.01
rec.logMaxPE	Floating Point	0.01
rec.logNPE	Floating Point	0.01
rec.CxPE40	Floating Point	0.01
rec.CxPE40SPTime	Floating Point	0.1
rec.LDFAge	Floating Point	0.01
rec.LDFAmp	Floating Point	0.01
rec.LDFChi2	Floating Point	0.01
rec.GamCoreAge	Floating Point	0.01
rec.GamCoreAmp	Floating Point	0.01
rec.GamCoreChi2	Floating Point	0.01
rec.GamCorePackInt	Floating Point	0.01
rec.mPFnHits	Floating Point	0
rec.mPFnPlanes	Floating Point	0
rec.mPFp0nAssign	Floating Point	0
rec.mPFp0Weight	Floating Point	0.01
rec.mPFp0toangleFit	Floating Point	0.001
rec.mPFp1nAssign	Floating Point	0
rec.mPFp1Weight	Floating Point	0.01
rec.mPFp1toangleFit	Floating Point	0.001
rec.PINC	Floating Point	0.01
rec.disMax	Floating Point	0.01
rec.TankLHR	Floating Point	0.01
rec.LHLatDistFitXmax	Floating Point	0.01
rec.LHLatDistFitEnergy	Floating Point	1e-06
rec.LHLatDistFitMinLikelihood	Floating Point	0.01
rec.LHLatDistFitGoF	Floating Point	0.01
rec.LHXcog	Floating Point	0.01
rec.LHYcog	Floating Point	0.01
rec.LHLatDistFitZeroMinLikelihood	Floating Point	0.01
rec.LHLatDistFitHitMinLikelihood	Floating Point	0.01
mc.radiusWeight	Floating Point	1e-06
mc.eventWeight	Floating Point	0
mc.coreX	Floating Point	0.01
mc.coreY	Floating Point	0.01
mc.coreR	Floating Point	0.01
mc.zenithAngle	Floating Point	0.0001
mc.azimuthAngle	Floating Point	0.0001
mc.delCore	Floating Point	0.01
mc.delAngle	Floating Point	0.0001
mc.logEnergy	Floating Point	0.001
mc.logGroundEnergy	Floating Point	0.01
mc.Xmax	Floating Point	0.01

Entries: 6956565

```
Comments:
---------
XCDF version 3.0.1
EMinMC=5
EMaxMC=2e+06
ThetaMinMC=0
ThetaMaxMC=65
ThrowAreaMC=3.14159e+12
NEventsMC=100000
SpectralIndex=-2
PID=Proton
Jitter=0
TNoise=0
QErr=0
MinHits=0
prescale=0
Noise=0
! units rec.zenithAngle          : radian
! units rec.azimuthAngle         : radian
! units rec.dec                  : radian
! units rec.ra                   : radian
! units rec.coreX                : meter
! units rec.coreY                : meter
! units rec.coreFitUnc           : meter
! units rec.CxPE40SPTime         : nanosecon
! units mc.coreX                 : meter
! units mc.coreY                 : meter
! units mc.coreR                 : meter
! units mc.zenithAngle           : radian
! units mc.azimuthAngle          : radian
! units mc.delCore               : meter
! units mc.delAngle              : radian
```

- Extended version contains this plus vectors of hits and a lot more info.

Simulation intro

HAWC Simulation

The HAWC simulation occurs in three stages:

1. **CORSIKA** simulations of cosmic rays and gamma rays.
2. **HAWCSim** GEANT4-based simulations of the response of the tanks to CORSIKA particles at ground level.
3. **AERIE Reconstruction** of simulated events which produce signals in the HAWC tanks.

A library of showers produced with CORSIKA, HAWCSim, and AERIE is maintained at UMD. The simulation is being continuously generated and updated, and the best place to track the current list of files you should use for analysis is given in the Monte Carlo Products wiki page.

CORSIKA Air Showers

CORSIKA is a simulation code that tracks the nuclear and electromagnetic interactions which occur in extensive air showers. In HAWC we generate air showers produced by gamma rays, protons, and the heavier nuclei ^{4}He, ^{12}C, ^{16}O, ^{20}Ne, ^{24}Mg, ^{28}Si, and ^{56}Fe.

CORSIKA data at UMD can be found in the directory

```
$HAWCROOT/sim/corsika
```

The details of the directory structure are described here.

The files are in a binary format which can be read using the code and scripts in the AERIE I/O project.

HAWCSim Tank Simulations

hawcsim is a part of AERIE that is built if the particle tracking code GEANT4 has been detected on your system. (GEANT4 and its associated interaction tables can be installed with `ape`; see Building with APE.)

HAWCSim will propagate particles at ground level from CORSIKA into a model of the HAWC tanks, calculate the Cherenkov photons produced when the particles enter the water in the tanks, and convert

HAWC
High Altitude Water Cherenkov
Gamma-Ray Observatory

Table Of Contents

Previous topic

Next topic

This Page

Show Source

Quick search

Go

Analysis workflow - Data, I/II

- Experimental data:
  - 300 large WCDs (7m diameter, 4.5m tall, 200 m^3 water):
    - 3x8" PMTs (24 kHz rate @ ~0.3 PE)
    - 1x10" high QE PMT (40 kHz rate @ ~0.3 PE)
  - 345 small WCDs (1.55m diameter, 1.4m tall, 2.5 m^3 water)
    - 1x8" PMTs (6 kHz rate @ ~? PE)
- Main array trigger:
  - ~30 hits within 500 ns window.
  - When trigger, record all hits within 2.4 us window (time, ToT)
  - => 2 TB/day.
- Reconstruction:
  - Calibration
  - Hit selection (remove ambiguous hits, afterpulse, etc.)
  - Core fit
  - Angle fit (need to know the core for shower curvature)
  - Gamma/Hadron separation variables
  - Energy estimators
  - Choice of minimal output or extended output (with every hit)

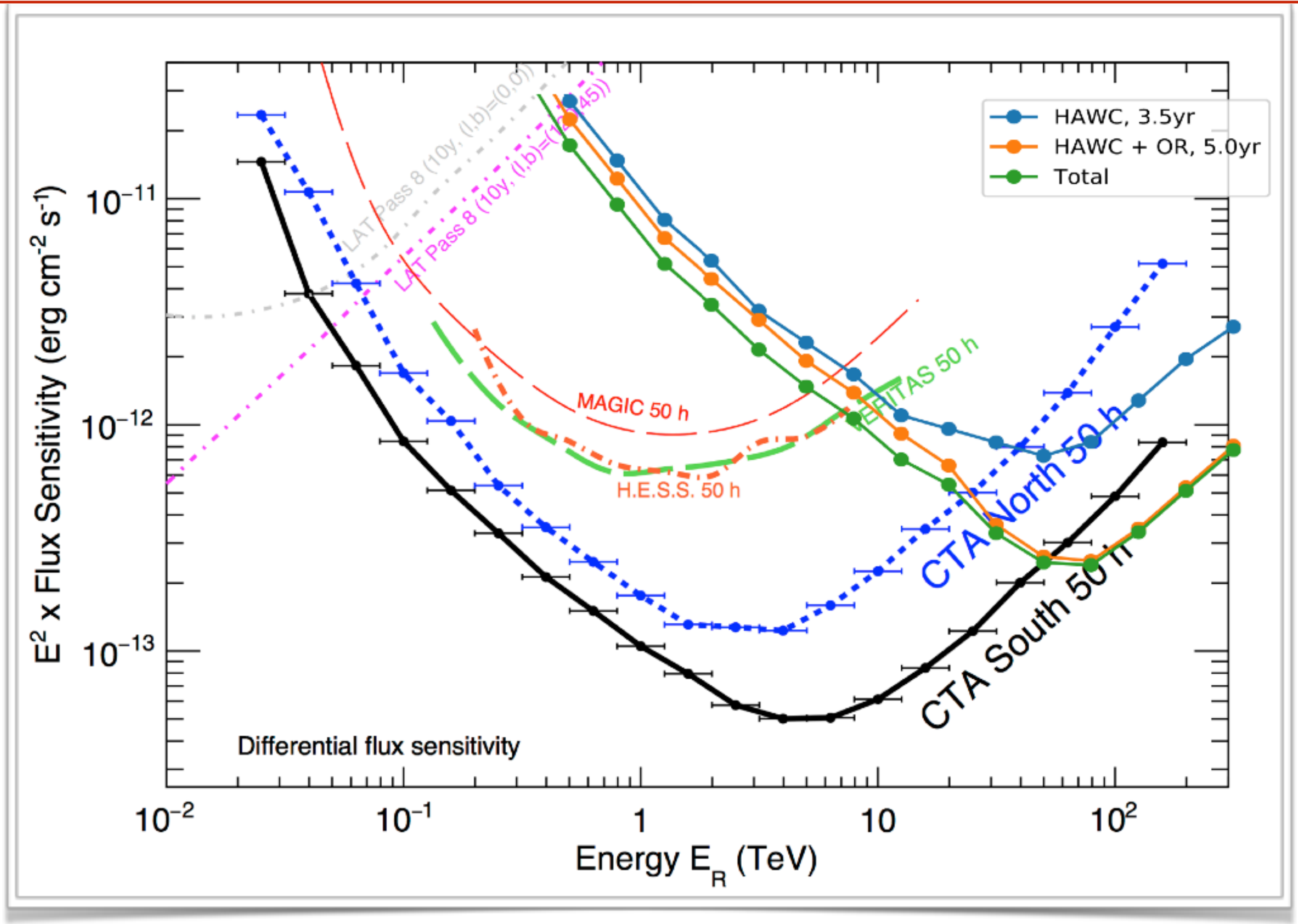
Analysis workflow - Data, II/II

- Make event and background maps
  - Classify reconstructed events in analysis bins (event size, energy, apply Gamma/Hadron cut). 9 for published analyses, ~30 for incoming publications (add energy estimators).
  - Using direct integration, method from Milagro days, for precise background estimate in high background bins.
- Likelihood source analysis:
  - Within aerie for (all sky) maps.
  - Moved to python framework (Multi Mission Maximum Likelihood, 3ML) for source analyses.

Analysis workflow - Simulation

- Corsika shower. Save particles at 4100m asl.
- HAWCSim (based on Geant4)
  - Inject particles right above HAWC (10 or 50m)
  - Propagation and light emission with Geant4 (with all WCD physics)
  - Record PEs hitting the PMTs (energy, position)
- Reconstruction:
  - DAQSim: Modular DAQ simulation package, with simulation of PMT response, electronics, calibration, channel status, noise model, etc.
  - Then the same reconstruction as for data.
- Weighting (SWEETS: *Software for Weighting Events and Event-like Things and Stuff*):
  - Remove simulation bias (injection spectrum, zenith distribution, etc.)
  - Weight for template source (whole source transit a declination, or burst at a zenith) for a spectrum. Or weight for isotropic hadrons (using Cream measurements).
- Detector response:
  - Event rate for reference source, as a function of energy.
  - Background rate (but we usually use data for that)
  - PSF, etc.
- Sensitivity study:
  - Use 3ML to add sources on top of background maps (from data) and estimate sensitivity. Can re-weight to arbitrary spectrum, model point or extended sources.

Sensitivity



- Ingredients:
  - Source characteristics: spectrum, declination, morphology.
  - Reconstructed gamma ray characteristics: selection cuts, PSFs, etc.
  - Hadronic background rate. Can be simulated, or inferred from data.
- For a start, $\sigma = S/\sqrt{B}$ in an optimal round bin can work. Then combine several analysis bins. Other extreme if needed, full PSF information, multiple extended sources and Poisson likelihood can be used (3ML). Or something in between.
- We need to make/share the tools to reproduce HAWC sensitivity easily (not just for SGSO).

Want to simulate another detector?

- (Corsika output another altitude. Also different geomagnetic field?)
- HAWCSim:
  - Add the Geant4 detector unit definition to aerie/trunk/src/hawcsim/src/Tank.cc (was done recently for outriggers). Define a new detector unit type (now: `enum TankType {MainTank=1, OutriggerTank=2};`).
  - Update the XML survey file (position and type of detectors)
- Reconstruction:
  - If you *replace* HAWC's WCD by another detector units, changes can be minimal. If you want multiple stream of hits (like we did for adding outriggers), need a bit more changes. Can look at the corresponding commits.
  - Can keep most algorithms, re-tune a few.

Where things are

- Currently:
  - Codebase: private, svn, self hosted.
  - Documentation: private, self hosted, wiki + document database
  - Data:
    - Private, computing clusters (University of Maryland, Universidad Nacional Autónoma de México).
    - Some public data, for published results. Plans to release more.

- Plan:
  - Move to Git soon (weeks).
    - Testing workflows with GitHub and GitLab now. The latter corresponds better to what we are looking for, but more risky?
  - Make AERIE public (months?). Probably write a paper like Auger's offline paper. Share the code privately before that?
  - Document.
  - What else would be useful? Corsika? Background maps? Internal notes?

Documentation

- Some documentation:
  - Sphinx, auto-generated, currently hosted internally. Should host it publicly (e.g. readthedocs).
  - Internal notes (e.g. event weighting, map-making, DAQ simulation).
- As always, there's room for improvement. Needed:
  - End to end analysis. From Corsika to sensitivity computation. Enough to reproduce HAWC's sensitivity and branch off of it.
- People will probably be available for help (e.g. Slack), probably easier/faster than waiting for full documentation.

- I'll put some of it in the next slides, but there are holes.

Dependencies

- Required: boost root xerces xcdf healpix-cxx photospline fftw.
- Optional: **geant4** mysql zeromq cppzmq.
- We typically use the Auger Package Environment to install them. Requests password for downloading dependencies, but can be lifted (all are open source except AERIE).
  - A bit painful to install Geant4 right now, but should fix it soon.
  - Should we try conda?

Summary

- AERIE is a well though framework.
  - Modular, should be easy to adapt to another detector
  - We are adding outriggers, so this is an example of "different detector" already
  - Various modules available
- We are working on making it public